Ngmath Library Overview

What the ngmath library is

The ngmath library is a collection of mathematical procedures for which there are equivalent Fortran, C, and NCL entries. These procedures are generally not available in other libraries local to the NCAR Scientific Computing Division.

What the ngmath library contains

As of January, 1999 the ngmath library contains a collection of interpolators and approximators for one-dimensional, two-dimensional, and three-dimensional data. The packages are:

- natgrid -- a two-dimensional random data interpolation package based on Dave Watson's package nngridr.
- dsgrid -- a three-dimensional random data interpolator based on a simple inverse distance weighting algorithm.
- fitgrid -- an interpolation package for one-dimensional and two-dimensional gridded data based on Alan Cline's package Fitpack, an early version of which is available from netlib. Fitpack uses splines under tension to interpolate in one and two dimensions.
- csagrid -- an approximation package for one-dimensional, two-dimensional, and three-dimensional random data based on David Fulker's package Splpack. Csagrid uses cubic splines to calculate its approximation function.

General background on interpolation and approximation methods

Computational interpolation and approximation methods can be divided into two basic classes: fitted function methods and weighted average methods.

The fitted function methods fit an algebraic surface to the known data and then pick the interpolated values from the fitted surface. The weighted average methods calculate interpolated values as weighted averages of known values.

Among the weighted average methods, the simplest are known as the inverse distance weighted average methods. These methods are implemented by calculating weights for all input coordinates such that the weights are inversely proportional to the distance an input coordinate is from the point where an interpolated value is desired. The sum of the weights is normalized to unity and an interpolated value is the sum of the products of the functional values and the calculated weights.

Comparative analysis of the ngmath packages

Three-dimensional problems

Only Dsgrid and Csagrid handle three-dimensional data. Csagrid will in general produce smoother results than Dsgrid (compare the plot for Csagrid example 5 to the plot for Dsgrid example 1).

Since Csagrid uses an approximation algorithm, the calculated surface will not necessarily pass through the original data points. Also, the Csagrid algorithm can become unstable in data sparse regions. The weighted average algorithm used in Dsgrid is stable in all cases and the resulting surface will pass through the original data values.

Two-dimensional problems

The four packages natgrid, csagrid, dsgrid, and fitgrid have entries for two-dimensional data.

In most all cases natgrid will be the package of choice. It implements a very stable algorithm and has parameters for adjusting the smoothness of the output surface.

The two-dimensional surface entries in Fitgrid offer user-settable parameters for specifying derivatives along the boundary of the output grid that are not available in Natgrid.

Csagrid will produce a two-dimensional surface that is an approximation and may produce results that are smoother in appearance than those produced by Fitgrid or Natgrid.

Dsgrid is not recommended for two-dimensional surfaces - Natgrid is superior in all respects.

One-dimensional problems

Only Fitgrid and Csagrid handle one-dimensional data and Fitgrid is definitely the package of choice. Fitgrid has an entry for producing an approximation that is essentially the same as that produced by Csagrid, but Fitgrid has many features not available in Csagrid, such as interpolating parametric curves, finding integrals, handling periodic functions, allowing for smoothing that varies from linear interpolation to a full cubic spline, and specifying slopes at the end points.

Accessing the ngmath library

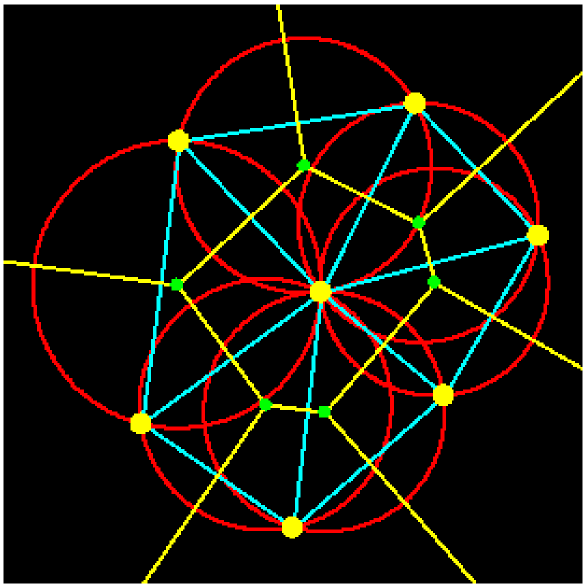
The ngmath library is a stand-alone library. Using the Fortran and C entries in the library in no way depends upon using NCAR Graphics. However, the library is distributed only with NCAR Graphics. The NCL interface to the ngmath entries is built into the NCL executable.

If you have NCAR Graphics installed at your site, then the ngmath library will be located in $NCARG_ROOT/lib/libngmath.a . If you want to load the library directly, check your environment variable settings to see if you have $NCARG_ROOT set and, if not, consult your local system administrator to see what you should set it to. If you are using NCAR Graphics, then, for Fortran and C codes, by adding a "-l ngmath" flag to "ncargf77" or "ncargcc" the ngmath library will be loaded.

Future plans for the ngmath library

The next package to be added is FFTPACK, a locally-developed library of one-dimensional FFT solvers, including complex, real, sin, cosine, quarter-sin, and quarter-cosine transforms. Also planned is a package for drawing Bezier curves and a random data two-dimensional interpolator that allows for constrained regions with holes. Also, NCL interfaces to commercial libraries such as Linpack and IMSL will be provided as time permits.

Natgrid, a natural neighbor gridding package (based on Dave Watson's nngridr)



Introduction (read this first)

Learning how to use the package

Natgrid Table of contents

Credits

Introduction

Learning how to use Natgrid

Natural Neighbor Linear Interpolation

Natural neighbor nonlinear interpolation

Scaling the input coordinates

Extrapolation

Efficiency considerations

Examples

References

Index

Natgrid Introduction

About Natgrid

Natgrid is a package in the ngmath library of Fortran subroutines and C functions that implements a natural neighbor interpolation method. The input for the interpolation is a set of randomly-spaced two-dimensional coordinates with functional values at those coordinates; the output is a set of interpolated function values at coordinates in a user-specified rectangular grid. The coordinates in the output grid must be monotone increasing in each coordinate direction, but need not be uniformly spaced. It is also possible to interpolate at a single point. See the comparative analysis of the ngmath packages for details on which ngmath package is most appropriate for your problem.

Functionally equivalent Fortran and C interfaces are provided in both single and double precision versions.

Natgrid is based on the work of Dr. Dave Watson and his *nngridr* package. Dr. Watson is well known in the field of interpolation. He has published many articles and two books in the area. See the references module for further information. Dr. Watson has our greatest gratitude for allowing us to incorporate segments of his code in our package. Information on how to obtain Dr. Watson's original *nngridr* code, as well as his accompanying book, can be obtained from URL
http://www.iinet.com.au/~watson/nngridr.html

General background on interpolation methods

Computational interpolation methods can be divided into two basic classes: fitted function methods and weighted average methods.

The fitted function methods fit an algebraic surface to the known data and then pick the interpolated values from the fitted surface. The weighted average methods calculate interpolated values as weighted averages of known values.

Among the weighted average methods, the simplest are known as the inverse distance weighted average methods. These methods are implemented by calculating weights for all input coordinates such that the weights are inversely proportional to the distance an input coordinate is from the point where an interpolated value is desired. The sum of the weights is normalized to unity and an interpolated value is the sum of the products of the functional values and the calculated weights.

The interpolation method used in Natgrid

Natgrid uses a weighted average method that is much more sophisticated than the inverse distance weighted average methods mentioned above. The method used in Natgrid is known as natural neighbor interpolation. Natural neighbor interpolation was introduced by R. Sibson in the early 1980s.

One distinguishing quality of natural neighbor interpolation is the way in which a set of neighboring points (the *natural neighbors*) is selected to use for interpolating at a given point. The natural neighbor selection process avoids the problems common to methods based on choosing a fixed number of neighboring points, or all points within a fixed distance.

Another distinguishing quality of natural neighbor interpolation is the way that the weights are calculated for the functional values at the natural neighbor coordinates. These weights are based on proportionate areas, rather than distances.

The method of finding the natural neighbors and calculating areas-based weights to produce interpolated values is called natural neighbor linear interpolation. This produces an interpolated surface that has a continuous slope at all points, except at the original input points. The result of natural neighbor linear interpolation can be visualized as producing a snugly fit sheet stretched over all of the input data.

The interpolation method in Natgrid also allows for natural neighbor linear interpolation augmented by blending in gradient estimates. This latter is referred to as natural neighbor nonlinear interpolation. Natural neighbor nonlinear interpolation produces an interpolated surface that has a continuous slope at all locations; two tautness parameters can be set by the user to control the apparent smoothness of the output surface.

If you are interested in the details of the interpolation method used in Natgrid, you will want to buy Dr. Watson's book mentioned above.

Learning how to use Natgrid

You should now proceed to the module on learning how to use Natgrid.

Learning how to use Natgrid

The programmatic interfaces

There are four interfaces to choose from: Fortran or C in either double precision or single precision. Each interface has one procedure for doing the gridding. Without worrying about understanding exactly what all the procedures do, browse the table of procedure names at this time. Notice that details on any of the procedures can be obtained by clicking on the procedure name.

A program using Natgrid will contain a section that sets values for appropriate control parameters, after which one of the gridding procedures will be invoked. In the case of using natural neighbor linear interpolation, you may not have to set values for any control parameters before invoking the gridding procedure.

To get a feeling for the available control parameters (again without worrying about understanding exactly what all of the parameters are), examine the table of control parameters at this time.

Complete details on all procedures and control parameters are presented in other modules.

Examples are provided illustrating the functionality of Natgrid. All of the examples are in the Examples module. Source codes and example plots are presented there, as well as instructions on how to create your own executables. Browse the introductory remarks in the Examples module now.

Learning guide

First, to get a feel for what is in the package, quickly browse the table of contents.

Reading through the first two modules in the bulleted list below and examining the examples cited there should serve as a quick start guide. Then, progressing through the remaining items in the list should give you a thorough understanding of Natgrid.

- Natural neighbor linear interpolation
- Natural neighbor nonlinear interpolation
- Scaling the input coordinates
- Extrapolation
- Efficiency considerations
- Some specialized control parameters:
  - Negative values
  - Vertical orientation
  - Duplicate input coordinates
- Computing aspects and slopes
- Interpolating at a single point

- Outlying points and roughness estimates
- Error handling
- Algorithmic display tool

Natgrid credits

Acknowledgments

Author: Fred Clare

Editor: Fred Clare. Final copyedit by Brian Bevirt.

Technical Reviewers: Tom Parker

Production: Fred Clare

Special thanks: Dr. Dave Watson, who allowed us to base this package on the prodigious work he did in producing his "nngridr" package. There would be no Natgrid if it were not for Dr. Watson's package and his permission to include parts of his work here.

Also, thanks to Dick Valent who told the author of the existence of *nngridr*.

Copyright

Trademark

Natural Neighbor Linear Interpolation

Simplest usage

The easiest way to use Natgrid is to do natural neighbor linear interpolation. For a discussion of natural neighbor linear interpolation, review the section in the introduction on the interpolation method used in Natgrid.

If the default settings for the control parameters are what you want, then you can do natural neighbor linear interpolation by invoking a single procedure. Example 1 presents a simple natural neighbor linear interpolation; it provides codes using all four of the user interfaces.

Smoothness and efficiency considerations

If you have a relatively large number of input data points (like a few hundred or more), then natural neighbor linear interpolation may produce a satisfactorily smooth surface for you. Doing natural neighbor linear interpolation is much less expensive than augmenting it, for appearance, by blending in gradient estimates (as described in the module Natural neighbor non-linear interpolation). See the module on efficiency considerations for details on efficiency.

Natural neighbor nonlinear interpolation

Producing smooth plots

If you want an interpolated surface that has continuous partial derivatives at all points (i.e. a smooth surface), then you will want to use natural neighbor non-linear interpolation. The control parameter igr governs whether Natgrid does linear or non-linear interpolation. If igr is set to 1, then, for a given interpolation point, gradients are estimated at all natural neighbors of this point, and these estimates are blended in to reflect nearby trends.

Examine the code for Example 2. Notice that the only difference between this example and Example 1 is the call to set igr to indicate that gradient estimates are to be used. Compare the plot from example 1 with the plot from example 2. The plot from example 2 has continuous partial derivatives, even at the original data values.

For further background on non-linear natural neighbor interpolation, review the section in the introduction on the interpolation method used in Natgrid. Complete details are in Dr. Watson's book nngridr.

Controlling the influence of the estimated gradients

There are two parameters, called *tautness parameters*, that can be set to control the influence of the gradient estimates on interpolated values. These parameters are bI and bJ. Increasing the value of bJ decreases the breadth of the region affected by a gradient estimate and increasing the value of bI increases the intensity of the influence of a gradient estimate. A value for bI or bJ can be anything greater than or equal to "1.". The effective upper range for each parameter is about 10., although larger values are allowed.

Increasing bI to values larger than the default value of 1.5 may produce an underflow, since bI is ultimately used in a negative exponent of an exponential function. If an underflow occurs, you need not be concerned about it.

The default values for bI and bJ are chosen to produce a relatively smooth plot for most data sets. You may want to modify the values of bI and bJ to see what effect this has on your interpolations. In general, the effects of changing bJ are fairly subtle. Setting bI and bJ both to "1." essentially produces natural neighbor linear interpolation; increasing their values produces a smoother plot.

Scaling the input coordinates

Scale requirements for the input data

For reasons of accuracy, Natgrid requires that the input data values not be terribly out of scale. In particular, where:

```
XB = the breadth of the X coordinate values (maximum X - minimum X)
YB = the breadth of the Y coordinate values (maximum Y - minimum Y)
ZB = the breadth of the Z coordinate values (maximum Z - minimum Z)
```

the ratio of XB to YB should lie between 1/2 and 2; the ratio of ZB to XB should be between 1/60 and 60; the ratio of ZB to YB should be between 1/60 and 60.

If you are computing aspects or slopes, then the above ratios are a requirement, otherwise see below.

Automatic scaling

If you are not computing aspects or slopes, then Natgrid provides an automatic scaling feature that adjusts the input values appropriately before any calculations are done, and readjusts the values appropriately on output. Automatic scaling is the default state.

Whether automatic scaling is in effect is governed by the control parameter asc. If you are computing aspects or slopes and your input data do not adhere to the ratios mentioned above, then, to avoid getting an error message, you will need to turn automatic scaling off and scale your data by setting one or more of the scaling control parameters magx, magy, and magz. For example, setting magx to 2. will scale all X coordinate values by 2. before calculations begin. Setting the value for magx in such a manner is equivalent to your having multiplied all X coordinate values by 2. before doing the gridding. Of course all output values will reflect any scaling values you set when automatic scaling has been turned off.

There are three parameters that can be interrogated to determine what automatic scaling values were in effect during the last call to one of the primary interpolation procedures NATGRIDS, NATGRIDD, c_natgrids, or c_natgridd. These parameters are xas, yas, zas. When any of the primary interpolation procedures is called, each of xas, yas, zas is initialized to 1., then it is determined if scaling is necessary, and if it is, then xas, yas, and zas are automatically set to appropriate values. It is these values that can be interrogated.

Extrapolation

Reliability of extrapolation

In general, extrapolation is not very reliable and the results so obtained are to be viewed with some lack of confidence. In order for extrapolation to be at all reliable, the original data must be very consistent.

How Natgrid does extrapolation

Natgrid does extrapolation by least squares fitting a plane to the original data and using that plane to produce pseudo data values at some distance from the convex hull. These pseudo data points are added to the input data points when natural neighbors are calculated. The result of this is that extrapolated data will asymptote to the least squares fitted plane.

Controlling whether extrapolation is done

Natgrid allows you to turn off extrapolation and set the interpolated functional values to a constant outside of the convex hull. If the control parameter ext is set to 1 (the default), extrapolation is allowed; if ext is set to 0, extrapolation is not allowed.

The parameter nul specifies the value to be assigned to the interpolated function outside of the convex hull when extrapolation is not allowed; by default nul is 0. .

Examples

Example 4 uses the same function as Example 3, but expands the output region to allow for extrapolation. As provided, the code for Example 4 allows for extrapolation. For an example that disallows extrapolation, change the setting of ext in the code for Example 4 from 1 to 0. Notice that in the plot that allows extrapolation, the function values tend toward the least squares fitted plane outside of the convex hull. The plot where extrapolation is not allowed uses the same function and illustrates turning extrapolation off and setting the function values to -1.0 outside of the convex hull. The only difference between the code that produced these plots is the setting of the value of the ext parameter.

Natgrid efficiency considerations

Main time components

The time required for executing Natgrid codes is primarily dependent on three things:

1. The time required to triangulate the input data and determine all natural neighbors.
2. The number of output points where interpolated values are desired.
3. Whether linear or nonlinear interpolation is used.

Detailed timing results

A table summarizing a timing analysis done on a Cray Y-MP is provided if you want details. The basic conclusions are:

- For a fixed number of input and output data, linear interpolation is, in general, over twice as fast as nonlinear interpolation.
- For a fixed number of output points, the timings for linear interpolation vary as the square of the number of input points.
- For a fixed number of input points, the efficiency of non-linear interpolation increases as the number of output points is increased (based on the time-per-point calculations).
- Interpolating at a single point is very expensive, especially if non-linear interpolation is being used.

General conclusions

Using Natgrid is expensive.

If you can get satisfactory results using linear interpolation, then that should be used. If you have a large number of input data that are randomly spaced, then linear interpolation will not differ too much from nonlinear.

Single point mode is very expensive and, in general, should be used only for filling in a few missing data values.

When time is a major concern

If execution time is a major consideration, then Natgrid allows for significant speed-up using a process called *subblocking*. The main idea is to split up the output array into subblocks, do the interpolation on each individual subblock, and then composite the results. Unfortunately, this process takes some work on the part of the Natgrid user.

For example, suppose that you are interpolating on a 100 x 100 output grid. You could split this into four separate interpolations of 50 x 50 (or one hundred of 10 x 10, and so forth). Since the time increases basically as a function of the square of the number of input points, much is saved by splitting the original problem.

Beyond the care needed to make sure that the individual interpolations are correctly stitched back together, there are constraints imposed by Natgrid to insure that the interpolated values along adjacent subblock edges are compatible. These constraints are:

1. *For linear interpolation*: if an input coordinate (X,Y) is contained in a subblock, then every

natural neighbor of (X,Y) must be included in calculating interpolated values.

2. *For nonlinear interpolation*: if an input coordinate (X,Y) is contained in a subblock, then every natural neighbor of (X,Y) must be included in calculating interpolated values *and* every natural neighbor of every natural neighbor of (X,Y) must also be included.

The set of points in the input dataset included in a calculation of interpolated values in a subblock is controlled by the *subblock overlap parameters* hor and ver. These parameters determine additional space around the interpolated grid that will include additional input data. For example, if your output grid for a subblock were the lower left quarter of the unit square and you chose hor=0.1 and ver=0.05, then all input points in the region bounded by X = -0.1 on the left, X = 0.60 on the right, Y = -0.05 on the bottom, and Y = 0.55 on the top would be included in the calculation.

What is a good way to determine values for *hor* and *ver*? One way would be to make a test run using a sparse output grid and then use the features of the algorithmic display tool for displaying natural neighbors; another way would be to make an educated guess based on your knowledge of what a natural neighbor is and what your data look like.

Some specialized Natgrid control parameters

This module discusses the control parameters for:

Negative values
Vertical orientation
Duplicate input coordinates

Negative values

The control parameter non determines whether negative values are allowed as interpolated values. The default is to allow them. If disallowed, by setting the value of the control parameter non to 1, then any value that would otherwise be negative is set to zero. The plots in Example 5 illustrate this. Compare the plot that allows negative values with the plot that does not allow them.

This feature is useful if you have a function, such a pressure, that should never have a negative value.

Vertical orientation

The control parameter upd determines the ordering of the second dimension of the output array. The default, "upd = 1", implies a *south to north* ordering; setting upd to 0 reverses the order.

Duplicate input coordinates

The control parameter dup determines whether duplicate input coordinates are allowed. By default they are not allowed, the assumption being that such points might represent a flaw in the input data. If duplicate input coordinates are allowed and different data values are assigned for these points, these values will be interpreted as different data values that are very closely spaced, since Natgrid adds a small random perturbation to each input coordinate so that all input coordinates are unique. See Dave Watson's book, *nngridr*, for details on this.

Computing aspects and slopes

On occasion it is useful to examine information on aspects and slopes. Calculating aspects and slopes is subject to the restrictions mentioned in the module on scaling input coordinates.

Whether or not aspects and slopes are calculated is governed by the control parameter sdi. By default, slopes and aspects are not calculated.

If slopes and aspects are calculated, then Natgrid allocates memory for storing this information internally so that it will be available to the procedures that retrieve aspect and slope values (NNGETASPECTS, NNGETSLOPES, NNGETASPECTD, NNGETSLOPED, c_nngetaspects, c_nngetslopes, c_nngetaspectd, and c_nngetsloped). The memory for the slope and aspect data is reallocated whenever one of the gridding procedures (NATGRIDS, NATGRIDD, c_natgrids, or c_natgridd) is called and sdi is set to indicate aspects and slopes are to be calculated.

The control parameter rad determines whether aspect and slope values are returned in degrees or radians. The default is in degrees.

Example 3 illustrates computing aspect and slopes. A vector plot of the aspects and a contour plot of the slopes are supplied in the example.

Interpolating at a single point

Slowness

Natgrid provides for interpolation at a single point but, as indicated and documented in the efficiency module, interpolation at single points is very expensive and, in general, should be used only for filling in a few missing data values.

How to do it

For the same input dataset, in order to avoid having to calculate the natural neighbor relationships every time interpolation is done on a single point, interpolation at single points is implemented as a three step process:

1. Invoke an initialization procedure that calculates the natural neighbor relationships. This procedure is NNPNTINITS in Fortran and c_nnpntinits in C.

2. Do the interpolation at the desired points. The procedure for this is NNPNTS in Fortran and c_nnpnts in C.

3. Terminate single point mode. This is done using the procedure NNPNTEND in Fortran and c_nnpntend in C.

You can set any control parameters you want before interpolating at individual points, except slopes and aspects cannot be calculated while in single point mode.

See Example 6 for Fortran and C examples of interpolation in single point mode.

Due to implementation details, the values calculated in single point mode may differ slightly from values at the same coordinates when calculated using a grid.

Outlying points and roughness estimates

Natgrid has facilities for special treatment of points near function values where the function values are not consistent with neighboring values. For each input point an *outlier index* is calculated if the point is not on the convex hull; for points on the convex hull, a *roughness index* is calculated. Both of these indices are a normalized measure of the consistency of an input function value with its neighbors. For details on how these indices are defined and calculated, see Dave Watson's book.

The effects of using the outlier and roughness indices are primarily to attenuate the influence of gradient estimates near points with high values for either of the two indices. This means that in the neighborhood of such points, the interpolation tends more toward natural neighbor linear interpolation. This prevents oversmoothing in those areas, and oversmoothing could result in incorrect deductions being made from analysis of the resulting surface. Exaggerated oscillations surrounding outlying points, that are produced by some curve fitting techniques, are also minimized. See Example 7.

Natgrid error handling

The error handling functions

Natgrid contains a set of procedures that implement a centralized error handling capability. The package invokes the C function ErrorHnd to produce all of its error messages. ErrorHnd simply invokes the ErrorLog function with the arguments specified in ErrorHnd. This allows a user to replace the error handling functionality by replacing or modifying the ErrorHnd function with one that handles errors in an application-dependent manner. All of the error handling functions are written in C; there is no way to replace the error handler using Fortran code.

Detailed description of the error functions

The error functions are:

- ErrorHnd - primary error handler
- ErrorLog - error logger
- ErrMsg - retrieves error messages
- ErrMax - retrieves the number of error messages

ErrorHnd - Primary error handler

ErrorHnd simply invokes the ErrorLog function with the arguments specified in ErrorHnd. This allows a user to replace the error handling functionality by replacing or modifying the ErrorHnd function with one that handles errors in an application-dependent manner.

Function prototype:

```
void ErrorHnd (int, char *, FILE *, char *);
```

Argument description:

```
---------------------------------
              Argument | Type
---------------------------------
void ErrorHnd (error,  | int
               func,   | char *
               efile,  | FILE *
               smsg);  | char *
---------------------------------
```

error
: The error number, as described in the error table.

func
: A string specifying the function that is invoking the error call.

efile
: Specifies the file where the error message will be written. By default, error messages are written to stderr.

smsg
: The error message; this will usually be ErrMsg(i) where i is an error number.

ErrorLog - error logger

Function prototype:

```
void ErrorLog (int, char *, FILE *, char *);
```

Argument description:

```
--------------------------------
              Argument | Type
--------------------------------
void ErrorLog (error,    | int
               func,     | char *
               efile,    | FILE *
               smsg);    | char *
--------------------------------
```

error
: The error number, as described in the error table.

func
: A string specifying the function that is invoking the error call.

efile
: Specifies the file where the error message will be written. By default error messages are written to stderr.

smsg
: The error message; this will usually be ErrMsg(i) where i is an error number.

ErrMsg - retrieves error messages

Function prototype:

```
char *ErrMsg(int i)
```

Return value:

ErrMsg returns the error message associated with its integer argument.

Argument description:

```
-----------------------------------
                  Argument | Type
-----------------------------------
char *ErrMsg (error_num    | int
                       );  |
-----------------------------------
```

error_num
: An error number.

ErrMax - retrieves the number of possible error messages

Function prototype:

```
int ErrMax()
```

Return value:

The value returned is the total number of possible error messages.

Error table

The following table lists all of the error numbers and the associated error messages:

Error number	Error message
1	Insufficient data in gridded region to triangulate.
2	Duplicate input data coordinates are not allowed.
3	Unable to open file for writing algorithmic data.
4	WARNING: The ratio of vertical to horizontal scales is too large for meaningful gradient estimation. Rescale the data if gradients are required.
5	WARNING: The ratio of vertical to horizontal scales is too small for meaningful gradient estimation. Rescale the data if gradients are required.
6	WARNING: The ratio of x-axis breadth to y-axis breadth of this gridded region may be too extreme for good interpolation. Changing the block proportions, or rescaling the x or y coordinate may be indicated. Gradient calculations have been disabled.
7	Unable to allocate storage for ivector.
8	Unable to allocate storage for dvector.
9	Unable to allocate storage for **imatrix.
10	Unable to allocate storage for imatrix[].
11	Unable to allocate storage for **fmatrix.
12	Unable to allocate storage for fmatrix[].
13	Unable to allocate storage for **dmatrix.
14	Unable to allocate storage for dmatrix[].
15	Unable to allocate storage for raw data.
16	Unable to allocate storage for a simplex.
17	Unable to allocate storage for temp.
18	Unable to allocate storage for neig.
19	Slopes have not been computed, set sdip.
20	Row argument out of range.
21	Column argument out of range.
22	Aspects have not been computed, set sdip.
23	Parameter name not known.
24	Cannot open error file.
25	Automatic scaling has been done - aspects will be distorted and consequently are not returned. Rescale your data manually, or by setting magx, magy, and magz appropriately.
26	Automatic scaling has been done - slopes will be distorted and consequently are not returned. Rescale your data manually, or by setting magx, magy, and magz appropriately.
27	Coordinate is outside of the gridded region for a single point interpolation.
28	Cannot compute aspects and slopes in conjunction with single point interpolation mode.
29	Fortran DOUBLE PRECISION entries are not supported on UNICOS.
30	Error number out of range.

Algorithmic display tool

What the tool is and how to invoke it

This module discusses a tool for displaying various features of the Natgrid algorithm. This tool is available only with the NCAR Graphics distribution.

The tool is an executable called "nnalg". If you are using NCAR Graphics and have properly set NCARG_ROOT, "nnalg" will be in your search path.

"nnalg" takes a single argument which is the name of a special file created by the Fortran subroutine NATGRIDS or the C function c_natgrids. This file contains data appropriate for displaying various features of the Natgrid algorithm. By default, this file is not created. In order to create it, you need to set the value of the control parameter adf to 1. You can additionally override the default file name of "nnalg.dat" by using the parameter alg. Example 8 shows an example of how to create a data file with a user-specified name.

To invoke "nnalg", simply type the command:

```
nnalg data_file
```

where "data_file" is a file that has been created by the Fortran subroutine NATGRIDS or the C function c_natgrids.

By editing the data file, you have many options for what is displayed and how the plot appears. All of these options are discussed below.

What can be displayed

The features that can be displayed are:

1. The input data points
2. The natural neighbor circumcircles and their centers
3. The Delaunay triangulation, derived from the natural neighbor circumcircles
4. The Voronoi polygons
5. The first order natural neighbors of a given input point (or points)
6. The second order natural neighbors of a given input point (or points)

The first four items in the above list are displayed in the default case. The plot in Example 8 illustrates plotting these four items.

Note that there is a very interesting relationship between the natural neighbor circumcircles, the Delaunay triangulation, and the Voronoi polygons: the aggregate of the triangles created within each circumcircle form a Delaunay triangulation; lines connecting the circumcircle centers that are perpendicular to the edges in the Delaunay triangulation form the Voronoi polygons.

Where the plot will be displayed

Using the file example8.dat for reference, the first non-comment line in the integer flags section controls where the output will be dispatched upon executing *nnalg example8.dat*. By default the plot will be displayed in an X11 window that is created at execution time. The other choices you have are producing an NCAR CGM file (with name "gmeta"), or a PostScript file (with name "gmeta1.ps").

Controlling what gets plotted

The integer flags section of example8.dat controls what gets plotted. The comments there are intended to be self-explanatory. The pseudo data are internally calculated as extreme points that lie on a least-squares plane fitted to the original data. The pseudo data points are used for extrapolation. If you choose to display them, what you will usually get is the interesting part of the plot squeezed into a tiny area with only the pseudo points being meaningfully displayed.

Changing colors

The color information section of example8.dat allows you to assign colors to various plot components, as described.

Changing sizes

The scale factors section of example8.dat allows you to scale various aspects of the plot, as described.

Changing the plot view window

The user coordinate specification section of example8.dat allows you to change what part of the plot you want to look at. The default setting of all zeros tells nnalg to compute data minimums and maximums and look at an area slightly larger than that.

In the non-default case, the four numbers in the user coordinate specification line control what part of the plot you want to look at. The numbers specify, in user coordinates, the minimum X coordinate value, the maximum X coordinate value, the minimum Y coordinate value, and the maximum Y coordinate value, respectively, of the viewing window. If any single value is changed, then no default applies and all four values will be used to determine the extent of the viewing area.

Plotting first and second order natural neighbors

Sometimes it is of interest to see what input data points are being used to calculate an interpolated value. These points are the natural neighbors. This could shed light on why an interpolated value might appear to be different from what is expected. This is particularly important for interpolated values near the convex hull of a dataset where the convex hull differs significantly from the bounding polygon.

Example 9 takes you through a sequence of plots illustrating the plotting of first and second order natural neighbors.

Table of Natgrid control parameters

The following table summarizes all of the parameters used for controlling the behavior of Natgrid.

Values for these parameters can be set by using the subroutines NNSETI, NNSETR, NNSETRD, and NNSETC for Fortran programs or the functions c_nnseti, c_nnsetr, c_nnsetrd, and c_nnsetc for C programs.

Parameter name	parameter type	description	legal values	default value
adf	int	flags if the data file of algorithmic info for display is produced (see parameter **alg**).	0 = no 1 = yes	0
alg	char	file name for the algorithm display tool (see parameter **adf**).	any file name	"nnalg.dat"
asc	int	flags if automatic scaling is allowed	0 = no 1 = yes	1
bI	float	tautness factor that increases the effect of the gradients as **bI** increases.	>= 1.	1.5
bJ	float	tautness factor that decreases the breadth of the region affected by the gradients.	>= 1.	7.0
dup	int	controls whether duplicate input coordinates are allowed.	0 = no 1 = yes	0
ext	int	flags whether extrapolation is allowed outside the convex hull.	0 = no 1 = yes	1
hor	float	Specifies the amount of horizontal overlap to be included outside of the current region.	>= 0.	0.1*(extent of X values in output grid in the natgrid invocation)
igr	int	flag indicating if	0 = no	0

		gradients are to be computed.	1 = yes	
magx	float	scale factor for X coordinate values.	> 0.	1.0
magy	float	scale factor for Y coordinate values.	> 0.	1.0
magz	float	scale factor for Z coordinate values.	> 0.	1.0
non	int	flags whether interpolated values are allowed to be negative.	0 = yes 1 = no	0
nul	float	the value to be used on output for points outside of the convex hull when extrapolation is not allowed.	Any float	0.0
rad	int	flags if slopes and aspects are returned in radians or degrees.	0 = radians 1 = degrees	0
sdi	int	flags if slopes and aspects are computed	0 = no 1 = yes	0
upd	int	flags if the output array goes from south to north or north to south.	0 = N to S 1 = S to N	1
ver	float	Specifies the amount of vertical overlap to be included outside of the current region.	>= 0.	0.1*(extent of Y values in the output grid in the natgrid invocation)
xas	float	Contains the scale value for automatic scaling of X values in the most recent interpolation call.	> 0.	1.
yas	float	Contains the scale value for automatic scaling of Y values in the most recent interpolation call.	> 0.	1.
zas	float	Contains the scale value for automatic scaling of Z values in the	> 0.	1.

```
         |            | most recent           |               |
         |            | interpolation call.   |               |
------------------------------------------------------------------------------
```

Detailed description of the procedures

NATGRIDS - primary gridding function

NATGRIDS is the Fortran single-precision subroutine that does an interpolation from 2D random data to an output grid. NATGRIDS is called after all of the desired values for control parameters have been set using the subroutines NNSETI, NNSETR, NNSETC.

```
--------------------------------------------------------------
               Argument | Type      |  Mode   | Dimension
--------------------------------------------------------------
CALL NATGRIDS (NPNTS,   | Integer   | Input   |
               X,       | Real      | Input   | NPNTS
               Y,       | Real      | Input   | NPNTS
               Z,       | Real      | Input   | NPNTS
               NUMXOUT, | Integer   | Input   |
               NUMYOUT, | Integer   | Input   |
               XI,      | Real      | Input   | NUMXOUT
               YI,      | Real      | Input   | NUMYOUT
               ZI,      | Real      | Output  | NUMXOUT x NUMYOUT
               IER)     | Integer   | Output  |
--------------------------------------------------------------
```

NPNTS
: The number of input data points. (NPNTS > 3)

X
: An array containing the X coordinates of the input data points.

Y
: An array containing the Y coordinates of the input data points.

Z
: An array containing the functional values of the input data points. That is, Z(L) is the value of the input function at coordinate (X(L),Y(L)), for L=1,NPNTS.

NUMXOUT
: The number of X values in the output grid.

NUMYOUT
: The number of Y values in the output grid.

XI
: An array of dimension NUMXOUT containing the X coordinates of the output data grid. The values in XI must be increasing, but need not be equally spaced.

YI
: An array of dimension NUMYOUT containing the Y coordinates of the output data grid. The values in YI must be increasing, but need not be equally spaced.

ZI

A two-dimensional array of dimension NUMXOUT x NUMYOUT containing the interpolated functional values. ZI(I,J) is the interpolated value at grid point (XI(I),YI(J)).

IER

An error return value. If IER is returned as 0, then no errors were detected. If IER is non-zero, then refer to the error list in the error table for details.

NNSETI - set INTEGER parameter values

NNSETI is used to set values for any of the control parameters that take integer values. The values set by NNSETI remain in effect until changed by subsequent calls to NNSETI.

```
-----------------------------------------------------
            Argument | Type       |  Mode    | Dimension
-----------------------------------------------------
CALL NNSETI  (PNAM,  | Character  | Input    |
              IVAL)  | Integer    | Input    |
-----------------------------------------------------
```

PNAM

The name of the control parameter to be set.

IVAL

The value to be assigned to the parameter.

NNGETI - retrieve values for INTEGER parameters

NNGETI is a called to obtain current values for any of the INTEGER valued control parameters.

```
-----------------------------------------------------
            Argument | Type       |  Mode    | Dimension
-----------------------------------------------------
CALL NNGETI  (PNAM,  | Character  | Input    |
              IVAL)  | Integer    | Output   |
-----------------------------------------------------
```

PNAM

The name of the control parameter whose value is to be retrieved.

IVAL

The current value assigned to the control parameter.

NNSETR - set REAL parameter values

NNSETR is used to set values for any of the control parameters that take REAL values. The values set by NNSETR remain in effect until changed by subsequent calls to NNSETR.

Argument	Type	Mode	Dimension
CALL NNSETR (PNAM,	Character	Input	
RVAL)	Real	Input	

PNAM
    The name of the control parameter to be set.
RVAL
    The value to be assigned to the parameter.

NNGETR - retrieve values for REAL parameters

NNGETR is a called to obtain current values for any of the REAL valued control parameters.

Argument	Type	Mode	Dimension
CALL NNGETR (PNAM,	Character	Input	
RVAL)	Real	Output	

PNAM
    The name of the control parameter whose value is to be retrieved.
RVAL
    The current value assigned to the control parameter.

NNSETC - set CHARACTER valued parameters

Argument	Type	Mode	Dimension
CALL NNSETC (PNAM,	Character	Input	
CVAL)	Character	Input	

NNSETC is used to set values for any of the control parameters that take CHARACTER values. The values set by NNSETC remain in effect until changed by subsequent calls to NNSETC.

PNAM
    The name of the control parameter to be set.
CVAL
    The value to be assigned to the parameter.

NNGETC - retrieve values for CHARACTER parameters

NNGETC is a called to obtain current values for any of the CHARACTER valued control parameters.

Argument	Type	Mode	Dimension
CALL NNGETC (PNAM,	Character	Input	
CVAL)	Character	Output	

PNAM
: The name of the control parameter whose value is to be retrieved.

CVAL
: The current value assigned to the control parameter. Be sure to provide enough space in the character variable to receive the value.

NNGETASPECTS - get aspect values, if calculated

NNGETASPECTS is called to retrieve an aspect, in single precision, at a specified coordinate value. For further details, see the module on computing aspects and slopes.

Argument	Type	Mode	Dimension
CALL NNGETASPECTS (I,	Integer	Input	
J,	Integer	Input	
ASPECT,	Real	Output	
IER)	Integer	Output	

I
: A subscript indexing the first dimensioned variable in the output grid of the most recent call to NATGRIDS.

J
: A subscript indexing the second dimensioned variable in the output grid in the most recent call to NATGRIDS.

ASPECT
: The aspect at the grid point Z(I,J), where Z is the output grid in the most recent call to NATGRIDS.

IER
: An error return value. If IER is returned as 0, then no errors were detected. If IER is non-zero, then refer to the error list in the error table for details.

NNGETSLOPES - get slope values, if calculated

NNGETSLOPES is called to retrieve a slope, in single precision, at a specified coordinate value. For further details, see the module on computing aspects and slopes.

```
-----------------------------------------------------------
                  Argument | Type        |  Mode   | Dimension
-----------------------------------------------------------
CALL NNGETSLOPES (I,       | Integer     | Input   |
                  J,       | Integer     | Input   |
                  SLOPE,   | Real        | Output  |
                  IER)     | Integer     | Output  |
-----------------------------------------------------------
```

I
: A subscript indexing the first dimensioned variable in the output grid of the most recent call to NATGRIDS.

J
: A subscript indexing the second dimensioned variable in the output grid in the most recent call to NATGRIDS.

SLOPE
: The slope at the grid point Z(I,J), where Z is the output grid in the most recent call to NATGRIDS.

IER
: An error return value. If IER is returned as 0, then no errors were detected. If IER is non-zero, then refer to the error list in the error table for details.

NNPNTINITS - initiate single-point mode

This subroutine calculates all natural neighbor relationships in an input data array and sets some internal parameters so that NNPNTS can be called to interpolate at individual points.

```
-----------------------------------------------------------
                  Argument | Type        |  Mode   | Dimension
-----------------------------------------------------------
CALL NNPNTINITS (NPNTS,    | Integer     | Input   |
                 X,        | Real        | Input   | NPNTS
                 Y,        | Real        | Input   | NPNTS
                 Z)        | Real        | Input   | NPNTS
-----------------------------------------------------------
```

NPNTS
: The number of input data points. (NPNTS > 3)

X
: An array of dimension NPNTS containing the X coordinates of the input data points.

Y
: An array of dimension NPNTS containing the Y coordinates of the input data points.

Z
: An array of dimension NPNTS containing the functional values of the input data points. That is, Z(L) is the value of the input function at coordinate (X(L),Y(L)), for L=1,NPNTS.

NNPNTS - interpolate at a single point

This subroutine is called to interpolate at a specified point. Before calling this subroutine, NNPNTINITS must be called.

```
-------------------------------------------------------
             Argument | Type        |  Mode  | Dimension
-------------------------------------------------------
CALL NNPNTS  (X,      | Real        | Input  |
              Y,      | Real        | Input  |
              Z)      | Real        | Output |
-------------------------------------------------------
```

X
: The X coordinate of the point where interpolation is desired.

Y
: The Y coordinate of the point where interpolation is desired.

Z
: The interpolated functional value at the point (X,Y).

NNPNTEND - terminate single-point mode

This subroutine is called to terminate interpolation at single points. It is called after having made previous calls to NNPNTINITS and NNPNTS.

```
--------------------------------------------
        Argument | Type |   Mode   | Dimension
--------------------------------------------
CALL NNPNTEND () | N/A  |   N/A    | N/A
--------------------------------------------
```

NATGRIDD - primary gridding function

NATGRIDD is the Fortran double-precision subroutine that does an interpolation from 2D random data to an output grid. NATGRIDD is called after all of the desired values for control parameters have been set using the subroutines NNSETI, NNSETRD, NNSETC.

```
------------------------------------------------------------------------
               Argument | Type                 |  Mode   | Dimension
------------------------------------------------------------------------
CALL NATGRIDD (NPNTS,   | Integer              | Input   |
               X,       | Double precision     | Input   | NPNTS
               Y,       | Double precision     | Input   | NPNTS
               Z,       | Double precision     | Input   | NPNTS
               NUMXOUT, | Integer              | Input   |
               NUMYOUT, | Integer              | Input   |
               XI,      | Double precision     | Input   | NUMXOUT
               YI,      | Double precision     | Input   | NUMYOUT
```

ZI,	Double precision	Output	NUMXOUT x NUMYOUT
IER)	Integer	Output	

NPNTS
: The number of input data points. (NPNTS > 3)

X
: An array of dimension NPNTS containing the X coordinates of the input data points.

Y
: An array of dimension NPNTS containing the Y coordinates of the input data points.

Z
: An array of dimension NPNTS containing the functional values of the input data points. That is, Z(L) is the value of the input function at coordinate (X(L),Y(L)), for L=1,NPNTS.

NUMXOUT
: The number of X values in the output grid.

NUMYOUT
: The number of Y values in the output grid.

XI
: An array of dimension NUMXOUT containing the X coordinates of the output data grid. The values in XI must be increasing, but need not be equally spaced.

YI
: An array of dimension NUMYOUT containing the Y coordinates of the output data grid. The values in YI must be increasing, but need not be equally spaced.

ZI
: A two-dimensional array of dimension NUMXOUT x NUMYOUT containing the interpolated functional values. ZI(I,J) is the interpolated value at grid point (XI(I),YI(J)).

IER
: An error return value. If IER is returned as 0, then no errors were detected. If IER is non-zero, then refer to the error list in the error table for details.

NNSETRD - set DOUBLE PRECISION parameter values

NNSETRD is used to set values for any of the control parameters that take DOUBLE PRECISION values. The values set by NNSETRD remain in effect until changed by subsequent calls to NNSETRD.

Argument	Type	Mode	Dimension
CALL NNSETRD (PNAM,	Character	Input	
RVAL)	Double precision	Input	

PNAM
: The name of the control parameter to be set.

RVAL
: The value to be assigned to the parameter.

NNGETRD - retrieve values for DOUBLE PRECISION parameters

NNGETRD is a called to obtain current values for any of the DOUBLE PRECISION valued control parameters.

```
-------------------------------------------------------------
             Argument | Type               |  Mode   | Dimension
-------------------------------------------------------------
CALL NNGETRD (PNAM,   | Character          | Input   |
              RVAL)   | Double precision   | Output  |
-------------------------------------------------------------
```

PNAM
: The name of the control parameter whose value is to be retrieved.

RVAL
: The current value assigned to the control parameter.

NNGETASPECTD - get aspect values, if calculated

NNGETASPECTD is called to retrieve an aspect, in double precision, at a specified coordinate value. For further details, see the module on computing aspects and slopes.

```
------------------------------------------------------------------
                  Argument | Type               |  Mode   | Dimension
------------------------------------------------------------------
CALL NNGETASPECTD (I,      | Integer            | Input   |
                   J,      | Integer            | Input   |
                   ASPECT, | Double precision   | Output  |
                   IER)    | Integer            | Output  |
------------------------------------------------------------------
```

I
: A subscript indexing the first dimensioned variable in the output grid of the most recent call to NATGRIDD.

J
: A subscript indexing the second dimensioned variable in the output grid in the most recent call to NATGRIDD.

ASPECT
: The aspect at the grid point Z(I,J), where Z is the output grid in the most recent call to NATGRIDD.

IER
: An error return value. If IER is returned as 0, then no errors were detected. If IER is non-zero, then refer to the error list in the error table for details.

NNGETSLOPED - get slope values, if calculated

NNGETSLOPED is called to retrieve a slope, in double precision, at a specified coordinate value. For further details, see the module on computing aspects and slopes.

```
------------------------------------------------------------------
                 Argument | Type                | Mode   | Dimension
------------------------------------------------------------------
CALL NNGETSLOPED (I,      | Integer             | Input  |
                  J,      | Integer             | Input  |
                  SLOPE,  | Double precision    | Output |
                  IER)    | Integer             | Output |
------------------------------------------------------------------
```

I
: A subscript indexing the first dimensioned variable in the output grid of the most recent call to NATGRIDD.

J
: A subscript indexing the second dimensioned variable in the output grid in the most recent call to NATGRIDD.

SLOPE
: The slope at the grid point Z(I,J), where Z is the output grid in the most recent call to NATGRIDD.

IER
: An error return value. If IER is returned as 0, then no errors were detected. If IER is non-zero, then refer to the error list in the error table for details.

NNPNTINITD - initiate single-point mode

This subroutine calculates all natural neighbor relationships in an input data array and sets some internal parameters so that NNPNTD can be called to interpolate at individual points.

```
------------------------------------------------------------------
                 Argument | Type                | Mode   | Dimension
------------------------------------------------------------------
CALL NNPNTINITD (NPNTS,   | Integer             | Input  |
                 X,       | Double precision    | Input  | NPNTS
                 Y,       | Double precision    | Input  | NPNTS
                 Z)       | Double precision    | Input  | NPNTS
------------------------------------------------------------------
```

NPNTS
: The number of input data points. (NPNTS > 3)

X
: An array of dimension NPNTS containing the X coordinates of the input data points.

Y
: An array of dimension NPNTS containing the Y coordinates of the input data points.

Z
: An array of dimension NPNTS containing the functional values of the input data points. That is,

Z(L) is the value of the input function at coordinate (X(L),Y(L)), for L=1,NPNTS.

NNPNTD - interpolate at a single point

This subroutine is called to interpolate at a specified point. Before calling this subroutine, NNPNTINITD must be called.

Argument	Type	Mode	Dimension
CALL NNPNTD (X,	Double precision	Input	
Y,	Double precision	Input	
Z)	Double precision	Output	

X
: The X coordinate of the point where interpolation is desired.

Y
: The Y coordinate of the point where interpolation is desired.

Z
: The interpolated functional value at the point (X,Y).

NNPNTENDD - terminate single-point mode

This subroutine is called to terminate interpolation at single points. It is called after having made previous calls to NNPNTINITD and NNPNTD.

Argument	Type	Mode	Dimension
CALL NNPNTENDD ()	N/A	N/A	N/A

---

c_natgrids - Primary gridding function

c_natgrids is the C single-precision function that does an interpolation from 2D random data to an output grid. c_natgrids is called after all of the desired values for control parameters have been set using the procedures c_nnseti, c_nnsetr, c_nnsetc.

Function prototype:

```
float *c_natgrids(int, float [], float [], float [],
                  int, int, float [], float [], int *);
```

Return value:

c_natgrids returns a pointer to a linear array of data that is the interpolated grid stored in row-major order. That is, if *out* is declared as

```
float *out;
```

and we set:

```
  out = c_natgrids(npnts, x, y, z, numxout, numyout, xo, yo, &ier);
```

then out[i*numyout+j] is the interpolated function value at coordinate point (xo[i],yo[j]) for 0 <= i < numxout and 0 <= j < numyout. The space for *out* is allocated internal to *c_natgrids* and is numxout*numyout floats in size.

Argument description:

```
--------------------------------------------------
                   Argument | Type       |  Size
--------------------------------------------------
float *c_natgrids (npnts,   | int        |
                   x,       | float []   | npnts
                   y,       | float []   | npnts
                   z,       | float []   | npnts
                   numxout, | int        |
                   numyout, | int        |
                   xi,      | float []   | numxout
                   yi,      | float []   | numyout
                   ier      | int *      |
                  );
--------------------------------------------------
```

npnts
: The number of input data points. (npnts > 3)

x
: An array of size npnts containing the X coordinates of the input data points.

y
: An array of size npnts containing the Y coordinates of the input data points.

z
: An array of size npnts containing the functional values of the input data points. That is, z[n] is the value of the input function at coordinate (x[n],y[n]), for 0 <= n < npnts.

numxout
: The number of X values in the output grid.

numyout
: The number of Y values in the output grid.

xi
: An array of size numxout containing the X coordinates of the output data grid. The values in xi must be increasing, but need not be equally spaced.

yi
: An array of size numyout containing the Y coordinates of the output data grid. The values in yi must be increasing, but need not be equally spaced.

ier

An error return value. If *ier is returned as 0, then no errors were detected. If *ier is non-zero, then refer to the error list in the error table for details.

---

c_nnseti - Set int valued parameters

c_nnseti is used to set values for any of the control parameters that take int values. The values set by c_nnseti remain in effect until changed by subsequent calls to c_nnseti.

Function prototype:

```
void c_nnseti(char *, int);
```

Argument description:

```
-------------------------------------------
              Argument | Type      |  Size
-------------------------------------------
void c_nnseti  (pnam,  | char *    |
               ival);  | int       |
-------------------------------------------
```

pnam
: The name of the control parameter to be assigned an int value.

ival
: The value to be assigned to the control parameter whose name is pointed to by pnam.

---

c_nngeti - Retrieve an int valued parameter

c_nngeti is a called to obtain current values for any of the int valued control parameters.

Function prototype:

```
void c_nngeti(char *, int);
```

Argument description:

```
-------------------------------------------
              Argument | Type      |  Size
-------------------------------------------
void c_nngeti  (pnam,  | char *    |
               ival);  | int   *   |
-------------------------------------------
```

pnam
: The name of the control parameter whose value is to be retrieved.

ival
: *ival will be the value currently assigned to the control parameter whose name is pointed to by

pnam.

---

c_nnsetr - Set float valued parameters

c_nnsetr is used to set values for any of the control parameters that take float values. The values set by c_nnsetr remain in effect until changed by subsequent calls to c_nnsetr.

Function prototype:

```
void c_nnsetr(char *, float);
```

Argument description:

Argument	Type	Size
void c_nnsetr (pnam,	char *	
fval);	float	

pnam
: The name of the control parameter to be assigned a float value.

fval
: The value to be assigned to the control parameter whose name is pointed to by pnam.

---

c_nngetr - Retrieve a float valued parameter

c_nngetr is a called to obtain current values for any of the float valued control parameters.

Function prototype:

```
void c_nngetr(char *, float *);
```

Argument description:

Argument	Type	Size
void c_nngetr (pnam,	char *	
fval);	float *	

pnam
: The name of the control parameter whose value is to be retrieved.

fval
: *fval will be the value currently assigned to the control parameter whose name is pointed to by pnam.

c_nnsetc - Set char valued parameters

c_nnsetc is used to set values for any of the control parameters that take string values. The values set by c_nnsetc remain in effect until changed by subsequent calls to c_nnsetc.

Function prototype:

```
void c_nnsetc(char *, char *);
```

Argument description:

Argument	Type	Size
void c_nnsetc (pnam,	char *	
cval);	char *	

pnam
: The name of the control parameter to be assigned a char value.

cval
: The value to be assigned to the control parameter whose name is pointed to by pnam.

c_nngetc - Retrieve a char valued parameter

c_nngetc is a called to obtain current values for any of the string valued control parameters.

Function prototype:

```
void c_nngetc(char *, char *);
```

Argument description:

Argument	Type	Size
void c_nngetc (pnam,	char *	
cval);	char *	

pnam
: The name of the control parameter whose value is to be retrieved.

cval
: cval will point to a string containing the returned value. The user is required to reserve enough space to store the string.

c_nngetaspects - Retrieve aspect values, if calculated

c_nngetaspects is called to retrieve an aspect, in single precision, at a specified coordinate value. For further details, see the module on computing aspects and slopes.

Function prototype:

```
void c_nngetaspects(int, int, float *, int *);
```

Argument description:

```
----------------------------------------------
                    Argument | Type     |  Size
----------------------------------------------
void c_nngetaspects (row,    | int      |
                     column, | int      |
                     aspect, | float *  |
                     ier);   | int *    |
----------------------------------------------
```

row
: A subscript indexing the first dimensioned variable in the 2D grid array returned from the most recent call to c_natgrids.

column
: A subscript indexing the second dimensioned variable in the 2D grid array returned from the most recent call to c_natgrids.

aspect
: *aspect is the aspect at the grid point z[i,j], where z is the output grid in the most recent call to c_natgrids.

ier
: *ier is an error return value. If *ier is 0, then no errors were detected. If *ier is non-zero, then refer to the error list in the error table for details.

---

c_nngetslopes - Retrieve slope values, if calculated

c_nngetslopes is called to retrieve a slope, in single precision, at a specified coordinate value. For further details, see the module on computing aspects and slopes.

Function prototype:

```
void c_nngetslopes(int, int, float *, int *);
```

Argument description:

```
----------------------------------------------
                    Argument | Type     |  Size
----------------------------------------------
```

```
void c_nngetslopes  (row,      | int        |
                     column,   | int        |
                     slope,    | float *    |
                     ier);     | int *      |
-----------------------------------------------
```

row
: A subscript indexing the first dimensioned variable in the 2D grid array returned from the most recent call to c_natgrids.

column
: A subscript indexing the second dimensioned variable in the 2D grid array returned from the most recent call to c_natgrids.

slope
: *slope is the slope at the grid point z[i,j], where z is the output grid in the most recent call to c_natgrids.

ier
: *ier is an error return value. If *ier is 0, then no errors were detected. If *ier is non-zero, then refer to the error list in the error table for details.

---

c_nnpntinits - Enter single-point mode

This function calculates all natural neighbor relationships in an input data array and sets some internal parameters so that c_nnpnts can be called to interpolate at individual points.

Function prototype:

```
  void c_nnpntinits(int, float [], float [], float []);
```

Argument description:

```
-----------------------------------------------
                  Argument | Type       |  Size
-----------------------------------------------
void c_nnpntinits (npnts,  | int        |
                   x,      | float []   | npnts
                   y,      | float []   | npnts
                   z);     | float []   | npnts
-----------------------------------------------
```

npnts
: The number of input data points. (npnts > 3)

x
: An array of size npnts containing the X coordinates of the input data points.

y
: An array of size npnts containing the Y coordinates of the input data points.

z
: An array of size npnts containing the functional values of the input data points. That is, z[n] is the value of the input function at coordinate (x[n],y[n]), for 0 <= n < npnts.

---

c_nnpnts - Interpolate at a single point

This function is called to interpolate at a specified point. Before calling this function, c_nnpntinits must be called.

Function prototype:

```
  void c_nnpnts(float, float, float *);
```

Argument description:

```
-----------------------------------------------
              Argument | Type       |  Size
-----------------------------------------------
void c_nnpnts (x,      | float      |
               y,      | float      |
               z);     | float *    |
-----------------------------------------------
```

x
: The X coordinate of the point where interpolation is desired.

y
: The Y coordinate of the point where interpolation is desired.

z
: *z is the interpolated functional value at the point (X,Y).

---

c_nnpntend - Exit single-point mode

This function is called to terminate interpolation at single points. It is called after having made previous calls to c_nnpntinits and c_nnpnts.

Function prototype:

```
  void c_nnpntend();
```

Argument description:

```
-----------------------------------------------
                Argument | Type       |  Size
-----------------------------------------------
void c_nnpntend ();      | N/A        |  N/A
-----------------------------------------------
```

---

c_natgridd - Primary gridding function

c_natgridd is the C double-precision function that does an interpolation from 2D random data to an output grid. c_natgrids is called after all of the desired values for control parameters have been set using the function c_nnseti, c_nnsetrd, c_nnsetc.

Function prototype:

```
  double *c_natgridd(int, double [], double [], double [],
                     int, int, double [], double [], int *);
```

Return value:

c_natgridd returns a pointer to a linear array of data that is the interpolated grid stored in row-major order. That is, if *out* is declared as

```
double *out;
```

and we set:

```
  out = c_natgridd(numin, x, y, z, numxout, numyout, xo, yo, &ier);
```

then out[i*numyout+j] is the interpolated function value at coordinate point (xo[i],yo[j]) for 0 <= i < numxout and 0 <= j < numyout. The space for *out* is allocated internal to *c_natgridd* and is numxout*numyout double-precision values in size.

Argument description:

```
-------------------------------------------------
                   Argument | Type        |  Size
-------------------------------------------------
double *c_natgridd (npnts,  | int         |
                   x,       | double []   | npnts
                   y,       | double []   | npnts
                   z,       | double []   | npnts
                   numxout, | int         |
                   numyout, | int         |
                   xi,      | double []   | numxout
                   yi,      | double []   | numyout
                   ier      | int *       |
                  );        |             |
-------------------------------------------------
```

npnts
: The number of input data points. (npnts > 3)

x
: An array of size npnts containing the X coordinates of the input data points.

y
: An array of size npnts containing the Y coordinates of the input data points.

z
: An array of size npnts containing the functional values of the input data points. That is, z[n] is the value of the input function at coordinate (x[n],y[n]), for 0 <= n < npnts.

numxout
: The number of X values in the output grid.

numyout

The number of Y values in the output grid.

xi
: An array of size numxout containing the X coordinates of the output data grid. The values in xi must be increasing, but need not be equally spaced.

yi
: An array of size numyout containing the Y coordinates of the output data grid. The values in yi must be increasing, but need not be equally spaced.

ier
: An error return value. If *ier is returned as 0, then no errors were detected. If *ier is non-zero, then refer to the error list in the error table for details.

---

c_nnsetrd - Set double-precision parameter values

c_nnsetrd is used to set values for any of the control parameters that take double-precision values. The values set by c_nnsetrd remain in effect until changed by subsequent calls to c_nnsetrd.

Function prototype:

```
void c_nnsetrd(char *, double);
```

Argument description:

Argument	Type	Size
void c_nnsetrd (pnam,	char *	
dval);	double	

pnam
: The name of the control parameter to be assigned a float value.

dval
: The value to be assigned to the control parameter whose name is pointed to by pnam.

---

c_nngetrd - Retrieve a double-precision parameter value

c_nngetrd is a called to obtain current values for any of the double-precision control parameters.

Function prototype:

```
void c_nngetrd(char *, double *);
```

Argument description:

Argument	Type	Size

```
-------------------------------------------
void c_nngetrd (pnam,     | char *   |
                dval);    | double * |
-------------------------------------------
```

pnam
: The name of the control parameter whose value is to be retrieved.

dval
: *dval is the value currently assigned to the control parameter whose name is pointed to by pnam.

---

c_nngetaspectd - Retrieve aspect values, if calculated

c_nngetaspectd is called to retrieve an aspect, in double precision, at a specified coordinate value. For further details, see the module on computing aspects and slopes.

Function prototype:

```
  void c_nngetaspectd(int, int, double *, int *);
```

Argument description:

```
-----------------------------------------------
                    Argument | Type       |  Size
-----------------------------------------------
void c_nngetaspectd (row,    | int        |
                     column, | int        |
                     aspect, | double *   |
                     ier);   | int *      |
-----------------------------------------------
```

row
: A subscript indexing the first dimensioned variable in the 2D grid array returned from the most recent call to c_natgrids.

column
: A subscript indexing the second dimensioned variable in the 2D grid array returned from the most recent call to c_natgrids.

aspect
: *aspect is the aspect at the grid point z[i,j], where z is the output grid in the most recent call to c_natgridd.

ier
: *ier is an error return value. If *ier is 0, then no errors were detected. If *ier is non-zero, then refer to the error list in the error table for details.

---

c_nngetsloped - Retrieve slope values, if calculated

c_nngetsloped is called to retrieve a slope, in double precision, at a specified coordinate value. For further details, see the module on computing aspects and slopes.

Function prototype:

```
void c_nngetsloped(int, int, double *, int *);
```

Argument description:

```
----------------------------------------------
                    Argument | Type       |  Size
----------------------------------------------
void c_nngetsloped (row,     | int        |
                    column,  | int        |
                    slope,   | double *   |
                    ier);    | int *      |
----------------------------------------------
```

row
: A subscript indexing the first dimensioned variable in the output grid of the most recent call to c_natgridd.

column
: A subscript indexing the second dimensioned variable in the output grid of the most recent call to c_natgridd.

slope
: *slope is the slope at the grid point z[i,j], where z is the output grid in the most recent call to c_natgridd.

ier
: *ier is an error return value. If *ier is 0, then no errors were detected. If *ier is non-zero, then refer to the error list in the error table for details.

---

c_nnpntinitd - Enter single-point mode

This function calculates all natural neighbor relationships in an input data array and sets some internal parameters so that c_nnpntd can be called to interpolate at individual points.

Function prototype:

```
void c_nnpntinitd(int, double [], double [], double []);
```

Argument description:

```
-----------------------------------------------
                   Argument | Type        |  Size
-----------------------------------------------
void c_nnpntinitd (npnts,   | int         |
                   x,       | double []   | npnts
                   y,       | double []   | npnts
                   z);      | double []   | npnts
-----------------------------------------------
```

npnts
: The number of input data points. (npnts > 3)

x
: An array of size npnts containing the X coordinates of the input data points.

y
: An array of size npnts containing the Y coordinates of the input data points.

z
: An array of size npnts containing the functional values of the input data points. That is, z[n] is the value of the input function at coordinate (x[n],y[n]), for 0 <= n < npnts.

---

c_nnpntd - Interpolate at a single point

This function is called to interpolate at a specified point. Before calling this function, c_nnpntinitd must be called.

Function prototype:

```
  void c_nnpntd(double, double, double *);
```

Argument description:

```
-------------------------------------------
              Argument | Type       |  Size
-------------------------------------------
void c_nnpntd (x,      | double     |
               y,      | double     |
               z);     | double *   |
-------------------------------------------
```

x
: The X coordinate of the point where interpolation is desired.

y
: The Y coordinate of the point where interpolation is desired.

z
: *z is the interpolated functional value at the point (X,Y).

---

c_nnpntendd - Exit single-point mode

This function is called to terminate interpolation at single points. It is called after having made previous calls to c_nnpntinitd and c_nnpntd.

Function prototype:

```
  void c_nnpntendd();
```

Argument description:

```
-----------------------------------------------
                 Argument | Type       |  Size
-----------------------------------------------
void c_nnpntendd ()       | N/A        |  N/A
-----------------------------------------------
```

Overview of Natgrid procedures

The table below summarizes all of the procedures in Natgrid for the four user interfaces (single and double precision in both Fortran and C).

Clicking on the procedure name will link to a detailed description of that procedure.

```
Fortran single precision:
------------------------

 NATGRIDS     -  primary function for gridding.
 NNSETI       -  set INTEGER parameter values.
 NNGETI       -  retrieve values for INTEGER parameters.
 NNSETR       -  set REAL parameter values.
 NNGETR       -  retrieve values for REAL parameters.
 NNSETC       -  set CHARACTER valued parameters.
 NNGETC       -  retrieve values for CHARACTER parameters.
 NNGETASPECTS -  get aspect values, if calculated.
 NNGETSLOPES  -  get slope values, if calculated.
 NNPNTINITS   -  initiate single point mode.
 NNPNTS       -  interpolate at a single point.
 NNPNTEND     -  terminate single point mode.


Fortran double precision (not supported on UNICOS):
--------------------------------------------------

 NATGRIDD     -  primary function for gridding.
 NNSETI       -  set INTEGER parameter values.
 NNGETI       -  retrieve values for INTEGER parameters.
 NNSETRD      -  set DOUBLE PRECISION parameter values.
 NNGETRD      -  retrieve values for DOUBLE PRECISION parameters.
 NNSETC       -  set CHARACTER valued parameters.
 NNGETC       -  retrieve values for CHARACTER parameters.
 NNGETASPECTD -  get aspect values, if calculated.
 NNGETSLOPED  -  get slope values, if calculated.
 NNPNTINITD   -  initiate single point mode.
 NNPNTD       -  interpolate at a single point.
 NNPNTENDD    -  terminate single point mode.


C single precision:
------------------

 c_natgrids     -  primary function for gridding.
 c_nnseti       -  set int parameter values.
 c_nngeti       -  retrieve values for int parameters.
```

```
 c_nnsetr       -  set float parameter values.
 c_nngetr       -  retrieve values for float parameters.
 c_nnsetc       -  set char valued parameters.
 c_nngetc       -  retrieve values for char parameters.
 c_nngetaspects -  get aspect values, if calculated.
 c_nngetslopes  -  get slope values, if calculated.
 c_nnpntinits   -  initiate single point mode.
 c_nnpnts       -  interpolate at a single point.
 c_nnpntend     -  terminate single point mode.


C double precision:
------------------

 c_natgridd     -  primary function for gridding.
 c_nnseti       -  set int parameter values.
 c_nngeti       -  retrieve values for int parameters.
 c_nnsetrd      -  set double parameter values.
 c_nngetrd      -  retrieve values for double parameters.
 c_nnsetc       -  set char valued parameters.
 c_nngetc       -  retrieve values for char parameters.
 c_nngetaspectd -  get aspect values, if calculated.
 c_nngetsloped  -  get slope values, if calculated.
 c_nnpntinitd   -  initiate single point mode.
 c_nnpntd       -  interpolate at a single point.
 c_nnpntendd    -  terminate single point mode.
```

Natgrid timing table

The following execution times were obtained on a Sun sparc Ultra-1 running SunOS 5.5.1 using version SC4.0 of the C compiler:

For interpolating on a grid:

Number of input points	Number of output points	Times for triangulation (in secs.)	Times for linear interpolation (in secs.)	Times for non-linear interp. (in secs.)	Time per point (linear) (secs.)	Time per point (non-lin) (secs.)
50	500	0.003	0.04	0.08	0.00008	0.00016
50	1000	0.003	0.07	0.14	0.00007	0.00014
50	2000	0.003	0.14	0.25	0.00007	0.00012
50	4000	0.003	0.27	0.45	0.00007	0.00011
50	8000	0.003	0.53	0.89	0.00007	0.00011
100	500	0.008	0.05	0.13	0.00010	0.00026
100	1000	0.008	0.10	0.20	0.00010	0.00020
100	2000	0.008	0.18	0.34	0.00009	0.00017
100	4000	0.008	0.36	0.61	0.00009	0.00015
100	8000	0.008	0.68	1.16	0.00008	0.00014
200	500	0.022	0.10	0.30	0.00020	0.00060

200	1000	0.022	0.18	0.40	0.00018	0.00040
200	2000	0.022	0.33	0.59	0.00016	0.00030
200	4000	0.022	0.62	0.97	0.00015	0.00024
200	8000	0.022	1.21	1.91	0.00015	0.00023
400	500	0.081	0.22	0.79	0.00044	0.00158
400	1000	0.081	0.39	0.94	0.00039	0.00094
400	2000	0.081	0.66	1.40	0.00033	0.00070
400	4000	0.081	1.22	1.91	0.00030	0.00048
400	8000	0.081	2.33	3.29	0.00029	0.00041
800	500	0.291	0.58	2.42	0.00116	0.00484
800	1000	0.291	0.86	2.77	0.00086	0.00277
800	2000	0.291	1.38	3.44	0.00069	0.00172
800	4000	0.291	2.43	4.64	0.00061	0.00116
800	8000	0.291	4.45	6.58	0.00056	0.00082
1600	500	1.108	1.75	8.98	0.00350	0.01796
1600	1000	1.108	2.24	9.89	0.00224	0.00989
1600	2000	1.108	3.21	10.99	0.00160	0.00550
1600	4000	1.108	5.31	12.95	0.00132	0.00323
1600	8000	1.108	9.75	18.12	0.00121	0.00227

For interpolating in single point mode:

Number of input points	Time per point for linear interpolation (in secs.)	Time per point for non-linear interpolation (in secs.)
50	0.00070	0.0269
100	0.00104	0.0765
200	0.00228	0.2195

Natgrid examples

Overview

This module contains source codes for all of the examples referenced by the other modules in the documentation. Links to all of the example plots are provided.

The following list summarizes the examples:

- Example 1 -- linear interpolation
- Example 2 -- nonlinear interpolation
- Example 3 -- computing aspects and slopes

- Example 4 -- extrapolation
- Example 5 -- negative values disallowed
- Example 6 -- single-point mode
- Example 7 -- outlying point
- Example 8 -- using the algorithmic display tool
- Example 9 -- displaying natural neighbors

All of the example codes isolate the drawing of graphics into single calls to procedures for drawing surface plots, vector plots, or contour plots. The supporting plotting procedures themselves are collected into a single file and use NCAR Graphics to draw the plots. If you are not using NCAR Graphics and want to draw the plots, you will have to substitute appropriate calls from the graphics package you are using.

Both C and Fortran versions of the plotting procedures are provided that are used by the examples. These are listed in the modules:

- nnplotf - Fortran subroutines for drawing the example plots.
- nnplotc - C functions for drawing the example plots.

Creating your own executable

If you have access to NCAR Graphics, you can create executables for all of the examples in one of two ways. The easiest is to use the ncargex utility which is documented with a man page. The other way is to use either *ncargf77* (for Fortran) or *ncargcc* (for C) to compile and execute the source codes. Use the source codes for the examples together with nnplotf or nnplotc.

If you are using NCAR Graphics and the C interface to Natgrid and you are compiling using *ncargcc*, then function prototypes for all Natgrid C functions are contained in a C header file. This can be accessed by inserting the line:

```
#include <ncarg/ngmath.h>
```

into your code.

If you do not have access to NCAR Graphics, replace the calls for drawing the plots with calls to the appropriate procedures at your site.

Example 1 -- linear interpolation

- Fortran single precision
- Fortran double precision
- C single precision
- C double precision
- Sample plot

Example 2 -- nonlinear interpolation

- Fortran single precision
- C single precision
- Sample plot

Example 3 -- computing aspects and slopes

- Fortran single precision
- C single precision
- surface plot
- vector plot of aspects
- contour plot of slopes

Example 4 -- extrapolation

- Fortran single precision
- extrapolation allowed plot
- extrapolation not allowed plot

Example 5 -- negative values disallowed

- Fortran single precision
- negative values allowed plot
- negative values not allowed plot

Example 6 -- single-point mode

- Fortran single precision
- C single precision
- single-point mode plot

Example 7 -- outlying point

- Fortran single precision
- natural neighbor vs. curve fitting

Example 8 -- using the algorithmic display tool

- Fortran single precision
- Default output file
- plot of default settings

Example 9 -- displaying first and second order natural neighbors

- Fortran single precision

- Modified output file
- first and second order natural neighbors

Fortran subroutine for drawing example plots

The following code uses NCAR Graphics to produce surface plots, vector plots, and contour plots for the Natgrid examples.

```
      SUBROUTINE DRWCON(NX,NY,XI,YI,ZDAT)
C
C  Use the NCAR Graphics CONPACK package to draw a color contour
C  plot of the data in ZDAT.
C
      PARAMETER (IERRF=6, LUNIT=2, IWKID=1, IWTYPE=1)
C
      DIMENSION ZDAT(NX,NY)
      DIMENSION RWRK(2000),IWRK(1000),IAMA(20000)
      DIMENSION XCRA(1000),YCRA(1000),IAIA(10),IGIA(10)
C
      EXTERNAL CPCOLR,CPDRPL
C
C  Open GKS if not open; open and activate a workstation; define
C  some colors.
C
      JTYPE = IWTYPE
      CALL GQOPS(ISTATE)
      IF (ISTATE .EQ. 0) THEN
        CALL GOPKS (IERRF, ISZDM)
        IF (JTYPE .EQ. 1) THEN
          CALL NGSETC('ME','con.ncgm')
        ELSE IF ( (JTYPE .GE. 20) .AND. (JTYPE .LE. 31) ) THEN
          CALL NGSETC('ME','con.ps')
        ENDIF
        CALL GOPWK (IWKID, LUNIT, JTYPE)
        CALL GACWK (IWKID)
        CALL GSCR(IWKID, 0, 1.00, 1.00, 1.00)
        CALL GSCR(IWKID, 1, 0.00, 0.00, 0.00)
        CALL GSCR(IWKID, 2, 0.00, 1.00, 1.00)
        CALL GSCR(IWKID, 3, 0.00, 1.00, 0.00)
        CALL GSCR(IWKID, 4, 0.70, 1.00, 0.00)
        CALL GSCR(IWKID, 5, 1.00, 1.00, 0.00)
        CALL GSCR(IWKID, 6, 1.00, 0.75, 0.00)
        CALL GSCR(IWKID, 7, 1.00, 0.50, 0.50)
        CALL GSCR(IWKID, 8, 1.00, 0.00, 0.00)
      ENDIF
C
      IERR = 0
C
      CALL CPSETI('CLS - CONTOUR LEVEL SELECTOR',0)
      CALL CPSETI('NCL - NUMBER OF CONTOUR LEVELS',7)
C
      DO 103 I=1,7
        CALL CPSETI('PAI - parameter array index',I)
        CALL CPSETR('CLV - contour level',10.*REAL(I))
```

```
          CALL CPSETI('CLU - contour level use',3)
          CALL CPSETI('LLC - contour label color',1)
  103 CONTINUE
C
C Initialize the drawing of the contour plot.
C
        CALL CPSETR('VPL - viewport left',0.05)
        CALL CPSETR('VPR - viewport right',0.95)
        CALL CPSETR('VPB - viewport bottom',0.05)
        CALL CPSETR('VPT - viewport top',0.95)
        CALL PCSETI('FN  - font number (Helvetica bold)' ,22)
        CALL PCSETI('CC  - font color',1)
        CALL CPSETR('T2D - tension of 2D splines',4.)
        CALL CPSETI('LLP - line label positioning, penalty scheme',3)
        CALL CPSETI('LLO - line label orientation',1)
        CALL CPSETC('LOT - low labels off',' ')
        CALL CPSETR('CWM - character width multiplier',2.5)
        CALL CPSETC('ILT - informational label off',' ')
        CALL CPRECT(ZDAT,NX,NX,NY,RWRK,2000,IWRK,1000)
C
C Initialize the area map and put the contour lines into it.
C
        CALL ARINAM (IAMA,20000)
        CALL CPCLAM (ZDAT,RWRK,IWRK,IAMA)
        CALL CPLBAM (ZDAT,RWRK,IWRK,IAMA)
C
C Color the map.
C
        CALL ARSCAM (IAMA,XCRA,YCRA,1000,IAIA,IGIA,7,CPCOLR)
C
C Put black contour lines over the colored map.
C
        CALL GSPLCI (1)
        CALL CPCLDM (ZDAT,RWRK,IWRK,IAMA,CPDRPL)
        CALL CPLBDR (ZDAT,RWRK,IWRK)
        CALL PERIM(1,0,1,0)
C
        CALL FRAME
C
C  Close down GKS.
C
        IF (ISTATE .EQ. 0) THEN
          CALL GDAWK (IWKID)
          CALL GCLWK (IWKID)
          CALL GCLKS
        ENDIF
C
        RETURN
        END
        SUBROUTINE CPCOLR (XCRA,YCRA,NCRA,IAIA,IGIA,NAIA)
C
        DIMENSION XCRA(*),YCRA(*),IAIA(*),IGIA(*)
C
        DO 102 I=1,NAIA
          IF (IGIA(I) .EQ. 3) IFLL = IAIA(I)
  102 CONTINUE
        IF (IFLL.GE.1 .AND. IFLL.LE.8) THEN
          CALL GSFACI (IFLL+1)
          CALL GFA (NCRA-1,XCRA,YCRA)
        END IF
C
```

```
      RETURN
      END
      SUBROUTINE DRWSRF(NX,NY,X,Y,Z,S1,S2,S3,IWK)
C
C  Procedure DRWSRF uses the NCAR Graphics function SRFACE to
C  draw a surface plot of the data values in Z.
C
C  The point of observation is calculated from the 3D coordinate
C  (S1, S2, S3); the point looked at is the center of the surface.
C
C   NX     -  Dimension of the X-axis variable X.
C   NY     -  Dimension of the Y-axis variable Y.
C   X      -  An array of X-axis values.
C   Y      -  An array of Y-axis values.
C   Z      -  An array dimensioned for NX x NY containing data
C             values for each (X,Y) coordinate.
C   S1     -  X value for the eye position.
C   S2     -  Y value for the eye position.
C   S3     -  Z value for the eye position.
C   IWK    -  Work space dimensioned for at least 2*NX*NY.
C
C
      DIMENSION X(NX),Y(NY),Z(NX,NY),IWK(*)
C
      PARAMETER (IERRF=6, LUNIT=2, IWKID=1, IWTYPE=8)
      DIMENSION S(6)
C
C  Open GKS, open and activate a workstation.
C
      JTYPE = IWTYPE
      CALL GQOPS(ISTATE)
      IF (ISTATE .EQ. 0) THEN
        CALL GOPKS (IERRF, ISZDM)
        IF (JTYPE .EQ. 1) THEN
          CALL NGSETC('ME','srf.ncgm')
        ELSE IF ( (JTYPE .GE. 20) .AND. (JTYPE .LE. 31) ) THEN
          CALL NGSETC('ME','srf.ps')
        ENDIF
        CALL GOPWK (IWKID, LUNIT, JTYPE)
        CALL GSCR(IWKID,0,1.,1.,1.)
        CALL GSCR(IWKID,1,0.,0.,0.)
        CALL GACWK (IWKID)
      ENDIF
C
C  Find the extreme values.
C
      XMX =  X(1)
      YMN =  Y(1)
      YMX =  Y(1)
      ZMN =  Z(1,1)
      ZMX =  Z(1,1)
C
      DO 10 I=2,NX
        XMN = MIN(XMN,X(I))
        XMX = MAX(XMX,X(I))
   10 CONTINUE
C
      DO 11 I=1,NY
        YMN = MIN(YMN,Y(I))
        YMX = MAX(YMX,Y(I))
   11 CONTINUE
```

```
C
      DO 12 I=1,NX
        DO 13 J=1,NY
          ZMN = MIN(ZMN,Z(I,J))
          ZMX = MAX(ZMX,Z(I,J))
   13   CONTINUE
   12 CONTINUE
C
      IF (S1.EQ.0. .AND. S2.EQ.0. .AND. S3.EQ.0.) THEN
        ST1 = -3.
        ST2 = -1.5
        ST3 = 0.75
      ELSE
        ST1 = S1
        ST2 = S2
        ST3 = S3
      ENDIF
      S(1) = 5.*ST1*(XMX-XMN)
      S(2) = 5.*ST2*(YMX-YMN)
      S(3) = 5.*ST3*(ZMX-ZMN)
      S(4) = 0.5*(XMX-XMN)
      S(5) = 0.5*(YMX-YMN)
      S(6) = 0.5*(ZMX-ZMN)
C
      CALL SRFACE (X,Y,Z,IWK,NX,NX,NY,S,0.)
C
C  Close down GKS.
C
      IF (ISTATE .EQ. 0) THEN
        CALL GDAWK (IWKID)
        CALL GCLWK (IWKID)
        CALL GCLKS
      ENDIF
C
      RETURN
      END
      SUBROUTINE DRWVCT(LX,LY,U,V)
C
C  Where U and V are 2D arrays, this subroutine uses NCAR Graphics to
C  draw a vector plot of the vectors (U(I,J),V(I,J))
C  for I=1,LX and J=1,LY.
C
      DIMENSION U(LX,LY),V(LX,LY)
      PARAMETER (IERRF=6, LUNIT=2, IWKID=1, IWTYPE=8)
C
      JTYPE = IWTYPE
      CALL GQOPS(ISTATE)
      IF (ISTATE .EQ. 0) THEN
        CALL GOPKS (IERRF, ISZDM)
        IF (JTYPE .EQ. 1) THEN
          CALL NGSETC('ME','vec.ncgm')
        ELSE IF ( (JTYPE .GE. 20) .AND. (JTYPE .LE. 31) ) THEN
          CALL NGSETC('ME','vec.ps')
        ENDIF
        CALL GOPWK (IWKID, LUNIT, JTYPE)
        CALL GACWK (IWKID)
        CALL GSCR(IWKID, 0, 1.00, 1.00, 1.00)
        CALL GSCR(IWKID, 1, 0.00, 0.00, 0.00)
      ENDIF
C
      CALL VVINIT(U,LX,V,LY,PDUM,1,LX,LY,WRK,1)
```

```
      CALL VVSETC('MNT',' ')
      CALL VVSETC('MXT',' ')
      CALL VVECTR(U,V,P,IAM,VVMSKD,WRK)
      CALL FRAME
C
      IF (ISTATE .EQ. 0) THEN
        CALL GDAWK (IWKID)
        CALL GCLWK (IWKID)
        CALL GCLKS
      ENDIF
C
      RETURN
      END
```

C functions for drawing example plots

The following code uses NCAR Graphics to produce surface plots, vector plots, and contour plots for the Natgrid examples.

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <ncarg/ncargC.h>
#include <ncarg/gks.h>

float armn(int, float *);
float armx(int, float *);

#if !defined (cray)
extern struct common {
#else
struct common {
#endif
    int ifr,istp,irots,idrx,idry,idrz,iupper,iskirt,ncla;
    float theta,hskirt,chi,clo,cinc;
    int ispval;
} NGCALLF(srfip1,SRFIP1);

/*
 *  Define the workstation type.  If not an X11 window, then
 *  appropritate metafile names will be generated for each plot
 *  in the subsequent code.
 */
#define IWTYPE 8
#define WKID   1

/*
 * Procedure drwsrfc uses the NCAR Graphics function c_srface to
 * draw a surface plot of the data values in z.
 *
 * The point of observation is calculated from the 3D coordinate
 * (s1, s2, s3); the point looked at is the center of the surface.
 *
 *  nx      -  Dimension of the X-axis variable x.
 *  ny      -  Dimension of the Y-axis variable y.
```

```
 *  x       -  An array of X-axis values.
 *  y       -  An array of Y-axis values.
 *  z       -  An array dimensioned for nx x ny containing data
 *             values for each (X,Y) coordinate.
 *  s1      -  X value for the eye position.
 *  s2      -  Y value for the eye position.
 *  s3      -  Z value for the eye position.
 *  skirt   -  Flags what type of skirt should be drawn:
 *               = -200.  A value is automatically calculated.
 *               = -100.  No skirt.
 *               = anthing else.  Value is used as the skirt value.
 *  iwk     -  Work space dimensioned for at least 2*nx*ny.
 *
 */

void drwsrfc (int nx, int ny, float *x, float *y, float *z,
              float s1, float s2, float s3, float skirt, int *iwk)
{

    Gop_st op_st;
    Gcolr_rep colval;
    float xmn, xmx, ymn, ymx, zmn, zmx, eye[6], *fz;
    int i,j;

/*
 *  If GKS is not open, open GKS, open workstation and activate a
 *  workstation.  Set the background color to white and the
 *  foreground color to balck.
 */
    ginq_op_st(&op_st);
    if (op_st == GST_GKCL) {
       gopen_gks ("stdout",0);
       if (IWTYPE == 1) {
         c_ngsetc("me","srf.ncgm");
       }
       else if ( (IWTYPE >= 20) && (IWTYPE <= 31) ) {
         c_ngsetc("me","srf.ps");
       }
       gopen_ws (WKID, NULL, IWTYPE);
       gactivate_ws (WKID);
       colval.rgb.red = 1.;
       colval.rgb.green = 1.;
       colval.rgb.blue = 1.;
       gset_colr_rep(WKID,0,&colval);
       colval.rgb.red = 0.;
       colval.rgb.green = 0.;
       colval.rgb.blue = 0.;
       gset_colr_rep(WKID,1,&colval);
    }

/*
 *  Find the extreme data values.
 */
    xmn = armn(nx, x);
    xmx = armx(nx, x);
    ymn = armn(ny, y);
    ymx = armx(ny, y);
    zmn = armn(nx * ny, z);
    zmx = armx(nx * ny, z);

    if ( (s1 == 0.) &&  (s2 == 0.) &&  (s3 == 0.) ) {
```

```
        s1 = -3.;
        s2 = -1.5;
        s3 = 0.75;
    }
    eye[0] = 5. * s1 * (xmx-xmn);
    eye[1] = 5. * s2 * (ymx-ymn);
    eye[2] = 5. * s3 * (zmx-zmn);
    eye[3] = 0.5 * (xmx-xmn);
    eye[4] = 0.5 * (ymx-ymn);
    eye[5] = 0.5 * (zmx-zmn);

    if (skirt == -200.) {
        NGCALLF(srfip1,SRFIP1).iskirt = 1;
        NGCALLF(srfip1,SRFIP1).hskirt = zmn-0.1*(zmx-zmn);
    }
    else if (skirt == -100.) {
        NGCALLF(srfip1,SRFIP1).iskirt = 0;
    }
    else {
        NGCALLF(srfip1,SRFIP1).iskirt = 1;
        NGCALLF(srfip1,SRFIP1).hskirt = skirt;
    }

/*
 *  Rearrange the array, since c_srface expects an array ordered as
 *  per Fortran.
 */
    fz = (float *) calloc(nx*ny,sizeof(float));

    for (i = 0; i < nx; i++) {
       for (j = 0; j < ny; j++) {

          fz[j*nx+i] = z[i*ny+j];
       }
    }

/*
 *  Plot the surface.
 */
    c_srface (x,y,fz,iwk,nx,nx,ny,eye,0.);
    free(fz);

/*
 *  Deactivate and close workstation, close GKS.
 */
    if (op_st == GST_GKCL) {
       gdeactivate_ws(WKID);
       gclose_ws(WKID);
       gclose_gks();
    }
}
float armn(int num, float *x)
{
   int i;
   float amin;
   amin = x[0];
   for (i = 1 ; i < num ; i++)
      if (x[i] < amin) amin = x[i];
   return(amin);
}
float armx(int num, float *x)
```

```
{
   int i;
   float amax;
   amax = x[0];
   for (i = 1 ; i < num ; i++)
      if (x[i] > amax) amax = x[i];
   return(amax);
}
/*
 * Procedure drwvctc uses the NCAR Graphics functions c_vvinit and
 * c_vvectr to draw a vector plot.
 *
 * Where u and v are 2D arrays, this function draws a vector plot
 * of the vectors (u[i,j],v[i,j]) for i=1,lx AND j=1,ly.
 *
 */

void drwvctc (int lx, int ly, float *u, float *v)
{

    Gop_st op_st;
    Gcolr_rep colval;
    float *fu, *fv;
    float p,wrk;
    int i,j,iam;

/*
 *  If GKS is not open, open GKS, open workstation and activate a
 *  workstation.  Set the background color to white and the
 *  foreground color to balck.
 */
    ginq_op_st(&op_st);
    if (op_st == GST_GKCL) {
       gopen_gks ("stdout",0);
       if (IWTYPE == 1) {
         c_ngsetc("me","vct.ncgm");
       }
       else if ( (IWTYPE >= 20) && (IWTYPE <= 31) ) {
         c_ngsetc("me","vct.ps");
       }
       gopen_ws (WKID, NULL, IWTYPE);
       gactivate_ws (WKID);
       colval.rgb.red = 1.;
       colval.rgb.green = 1.;
       colval.rgb.blue = 1.;
       gset_colr_rep(WKID,0,&colval);
       colval.rgb.red = 0.;
       colval.rgb.green = 0.;
       colval.rgb.blue = 0.;
       gset_colr_rep(WKID,1,&colval);
    }
/*
 *  Rearrange the arrays, since c_vvinit expects an array ordered as
 *  per Fortran.
 */
    fu = (float *) calloc(lx*ly,sizeof(float));
    fv = (float *) calloc(lx*ly,sizeof(float));

    for (i = 0; i < lx; i++) {
       for (j = 0; j < ly; j++) {
          fu[j*lx+i] = u[i*ly+j];
```

```
          fv[j*lx+i] = v[i*ly+j];
       }
    }
    c_vvinit(fu, lx, fv, ly, &p, 1, lx, ly, &wrk, 1);
    c_vvsetc("mnt"," ");
    c_vvsetc("mxt"," ");
    c_vvectr(fu, fv, &p, &iam, 0, &wrk);
    c_frame();
    free(fu);
    free(fv);

/*
 *  Deactivate and close workstation, close GKS.
 */
    if (op_st == GST_GKCL) {
       gdeactivate_ws(WKID);
       gclose_ws(WKID);
       gclose_gks();
    }
}

/*
 * Procedure drwconc uses NCAR Graphics functions to draw a
 * color contour plot.
 *
 * Where u is a 2D array, this function draws a contour plot
 * of u.
 *
 */

void drwconc (int lx, int ly, float *zdat)
{

    Gop_st op_st;
    Gcolr_rep colval;

    float xcra[1000],ycra[1000],rwrk[2000],*fz;
    int   iwrk[1000],iama[20000],iaia[10],igia[10],i,j;
    extern int cpcolr(float *, float *, int *, int *, int *, int *);
    extern int mask(float *, float *, int *, int *, int *, int *);

/*
 *  If GKS is not open, open GKS, open workstation and activate a
 *  workstation.  Set the background color to white and the
 *  foreground color to balck.
 */
    ginq_op_st(&op_st);
    if (op_st == GST_GKCL) {
       gopen_gks ("stdout",0);
       if (IWTYPE == 1) {
         c_ngsetc("me","con.ncgm");
       }
       else if ( (IWTYPE >= 20) && (IWTYPE <= 31) ) {
         c_ngsetc("me","con.ps");
       }
       gopen_ws (WKID, NULL, IWTYPE);
       gactivate_ws (WKID);
       colval.rgb.red = 1.;
       colval.rgb.green = 1.;
       colval.rgb.blue = 1.;
       gset_colr_rep(WKID,0,&colval);
```

```
        colval.rgb.red = 0.;
        colval.rgb.green = 0.;
        colval.rgb.blue = 0.;
        gset_colr_rep(WKID,1,&colval);
    }

/*
 *  Rearrange the array, since c_cprect expects an array ordered as
 *  per Fortran.
 */
    fz = (float *) calloc(lx*ly,sizeof(float));

    for (i = 0; i < lx; i++) {
       for (j = 0; j < ly; j++) {
          fz[j*lx+i] = zdat[i*ly+j];
       }
    }

    cpclrs();
    c_cpseti("CLS - contour level selector", 0);
    c_cpseti("NCL - number of contour levels", 7);

    for (i = 1 ; i < 8 ; i++) {
      c_cpseti("PAI - parameter array index", i);
      c_cpsetr("CLV - contour level", 10. * (float) i );
      c_cpseti("CLU - contour level use", 3);
      c_cpseti("LLC - contour label color", 1);
    }

    c_cpsetr("VPL - viewport left",0.05);
    c_cpsetr("VPR - viewport right",0.95);
    c_cpsetr("VPB - viewport bottom",0.05);
    c_cpsetr("VPT - viewport top",0.95);
    c_pcseti("FN  - font number (Helvetica bold)" ,22);
    c_pcseti("CC  - font color",1);
    c_cpsetr("T2D - tension of 2D splines",4.);
    c_cpseti("LLP - line label positioning, penalty scheme",3);
    c_cpseti("LLO - line label orientation",1);
    c_cpsetc("LOT - low labels off"," ");
    c_cpsetr("CWM - character width multiplier",2.5);
    c_cpsetc("ILT - informational label off"," ");

    c_cprect(fz,lx,lx,ly,rwrk,2000,iwrk,1000);
    c_arinam(iama,20000);
    c_cpclam(fz,rwrk,iwrk,iama);
    c_cplbam(fz,rwrk,iwrk,iama);

    c_arscam(iama,xcra,ycra,1000,iaia,igia,7,cpcolr);

    gset_line_colr_ind(1);

    c_cpcldm(fz,rwrk,iwrk,iama,mask);

    c_cplbdr(fz,rwrk,iwrk);
    gset_line_colr_ind(1);
    c_perim(1,0,1,0);
    c_frame();

    free(fz);

/*
```

```
 *  Deactivate and close workstation, close GKS.
 */
    if (op_st == GST_GKCL) {
       gdeactivate_ws(WKID);
       gclose_ws(WKID);
       gclose_gks();
    }
}

cpclrs()
{
    Gcolr_rep rgbv[9];
    int i;
/*
 * Define colors.
 */
    rgbv[0].rgb.red   = 1.00;
    rgbv[0].rgb.green = 1.00;
    rgbv[0].rgb.blue  = 1.00;
    rgbv[1].rgb.red   = 0.00;
    rgbv[1].rgb.green = 0.00;
    rgbv[1].rgb.blue  = 0.00;
    rgbv[2].rgb.red   = 0.00;
    rgbv[2].rgb.green = 1.00;
    rgbv[2].rgb.blue  = 1.00;
    rgbv[3].rgb.red   = 0.00;
    rgbv[3].rgb.green = 1.00;
    rgbv[3].rgb.blue  = 0.00;
    rgbv[4].rgb.red   = 0.70;
    rgbv[4].rgb.green = 1.00;
    rgbv[4].rgb.blue  = 0.00;
    rgbv[5].rgb.red   = 1.00;
    rgbv[5].rgb.green = 1.00;
    rgbv[5].rgb.blue  = 0.00;
    rgbv[6].rgb.red   = 1.00;
    rgbv[6].rgb.green = 0.75;
    rgbv[6].rgb.blue  = 0.00;
    rgbv[7].rgb.red   = 1.00;
    rgbv[7].rgb.green = 0.50;
    rgbv[7].rgb.blue  = 0.50;
    rgbv[8].rgb.red   = 1.00;
    rgbv[8].rgb.green = 0.00;
    rgbv[8].rgb.blue  = 0.00;

    for( i = 0; i < 9 ; i++ ) {
        gset_colr_rep(WKID,i,&rgbv[i]);
    }
    return(1);
}

int cpcolr(float *xcra, float *ycra, int *ncra, int *iaia,
           int *igia, int *naia)
{
    int i, ifll;
    Gpoint_list fill_area;

    for( i = 0; i <= *naia; i++ ) {
        if (igia[i] == 3) ifll=iaia[i];
    }
    if (ifll >= 1 && ifll <= 8) {
        gset_fill_colr_ind (ifll+1);
```

```
        fill_area.num_points = *ncra-1;
        fill_area.points =
           (Gpoint *) malloc(fill_area.num_points*sizeof(Gpoint));

        if( !fill_area.points ) {
           fprintf( stderr,
             "colram: Not enough memory to create fill area structure\n" );
           gemergency_close_gks();
           exit(1);
        }
        for( i = 0; i < *ncra-1; i++ ) {
          fill_area.points[i].x = xcra[i];
          fill_area.points[i].y = ycra[i];
        }
        gfill_area (&fill_area);
        free(fill_area.points);
    }

    return(1);
}

int mask(float *xwrk, float *ywrk, int *n, int *iarea, int *igrp,
         int *ngrps)
{
  int i,idr;

  idr = 1;

  for ( i = 0 ; i < *ngrps ; i++ ) {
    if (iarea[i] < 0) idr = 0;
  }

  if (idr != 0) c_curve(xwrk, ywrk, *n);

  return(1);
}
```

Example 1 - linear interpolation (Fortran single precision)

```
      PROGRAM NNEX01
C
C  Simple example of natural neighbor linear interpolation.
C
      PARAMETER(ISLIM = 6, NUMXOUT = 21, NUMYOUT = 21)
C
C  Dimension for the work space for the NCAR Graphics call to
C  SRFACE to plot the interpolated grid.
C
      PARAMETER(IDIM=2*NUMXOUT*NUMYOUT)
C
C  Input data arrays.
C
```

```
      REAL X(ISLIM), Y(ISLIM), Z(ISLIM)
C
C  Output grid arrays.
C
      REAL XI(NUMXOUT), YI(NUMYOUT), ZI(NUMXOUT,NUMYOUT)
C
C  Define the input data arrays.
C
      DATA X/0.00, 1.00, 0.00, 1.00, 0.40, 0.75 /
      DATA Y/0.00, 0.00, 1.00, 1.00, 0.20, 0.65 /
      DATA Z/0.00, 0.00, 0.00, 0.00, 1.25, 0.80 /
C
      DIMENSION IWORK(IDIM)
C
C  Define the output grid.
C
      XMIN = 0.
      XMAX = 1.
      XINC = (XMAX-XMIN)/(NUMXOUT-1.)
      DO 20 I=1,NUMXOUT
        XI(I) = XMIN+REAL(I-1) * XINC
   20 CONTINUE
C
      YMAX =  1.
      YMIN =  0.
      YINC = (YMAX-YMIN)/(NUMYOUT-1.)
      DO 30 J=1,NUMYOUT
        YI(J) = YMIN+REAL(J-1) * YINC
   30 CONTINUE
C
C  Do the gridding.
C
      CALL NATGRIDS(ISLIM,X,Y,Z,NUMXOUT,NUMYOUT,XI,YI,ZI,IER)
      IF (IER .NE. 0) THEN
        WRITE (6,510) IER
  510   FORMAT('Error return from NATGRIDS = ',I3)
      ENDIF
C
C  Draw a plot of the interpolated surface.
C
      CALL DRWSRF (NUMXOUT,NUMYOUT,XI,YI,ZI,15.,-25.,90.,IWORK)
C
      STOP
      END
```

Example 1 - linear interpolation (Fortran double precision)

```
      PROGRAM NNEX01
C
C  Simple example of natural neighbor linear interpolation.
C
      PARAMETER(ISLIM = 6, NUMXOUT = 21, NUMYOUT = 21)
```

```
C
C  Dimension for the work space for the NCAR Graphics call to
C  SRFACE to plot the interpolated grid.
C
      PARAMETER(IDIM=2*NUMXOUT*NUMYOUT)
C
C  Input data arrays.
C
      DOUBLE PRECISION X(ISLIM), Y(ISLIM), Z(ISLIM)
C
C  Output grid arrays.
C
      DOUBLE PRECISION XI(NUMXOUT), YI(NUMYOUT), ZI(NUMXOUT,NUMYOUT)
      REAL             XP(NUMXOUT), YP(NUMYOUT), ZP(NUMXOUT,NUMYOUT)
C
C  Define the input data arrays.
C
      DATA X/0.00, 1.00, 0.00, 1.00, 0.40, 0.75 /
      DATA Y/0.00, 0.00, 1.00, 1.00, 0.20, 0.65 /
      DATA Z/0.00, 0.00, 0.00, 0.00, 1.25, 0.80 /
C
      DIMENSION IWORK(IDIM)
C
C  Define the output grid.
C
      XMIN = 0.
      XMAX = 1.
      XINC = (XMAX-XMIN)/(NUMXOUT-1.)
      DO 20 I=1,NUMXOUT
        XI(I) = XMIN+REAL(I-1) * XINC
   20 CONTINUE
C
      YMAX =  1.
      YMIN =  0.
      YINC = (YMAX-YMIN)/(NUMYOUT-1.)
      DO 30 J=1,NUMYOUT
        YI(J) = YMIN+REAL(J-1) * YINC
   30 CONTINUE
C
C  Do the gridding.
C
      CALL NATGRIDD(ISLIM,X,Y,Z,NUMXOUT,NUMYOUT,XI,YI,ZI,IER)
      IF (IER .NE. 0) THEN
        WRITE (6,510) IER
  510   FORMAT('Error return from NATGRIDS = ',I3)
      ENDIF
C
C  Draw a plot of the interpolated surface.
C
      DO 333 I=1,NUMXOUT
        XP(I) = XI(I)
        DO 334 J=1,NUMYOUT
          YP(J) = YI(J)
          ZP(I,J) = ZI(I,J)
  334   CONTINUE
  333 CONTINUE
C
      CALL DRWSRF (NUMXOUT,NUMYOUT,XP,YP,ZP,15.,-25.,90.,IWORK)
C
      STOP
      END
```

Example 1 - linear interpolation (C single precision)

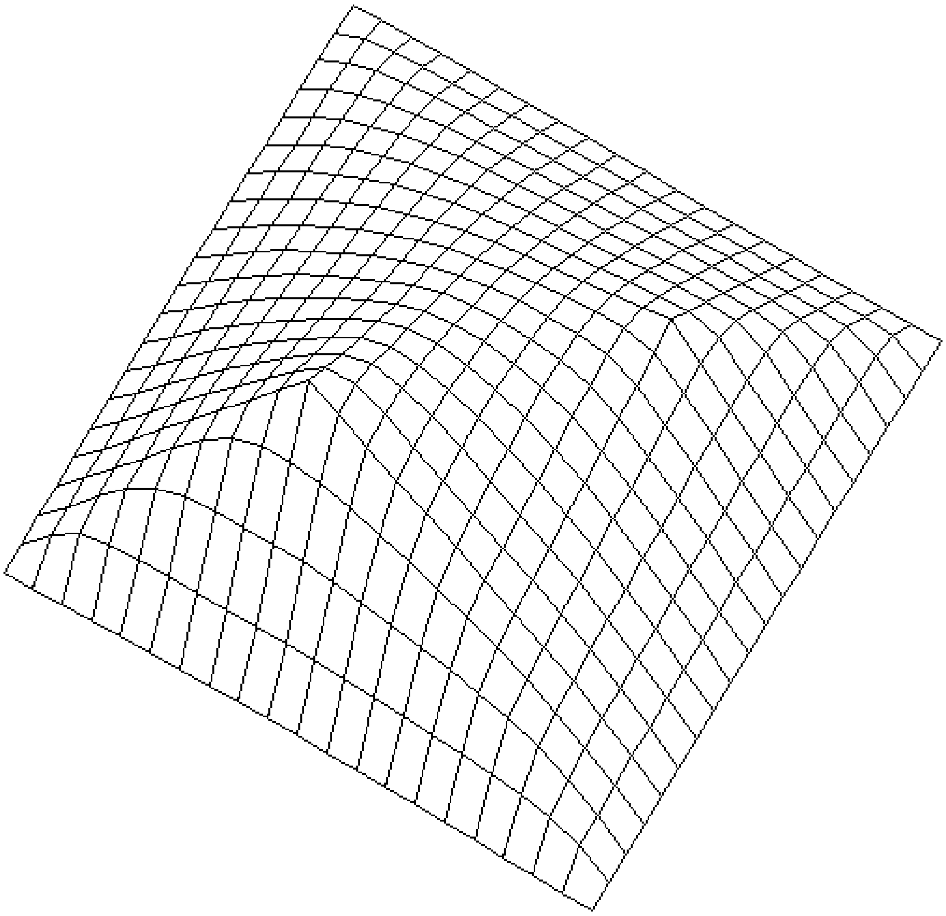
```
#include <stdio.h>

extern float *c_natgrids(int, float [], float [], float [],
                         int, int, float [], float [], int *);
extern void drwsrfc (int, int, float *, float *, float *,
                     float, float, float, int *);

#define NUMIN 6
#define NUMXOUT 21
#define NUMYOUT 21

main()
{
  int i, j, ier, iert, *iwk;

  float x[] = {0.00, 1.00, 0.00, 1.00, 0.40, 0.75},
        y[] = {0.00, 0.00, 1.00, 1.00, 0.20, 0.65},
        z[] = {0.00, 0.00, 0.00, 0.00, 1.25, 0.80};
  float *out, xo[NUMXOUT], yo[NUMYOUT], xc, yc;

  iwk = (int *) calloc(2*NUMXOUT*NUMYOUT,sizeof(float));

  xc = 1./(NUMXOUT-1.);
  for (i = 0 ; i < NUMXOUT ; i++) {
    xo[i] = i * xc;
  }

  yc = 1./(NUMYOUT-1.);
  for (j = 0 ; j < NUMYOUT ; j++) {
    yo[j] = j * yc;
  }

  out = c_natgrids(NUMIN, x, y, z, NUMXOUT, NUMYOUT, xo, yo, &ier);
  if (ier != 0) {
     printf (" Error return from c_natgrids = %d\n",ier);
  }

/*
 *  Draw the surface plot.
 */
  drwsrfc (NUMXOUT, NUMYOUT, xo, yo, out, 15.,-25.,90., iwk);

}
```

Example 1 - linear interpolation (C double

precision)

```
#include <stdio.h>

extern double *c_natgridd(int, double [], double [], double [],
                          int, int, double [], double [], int *);
extern void drwsrfc (int, int, float *, float *, float *,
                     float, float, float, int *);

#define NUMIN 6
#define NUMXOUT 21
#define NUMYOUT 21

main()
{
  int i, j, ier, iert, *iwk;

  double x[] = {0.00, 1.00, 0.00, 1.00, 0.40, 0.75},
         y[] = {0.00, 0.00, 1.00, 1.00, 0.20, 0.65},
         z[] = {0.00, 0.00, 0.00, 0.00, 1.25, 0.80};
  double *out, xo[NUMXOUT], yo[NUMYOUT], xc, yc;
  float outd[NUMXOUT * NUMYOUT], xp[NUMXOUT], yp[NUMYOUT];

  iwk = (int *) calloc(2*NUMXOUT*NUMYOUT,sizeof(float));

  xc = 1./(NUMXOUT-1.);
  for (i = 0 ; i < NUMXOUT ; i++) {
    xo[i] = i * xc;
  }

  yc = 1./(NUMYOUT-1.);
  for (j = 0 ; j < NUMYOUT ; j++) {
    yo[j] = j * yc;
  }

  out = c_natgridd(NUMIN, x, y, z, NUMXOUT, NUMYOUT, xo, yo, &ier);
  if (ier != 0) {
     printf (" Error return from c_natgrids = %d\n",ier);
  }

  for (i = 0; i < NUMXOUT; i++) {
    xp[i] = xo[i];
    for (j = 0; j < NUMYOUT; j++) {
      yp[j] = yo[j];
      outd[i*NUMYOUT+j] = (float) out[i*NUMYOUT+j];
    }
  }

/*
 *  Draw the surface plot.
 */
  drwsrfc (NUMXOUT, NUMYOUT, xp, yp, outd, 15.,-25.,90., iwk);

}
```

Plot for example 1 - linear interpolation



Example 2 - nonlinear interpolation (Fortran single precision)

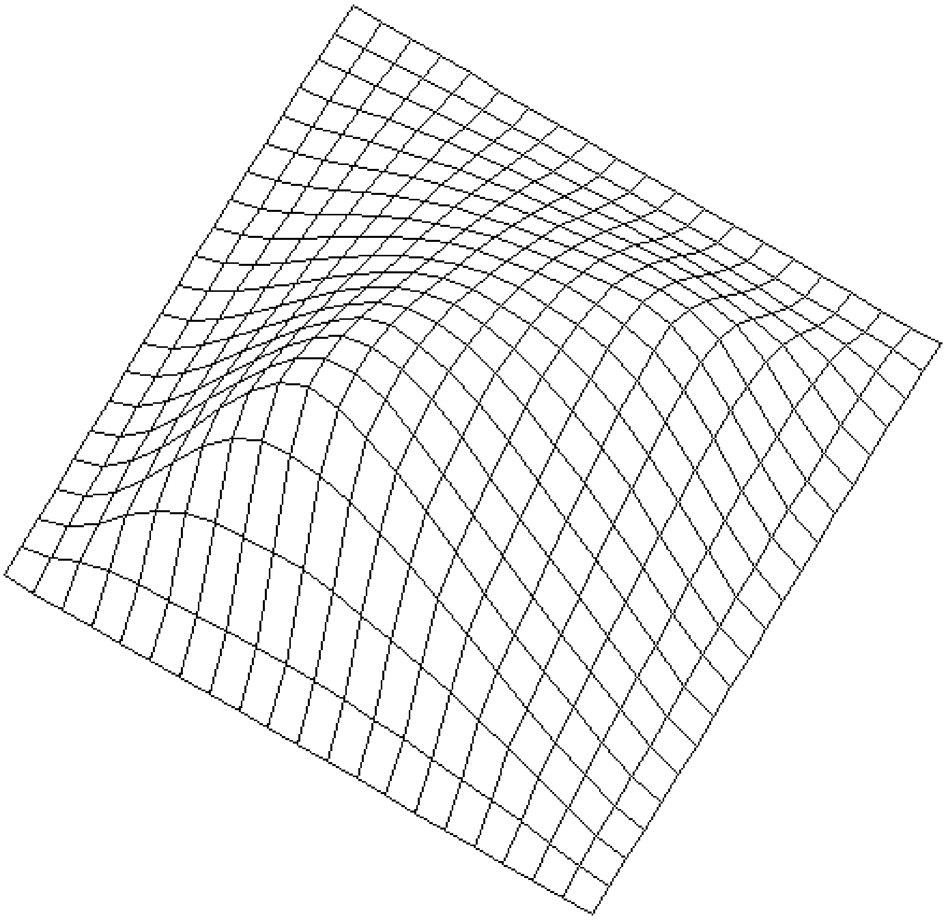
```
      PROGRAM NNEX02
C
C  Simple example of natural neighbor linear regridding.
C
      PARAMETER(ISLIM = 6, NUMXOUT = 21, NUMYOUT = 21)
C
C  Dimension for the work space for the NCAR Graphics call to
C  SRFACE to plot the interpolated grid.
C
      PARAMETER(IDIM=2*NUMXOUT*NUMYOUT)
```

```
C
C  Input data arrays.
C
      REAL X(ISLIM), Y(ISLIM), Z(ISLIM)
C
C  Output grid arrays.
C
      REAL XI(NUMXOUT), YI(NUMYOUT), ZI(NUMXOUT,NUMYOUT)
C
C  Define the input data arrays.
C
      DATA X/0.00, 1.00, 0.00, 1.00, 0.40, 0.75 /
      DATA Y/0.00, 0.00, 1.00, 1.00, 0.20, 0.65 /
      DATA Z/0.00, 0.00, 0.00, 0.00, 1.25, 0.80 /
C
      DIMENSION IWORK(IDIM)
C
C  Define the output grid.
C
      XMIN = 0.
      XMAX = 1.
      XINC = (XMAX-XMIN)/(NUMXOUT-1.)
      DO 20 I=1,NUMXOUT
        XI(I) = XMIN+REAL(I-1) * XINC
   20 CONTINUE
C
      YMAX =  1.
      YMIN =  0.
      YINC = (YMAX-YMIN)/(NUMYOUT-1.)
      DO 30 J=1,NUMYOUT
        YI(J) = YMIN+REAL(J-1) * YINC
   30 CONTINUE
C
C  Set the flag for using estimated gradients.
C
      CALL NNSETI('IGR',1)
C
C  Do the regridding.
C
      CALL NATGRIDS(ISLIM,X,Y,Z,NUMXOUT,NUMYOUT,XI,YI,ZI,IER)
      IF (IER .NE. 0) THEN
        WRITE (6,510) IER
  510   FORMAT('Error return from NATGRIDS = ',I3)
      ENDIF
C
C  Draw a plot of the interpolated surface.
C
      CALL DRWSRF(NUMXOUT,NUMYOUT,XI,YI,ZI,15.,-25.,90.,IWORK)
C
      STOP
      END
```

Example 2 - nonlinear interpolation (C single precision)

```
#include <stdio.h>

extern float *c_natgrids(int, float [], float [], float [],
                         int, int, float [], float [], int *);
extern void drwsrfc (int, int, float *, float *, float *,
                     float, float, float, int *);

#define NUMIN 6
#define NUMXOUT 21
#define NUMYOUT 21

main()
{
  int i, j, ier, iert, *iwk;

  float x[] = {0.00, 1.00, 0.00, 1.00, 0.40, 0.75},
        y[] = {0.00, 0.00, 1.00, 1.00, 0.20, 0.65},
        z[] = {0.00, 0.00, 0.00, 0.00, 1.25, 0.80};
  float *out, xo[NUMXOUT], yo[NUMYOUT], xc, yc;

  iwk = (int *) calloc(2*NUMXOUT*NUMYOUT,sizeof(float));

  xc = 1./(NUMXOUT-1.);
  for (i = 0 ; i < NUMXOUT ; i++) {
    xo[i] = i * xc;
  }

  yc = 1./(NUMYOUT-1.);
  for (j = 0 ; j < NUMYOUT ; j++) {
    yo[j] = j * yc;
  }

/*
 *  Set the flag to indicate use of gradient estimates.
 */
  c_nnseti("igr",1);

/*
 *  Do the regridding.
 */
  out = c_natgrids(NUMIN, x, y, z, NUMXOUT, NUMYOUT, xo, yo, &ier);
  if (ier != 0) {
     printf (" Error return from c_natgrids = %d\n",ier);
  }

/*
 *  Draw the surface plot.
 */
  drwsrfc (NUMXOUT, NUMYOUT, xo, yo, out, 15.,-25.,90., iwk);

}
```

Plot for example 2 - nonlinear interpolation



Example 3 - computing aspects and slopes (Fortran single precision)

```
      PROGRAM NNEX03
C
      PARAMETER(ISLIM = 171, NUMXOUT = 21, NUMYOUT = 21,
     +          IDIM=2*NUMXOUT*NUMYOUT)
      PARAMETER(RAD2DEG = 57.29578)
C
      REAL X(ISLIM), Y(ISLIM), Z(ISLIM)
      REAL XI(NUMXOUT), YI(NUMYOUT), ZI(NUMXOUT,NUMYOUT)
      REAL RTMP(NUMXOUT,NUMYOUT),U(NUMXOUT,NUMYOUT),V(NUMXOUT,NUMYOUT)
C
      INTEGER I,J,IWORK(IDIM)
C
C  The data values for X and Y (defined in BLOCKDATA) are random numbers
C  in the range  -0.2 to 1.2.  They are explicitly set so that they
C  will be uniform across all compilers.
C
```

```
      COMMON /DVALUE/X,Y
C
C  Define the function values.
C
      DO 5 I=1,ISLIM
        Z(I) = (X(I)-0.25)**2 + (Y(I)-0.50)**2
    5 CONTINUE
C
C  Define the output grid.
C
      XMIN = 0.0
      XMAX = 1.0
      XINC = (XMAX-XMIN)/(NUMXOUT-1.)
      DO 20 I=1,NUMXOUT
        XI(I) = XMIN+REAL(I-1) * XINC
   20 CONTINUE
C
      YMIN =  0.0
      YMAX =  1.0
      YINC = (YMAX-YMIN)/(NUMYOUT-1.)
      DO 30 J=1,NUMYOUT
        YI(J) = YMIN+REAL(J-1) * YINC
   30 CONTINUE
C
      CALL NNSETI('SDI - compute slopes and aspects',1)
      CALL NNSETI('IGR - use gradient estimates',1)
      CALL NNSETI('RAD - return results in radians',1)
      CALL NATGRIDS(ISLIM,X,Y,Z,NUMXOUT,NUMYOUT,XI,YI,ZI,IER)
C
C  Plot the interpolated surface.
C
      CALL DRWSRF(NUMXOUT,NUMYOUT,XI,YI,ZI,10.,-25.,50.,IWORK)
C
C  Get the aspects.
C
      DO 60 J=1,NUMYOUT
        DO 65 I=1,NUMXOUT
          CALL NNGETASPECTS(I,J,RTMP(I,J),IER)
   65   CONTINUE
   60 CONTINUE
C
      DO 100 I=1,NUMXOUT
        DO 101 J=1,NUMYOUT
          U(I,J) = SIN(RTMP(I,J))
          V(I,J) = COS(RTMP(I,J))
  101   CONTINUE
  100 CONTINUE
C
C  Plot the aspects as a vector plot.
C
      CALL DRWVCT(NUMXOUT,NUMYOUT,U,V)
C
C  Get the slopes; convert to degrees.
C
      DO 70 I=1,NUMXOUT
        DO 75 J=1,NUMYOUT
          CALL NNGETSLOPES(I,J,RTMP(I,J),IER)
          RTMP(I,J) = RAD2DEG*RTMP(I,J)
   75   CONTINUE
   70 CONTINUE
C
```

```
C  Plot the slopes as a contour plot.
C
      CALL DRWCON(NUMXOUT,NUMYOUT,XI,YI,RTMP,IWORK)
C
      STOP
      END
      BLOCKDATA DATAV
      COMMON /DVALUE/X,Y
      DIMENSION X(171),Y(171)
C
      DATA X/
     +    1.16,  0.47,  0.29,  0.72,  0.52,  1.12,  0.33,  0.20,  0.30,
     +    0.78,  0.92,  0.52,  0.44,  0.22, -0.10,  0.11,  0.59,  1.13,
     +    0.68,  1.11,  0.93,  0.29,  0.74,  0.43,  0.87,  0.87, -0.10,
     +    0.26,  0.85,  0.00, -0.02,  1.01, -0.12,  0.65,  0.39,  0.96,
     +    0.39,  0.38,  0.94, -0.03, -0.17,  0.00,  0.03,  0.67, -0.06,
     +    0.82, -0.03,  1.08,  0.37,  1.02, -0.11, -0.13,  1.03,  0.61,
     +    0.26,  0.18,  0.62,  0.42,  1.03,  0.72,  0.97,  0.08,  1.18,
     +    0.00,  0.69,  0.10,  0.80,  0.06,  0.82,  0.20,  0.46,  0.37,
     +    1.16,  0.93,  1.09,  0.96,  1.00,  0.80,  0.01,  0.12,  1.01,
     +    0.48,  0.79,  0.04,  0.42,  0.48, -0.18,  1.16,  0.85,  0.97,
     +    0.14,  0.40,  0.78,  1.12,  1.19,  0.68,  0.65,  0.41,  0.90,
     +    0.84, -0.11, -0.01, -0.02, -0.10,  1.04,  0.58,  0.61,  0.12,
     +   -0.02, -0.03,  0.27,  1.17,  1.02,  0.16, -0.17,  1.03,  0.13,
     +    0.04, -0.03,  0.15,  0.00, -0.01,  0.91,  1.20,  0.54, -0.14,
     +    1.03,  0.93,  0.42,  0.36, -0.10,  0.57,  0.22,  0.74,  1.15,
     +    0.40,  0.82,  0.96,  1.09,  0.42,  1.13,  0.24,  0.51,  0.60,
     +    0.06,  0.38,  0.15,  0.59,  0.76,  1.16,  0.02,  0.86,  1.14,
     +    0.37,  0.38,  0.26,  0.26,  0.07,  0.87,  0.90,  0.83,  0.09,
     +    0.03,  0.56, -0.19,  0.51,  1.07, -0.13,  0.99,  0.84,  0.22/
      DATA Y/
     +   -0.11,  1.07,  1.11, -0.17,  0.08,  0.09,  0.91,  0.17, -0.02,
     +    0.83,  1.08,  0.87,  0.46,  0.66,  0.50, -0.14,  0.78,  1.08,
     +    0.65,  0.00,  1.03,  0.06,  0.69, -0.16,  0.02,  0.59,  0.19,
     +    0.54,  0.68,  0.95,  0.30,  0.77,  0.94,  0.76,  0.56,  0.12,
     +    0.05, -0.07,  1.01,  0.61,  1.04, -0.07,  0.46,  1.07,  0.87,
     +    0.11,  0.63,  0.06,  0.53,  0.95,  0.78,  0.48,  0.45,  0.77,
     +    0.78,  0.29,  0.38,  0.85, -0.10,  1.17,  0.35,  1.14, -0.04,
     +    0.34, -0.18,  0.78,  0.17,  0.63,  0.88, -0.12,  0.58, -0.12,
     +    1.00,  0.99,  0.45,  0.86, -0.15,  0.97,  0.99,  0.90,  0.42,
     +    0.61,  0.74,  0.41,  0.44,  1.08,  1.06,  1.18,  0.89,  0.74,
     +    0.74, -0.06,  0.00,  0.99,  0.03,  1.00, -0.04,  0.24,  0.65,
     +    0.12,  0.13, -0.09, -0.05,  1.03,  1.07, -0.02,  1.18,  0.19,
     +    0.03, -0.03,  0.86,  1.12,  0.38,  0.72, -0.20, -0.08, -0.18,
     +    0.32,  0.13, -0.19,  0.93,  0.81,  0.31,  1.09, -0.03,  1.01,
     +   -0.17,  0.84, -0.11,  0.45,  0.18,  0.23,  0.81,  0.39,  1.09,
     +   -0.05,  0.58,  0.53,  0.96,  0.43,  0.48,  0.96, -0.03,  1.13,
     +    1.16,  0.16,  1.15,  0.57,  0.13,  0.71,  0.35,  1.04,  0.62,
     +    1.03,  0.98,  0.31,  0.70,  0.97,  0.87,  1.14,  0.08,  1.19,
     +    0.88,  1.00,  0.51,  0.03,  0.17,  1.01,  0.44,  0.17, -0.11/
C
      END
```

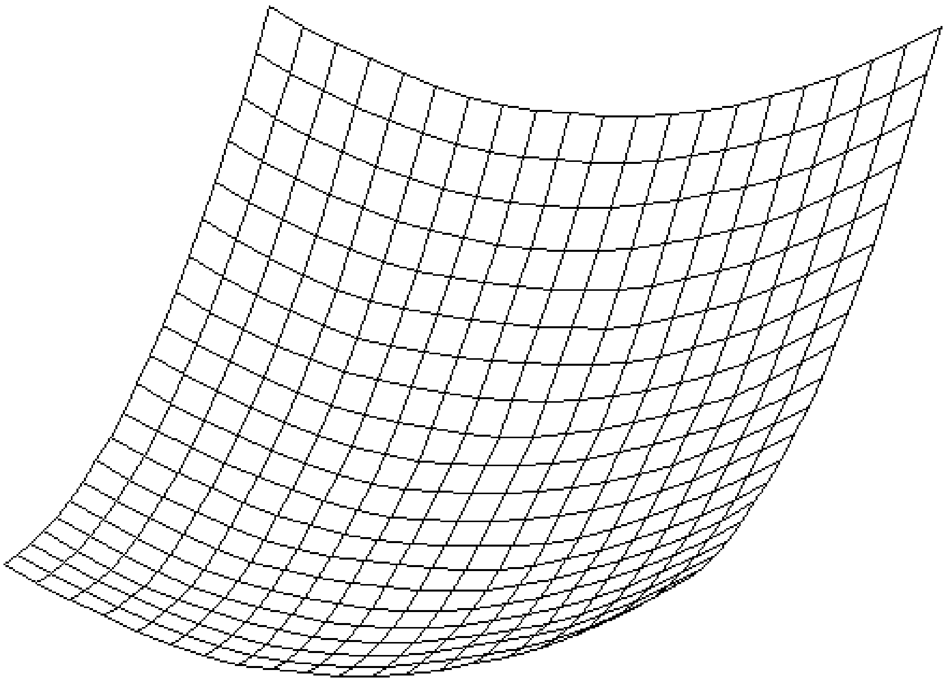
Example 3 - computing aspects and slopes (C single precision)

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

extern float *c_natgrids(int, float [], float [], float [],
                         int, int, float [], float [], int *);
extern void drwsrfc (int, int, float *, float *, float *,
              float, float, float, int *);
extern void drwvctc (int, int, float *, float*);
extern void drwconc (int, int, float *);

#define NUMIN 171
#define NUMXOUT 21
#define NUMYOUT 21
#define RAD2DEG 57.29578

main()
{
  int i, j, ier, iert, *iwk;

/*
 *  Coordinate data are defined as random numbers between
 *  -0.2 and 1.2. and are explicitly defined here for uniformity
 *  across platforms.
 */
float x[] = {
             1.16,  0.47,  0.29,  0.72,  0.52,  1.12,  0.33,  0.20,  0.30,
             0.78,  0.92,  0.52,  0.44,  0.22, -0.10,  0.11,  0.59,  1.13,
             0.68,  1.11,  0.93,  0.29,  0.74,  0.43,  0.87,  0.87, -0.10,
             0.26,  0.85,  0.00, -0.02,  1.01, -0.12,  0.65,  0.39,  0.96,
             0.39,  0.38,  0.94, -0.03, -0.17,  0.00,  0.03,  0.67, -0.06,
             0.82, -0.03,  1.08,  0.37,  1.02, -0.11, -0.13,  1.03,  0.61,
             0.26,  0.18,  0.62,  0.42,  1.03,  0.72,  0.97,  0.08,  1.18,
             0.00,  0.69,  0.10,  0.80,  0.06,  0.82,  0.20,  0.46,  0.37,
             1.16,  0.93,  1.09,  0.96,  1.00,  0.80,  0.01,  0.12,  1.01,
             0.48,  0.79,  0.04,  0.42,  0.48, -0.18,  1.16,  0.85,  0.97,
             0.14,  0.40,  0.78,  1.12,  1.19,  0.68,  0.65,  0.41,  0.90,
             0.84, -0.11, -0.01, -0.02, -0.10,  1.04,  0.58,  0.61,  0.12,
            -0.02, -0.03,  0.27,  1.17,  1.02,  0.16, -0.17,  1.03,  0.13,
             0.04, -0.03,  0.15,  0.00, -0.01,  0.91,  1.20,  0.54, -0.14,
             1.03,  0.93,  0.42,  0.36, -0.10,  0.57,  0.22,  0.74,  1.15,
             0.40,  0.82,  0.96,  1.09,  0.42,  1.13,  0.24,  0.51,  0.60,
             0.06,  0.38,  0.15,  0.59,  0.76,  1.16,  0.02,  0.86,  1.14,
             0.37,  0.38,  0.26,  0.26,  0.07,  0.87,  0.90,  0.83,  0.09,
             0.03,  0.56, -0.19,  0.51,  1.07, -0.13,  0.99,  0.84,  0.22
            };
float y[] = {
            -0.11,  1.07,  1.11, -0.17,  0.08,  0.09,  0.91,  0.17, -0.02,
             0.83,  1.08,  0.87,  0.46,  0.66,  0.50, -0.14,  0.78,  1.08,
             0.65,  0.00,  1.03,  0.06,  0.69, -0.16,  0.02,  0.59,  0.19,
             0.54,  0.68,  0.95,  0.30,  0.77,  0.94,  0.76,  0.56,  0.12,
             0.05, -0.07,  1.01,  0.61,  1.04, -0.07,  0.46,  1.07,  0.87,
             0.11,  0.63,  0.06,  0.53,  0.95,  0.78,  0.48,  0.45,  0.77,
             0.78,  0.29,  0.38,  0.85, -0.10,  1.17,  0.35,  1.14, -0.04,
             0.34, -0.18,  0.78,  0.17,  0.63,  0.88, -0.12,  0.58, -0.12,
             1.00,  0.99,  0.45,  0.86, -0.15,  0.97,  0.99,  0.90,  0.42,
             0.61,  0.74,  0.41,  0.44,  1.08,  1.06,  1.18,  0.89,  0.74,
             0.74, -0.06,  0.00,  0.99,  0.03,  1.00, -0.04,  0.24,  0.65,
```

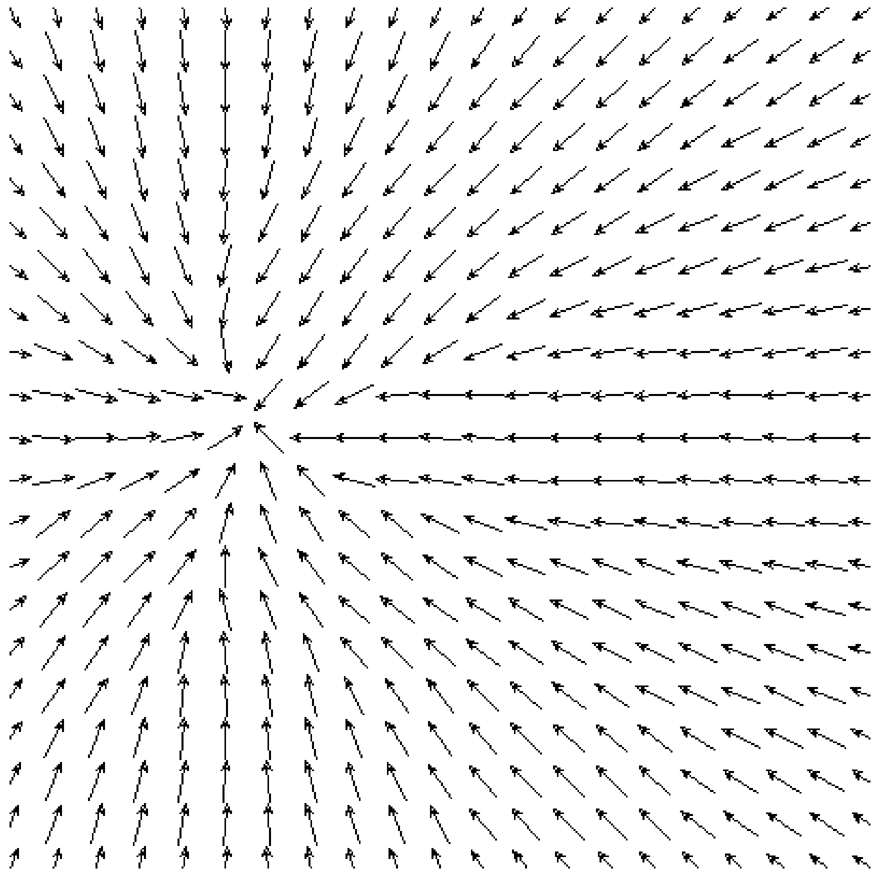
```
           0.12,  0.13, -0.09, -0.05,  1.03,  1.07, -0.02,  1.18,  0.19,
           0.03, -0.03,  0.86,  1.12,  0.38,  0.72, -0.20, -0.08, -0.18,
           0.32,  0.13, -0.19,  0.93,  0.81,  0.31,  1.09, -0.03,  1.01,
          -0.17,  0.84, -0.11,  0.45,  0.18,  0.23,  0.81,  0.39,  1.09,
          -0.05,  0.58,  0.53,  0.96,  0.43,  0.48,  0.96, -0.03,  1.13,
           1.16,  0.16,  1.15,  0.57,  0.13,  0.71,  0.35,  1.04,  0.62,
           1.03,  0.98,  0.31,  0.70,  0.97,  0.87,  1.14,  0.08,  1.19,
           0.88,  1.00,  0.51,  0.03,  0.17,  1.01,  0.44,  0.17, -0.11
          };

  float z[NUMIN];
  float *out, xo[NUMXOUT], yo[NUMYOUT], xc, yc;
  float u[NUMXOUT][NUMYOUT], v[NUMXOUT][NUMYOUT], uvtmp;

  iwk = (int *) calloc(2*NUMXOUT*NUMYOUT,sizeof(float));

  for (i = 0 ; i < NUMIN ; i++) {
    z[i] = (x[i]-0.25)*(x[i]-0.25) + (y[i]-0.50)*(y[i]-0.50);
  }

  xc = 1./(NUMXOUT-1.);
  for (i = 0 ; i < NUMXOUT ; i++) {
    xo[i] = i * xc;
  }

  yc = 1./(NUMYOUT-1.);
  for (j = 0 ; j < NUMYOUT ; j++) {
    yo[j] = j * yc;
  }

/*
 *  Turn on gradient estimate calculations, flag calculation of
 *  aspects and slopes, set flag to return aspects and slopes in
 *  radians.
 */
  c_nnseti("igr",1);
  c_nnseti("sdi",1);
  c_nnseti("rad",1);

/*
 *  Do the interpolation.
 */
  out = c_natgrids(NUMIN, x, y, z, NUMXOUT, NUMYOUT, xo, yo, &ier);
  if (ier != 0) {
     printf (" Error return from c_natgrids = %d\n",ier);
  }

/*
 *  Draw the surface plot.
 */
  drwsrfc (NUMXOUT, NUMYOUT, xo, yo, out, 15.,-25.,90., iwk);

/*
 *  Get the aspects.
 */
  for (i = 0 ; i < NUMXOUT ; i++) {
    for (j = 0 ; j < NUMYOUT ; j++) {
      c_nngetaspects(i, j, &uvtmp, &ier);
      u[i][j] = sin(uvtmp);
      v[i][j] = cos(uvtmp);
    }
```

```
  }

/*
 *  Plot the aspects as a vector plot.
 */
  drwvctc(NUMXOUT, NUMYOUT, &u[0][0], &v[0][0]);

/*
 *  Get the slopes.
 */
  for (i = 0 ; i < NUMXOUT ; i++) {
    for (j = 0 ; j < NUMYOUT ; j++) {
      c_nngetslopes(i, j, &uvtmp, &ier);
      u[i][j] = RAD2DEG*uvtmp;
    }
  }

/*
 *  Plot the slopes as a contour plot.
 */
  drwconc(NUMXOUT, NUMYOUT, &u[0][0]);

}
```

Functional surface plot for example 3

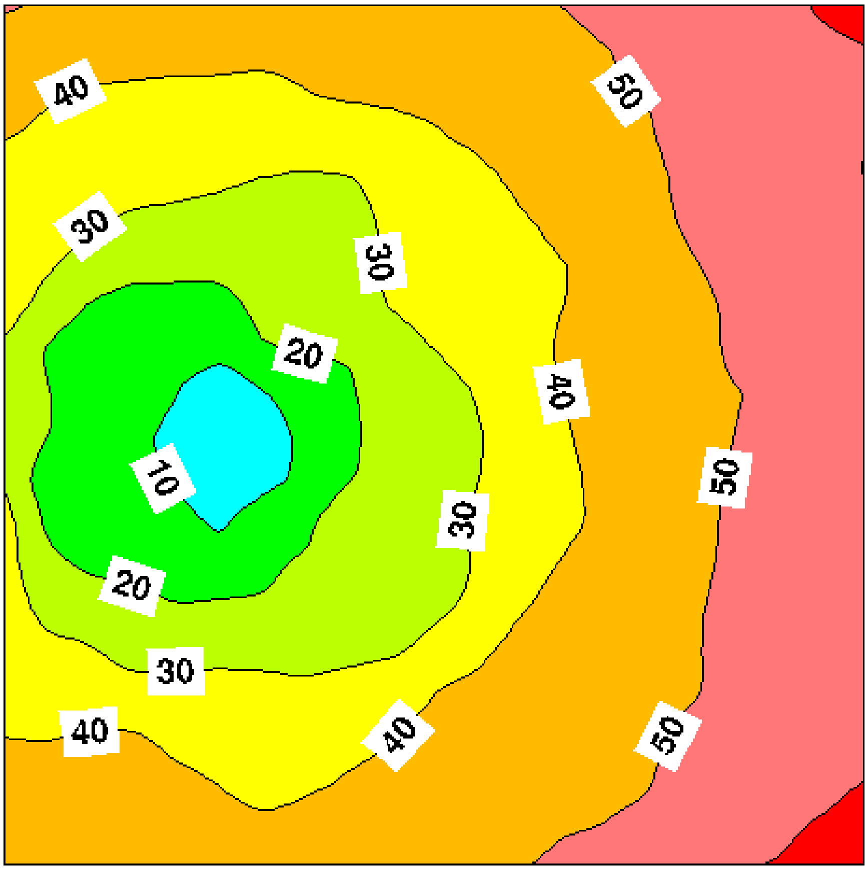


Vector plot of aspects for example 3



Contour plot of slopes for example 3

In the following plot, the slopes are plotted in degrees.



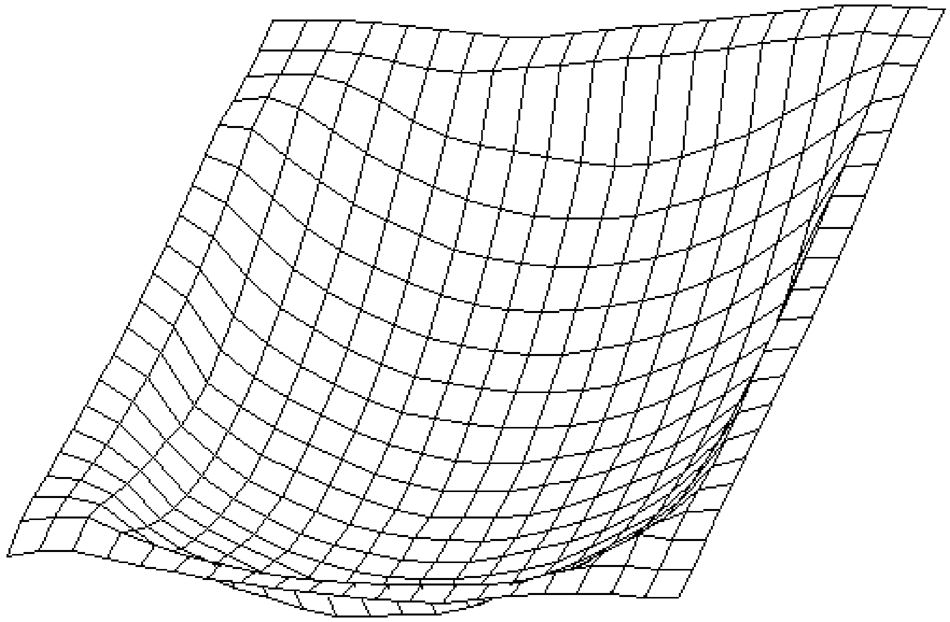
Example 4 - extrapolation (Fortran single precision)

```
      PROGRAM NNEX04
C
C  Code to illustrate the extrapolation features of the natural
C  neighbor regridding package.  The only differecne between the
C  code used with extrapolation allowed and extrapolation not
C  allowed is the setting of the value of the parameter "ext".
C
      PARAMETER(ISLIM = 171, NUMXOUT = 21, NUMYOUT = 21,
     +          IDIM=2*NUMXOUT*NUMYOUT)
      PARAMETER(RAD2DEG = 57.29578)
C
      REAL X(ISLIM), Y(ISLIM), Z(ISLIM)
      REAL XI(NUMXOUT), YI(NUMYOUT), ZI(NUMXOUT,NUMYOUT)
C
      INTEGER I,J,IWORK(IDIM)
C
```

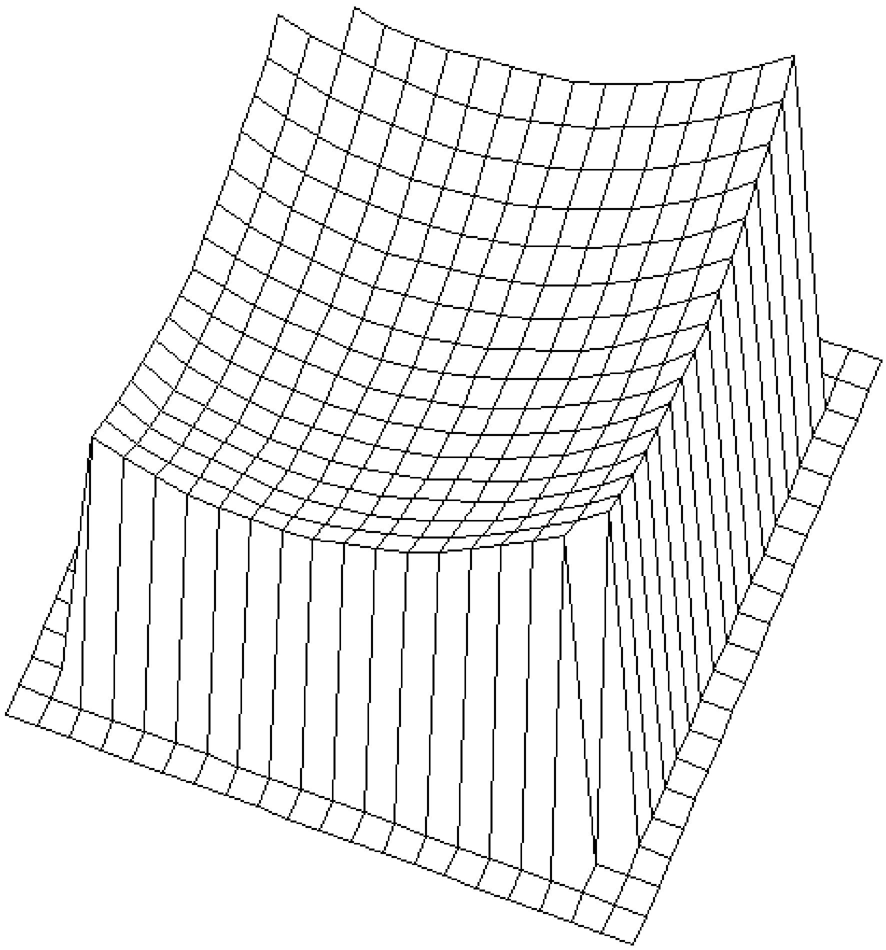
```
C  The data values for X and Y (defined in BLOCKDATA) are random numbers
C  in the range  -0.2 to 1.2.  They are explicitly set so that they
C  will be uniform across all compilers.
C
      COMMON /DVALUE/X,Y
C
C  Define the function values.
C
      DO 5 I=1,ISLIM
        Z(I) = (X(I)-0.25)**2 + (Y(I)-0.50)**2
    5 CONTINUE
C
C  Define the output grid.
C
      XMIN = -0.3
      XMAX =  1.3
      XINC = (XMAX-XMIN)/(NUMXOUT-1.)
      DO 20 I=1,NUMXOUT
        XI(I) = XMIN+REAL(I-1) * XINC
   20 CONTINUE
C
      YMIN = -0.3
      YMAX =  1.3
      YINC = (YMAX-YMIN)/(NUMYOUT-1.)
      DO 30 J=1,NUMYOUT
        YI(J) = YMIN+REAL(J-1) * YINC
   30 CONTINUE
C
      CALL NNSETI('EXT - extrapolation control',1)
      CALL NNSETR('NUL - function value when extrapolation not allowed',
     +             -1.)
      CALL NATGRIDS(ISLIM,X,Y,Z,NUMXOUT,NUMYOUT,XI,YI,ZI,IER)
C
C  Plot the interpolated surface.
C
      CALL DRWSRF(NUMXOUT,NUMYOUT,XI,YI,ZI,10.,-25.,50.,IWORK)
C
      STOP
      END
      BLOCKDATA DATAV
      COMMON /DVALUE/X,Y
      DIMENSION X(171),Y(171)
C
      DATA X/
     +    1.16,  0.47,  0.29,  0.72,  0.52,  1.12,  0.33,  0.20,  0.30,
     +    0.78,  0.92,  0.52,  0.44,  0.22, -0.10,  0.11,  0.59,  1.13,
     +    0.68,  1.11,  0.93,  0.29,  0.74,  0.43,  0.87,  0.87, -0.10,
     +    0.26,  0.85,  0.00, -0.02,  1.01, -0.12,  0.65,  0.39,  0.96,
     +    0.39,  0.38,  0.94, -0.03, -0.17,  0.00,  0.03,  0.67, -0.06,
     +    0.82, -0.03,  1.08,  0.37,  1.02, -0.11, -0.13,  1.03,  0.61,
     +    0.26,  0.18,  0.62,  0.42,  1.03,  0.72,  0.97,  0.08,  1.18,
     +    0.00,  0.69,  0.10,  0.80,  0.06,  0.82,  0.20,  0.46,  0.37,
     +    1.16,  0.93,  1.09,  0.96,  1.00,  0.80,  0.01,  0.12,  1.01,
     +    0.48,  0.79,  0.04,  0.42,  0.48, -0.18,  1.16,  0.85,  0.97,
     +    0.14,  0.40,  0.78,  1.12,  1.19,  0.68,  0.65,  0.41,  0.90,
     +    0.84, -0.11, -0.01, -0.02, -0.10,  1.04,  0.58,  0.61,  0.12,
     +   -0.02, -0.03,  0.27,  1.17,  1.02,  0.16, -0.17,  1.03,  0.13,
     +    0.04, -0.03,  0.15,  0.00, -0.01,  0.91,  1.20,  0.54, -0.14,
     +    1.03,  0.93,  0.42,  0.36, -0.10,  0.57,  0.22,  0.74,  1.15,
     +    0.40,  0.82,  0.96,  1.09,  0.42,  1.13,  0.24,  0.51,  0.60,
     +    0.06,  0.38,  0.15,  0.59,  0.76,  1.16,  0.02,  0.86,  1.14,
```

```
     +    0.37,  0.38,  0.26,  0.26,  0.07,  0.87,  0.90,  0.83,  0.09,
     +    0.03,  0.56, -0.19,  0.51,  1.07, -0.13,  0.99,  0.84,  0.22/
      DATA Y/
     +   -0.11,  1.07,  1.11, -0.17,  0.08,  0.09,  0.91,  0.17, -0.02,
     +    0.83,  1.08,  0.87,  0.46,  0.66,  0.50, -0.14,  0.78,  1.08,
     +    0.65,  0.00,  1.03,  0.06,  0.69, -0.16,  0.02,  0.59,  0.19,
     +    0.54,  0.68,  0.95,  0.30,  0.77,  0.94,  0.76,  0.56,  0.12,
     +    0.05, -0.07,  1.01,  0.61,  1.04, -0.07,  0.46,  1.07,  0.87,
     +    0.11,  0.63,  0.06,  0.53,  0.95,  0.78,  0.48,  0.45,  0.77,
     +    0.78,  0.29,  0.38,  0.85, -0.10,  1.17,  0.35,  1.14, -0.04,
     +    0.34, -0.18,  0.78,  0.17,  0.63,  0.88, -0.12,  0.58, -0.12,
     +    1.00,  0.99,  0.45,  0.86, -0.15,  0.97,  0.99,  0.90,  0.42,
     +    0.61,  0.74,  0.41,  0.44,  1.08,  1.06,  1.18,  0.89,  0.74,
     +    0.74, -0.06,  0.00,  0.99,  0.03,  1.00, -0.04,  0.24,  0.65,
     +    0.12,  0.13, -0.09, -0.05,  1.03,  1.07, -0.02,  1.18,  0.19,
     +    0.03, -0.03,  0.86,  1.12,  0.38,  0.72, -0.20, -0.08, -0.18,
     +    0.32,  0.13, -0.19,  0.93,  0.81,  0.31,  1.09, -0.03,  1.01,
     +   -0.17,  0.84, -0.11,  0.45,  0.18,  0.23,  0.81,  0.39,  1.09,
     +   -0.05,  0.58,  0.53,  0.96,  0.43,  0.48,  0.96, -0.03,  1.13,
     +    1.16,  0.16,  1.15,  0.57,  0.13,  0.71,  0.35,  1.04,  0.62,
     +    1.03,  0.98,  0.31,  0.70,  0.97,  0.87,  1.14,  0.08,  1.19,
     +    0.88,  1.00,  0.51,  0.03,  0.17,  1.01,  0.44,  0.17, -0.11/
C
      END
```

Example 4 - extrapolation allowed



Example 4 - extrapolation not allowed



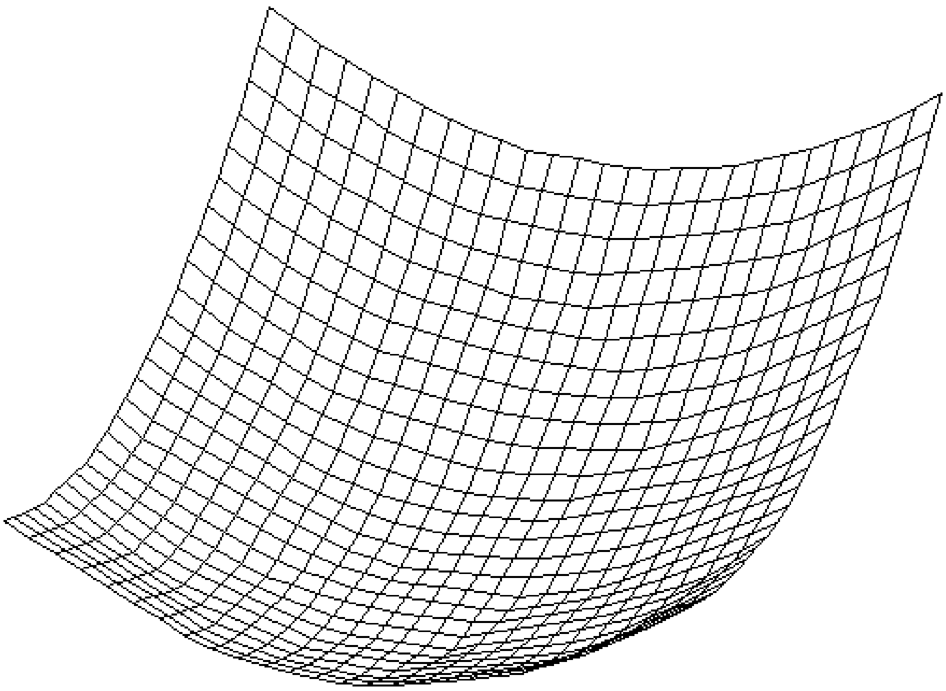
Example 5 - negative values disallowed (Fortran single precision)

```
      PROGRAM NNEX05
C
C  Code to illustrate the negative value feature.
C
      PARAMETER(ISLIM = 171, NUMXOUT = 27, NUMYOUT = 27,
     +          IDIM=2*NUMXOUT*NUMYOUT)
C
      REAL X(ISLIM), Y(ISLIM), Z(ISLIM)
      REAL XI(NUMXOUT), YI(NUMYOUT), ZI(NUMXOUT,NUMYOUT)
C
      INTEGER I,J,IWORK(IDIM)
C
C  The data values for X and Y (defined in BLOCKDATA) are random numbers
C  in the range  -0.2 to 1.2.  They are explicitly set so that they
C  will be uniform across all compilers.
```

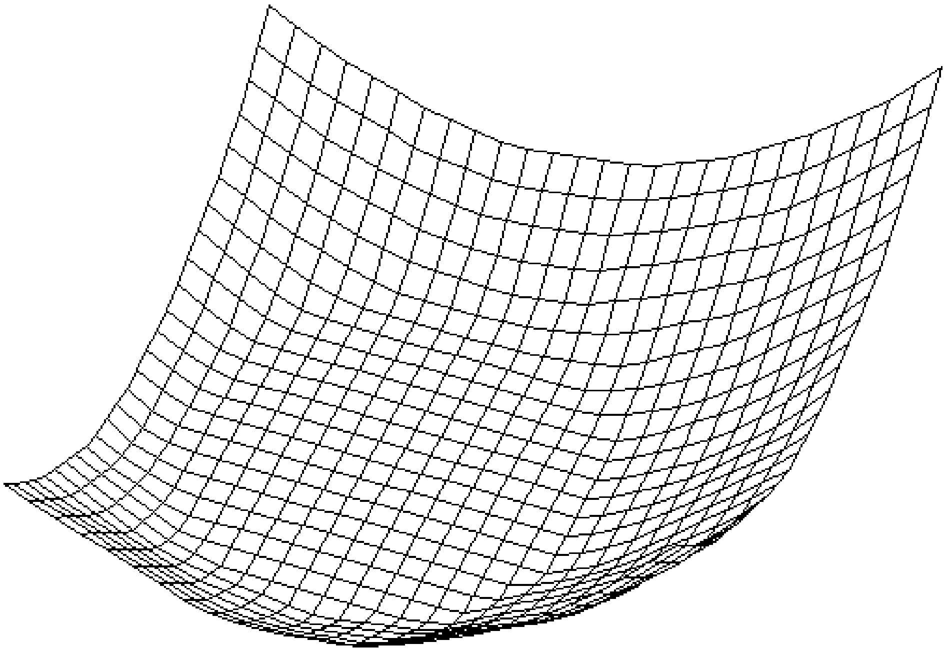
```
C
      COMMON /DVALUE/X,Y
C
C  Define the function values.
C
      DO 5 I=1,ISLIM
        Z(I) = (X(I)-0.375)**2 + (Y(I)-0.50)**2 - 0.1
    5 CONTINUE
C
C  Define the output grid.
C
      XMIN =  0.0
      XMAX =  1.0
      XINC = (XMAX-XMIN)/(NUMXOUT-1.)
      DO 20 I=1,NUMXOUT
        XI(I) = XMIN+REAL(I-1) * XINC
   20 CONTINUE
C
      YMIN =  0.0
      YMAX =  1.0
      YINC = (YMAX-YMIN)/(NUMYOUT-1.)
      DO 30 J=1,NUMYOUT
        YI(J) = YMIN+REAL(J-1) * YINC
   30 CONTINUE
C
      CALL NNSETI('NON - control of negative values',0)
      CALL NATGRIDS(ISLIM,X,Y,Z,NUMXOUT,NUMYOUT,XI,YI,ZI,IER)
C
C  Plot the interpolated surface.
C
      CALL DRWSRF(NUMXOUT,NUMYOUT,XI,YI,ZI,10.,-25.,50.,IWORK)
C
      STOP
      END
      BLOCKDATA DATAV
      COMMON /DVALUE/X,Y
      DIMENSION X(171),Y(171)
C
      DATA X/
     +    1.16,  0.47,  0.29,  0.72,  0.52,  1.12,  0.33,  0.20,  0.30,
     +    0.78,  0.92,  0.52,  0.44,  0.22, -0.10,  0.11,  0.59,  1.13,
     +    0.68,  1.11,  0.93,  0.29,  0.74,  0.43,  0.87,  0.87, -0.10,
     +    0.26,  0.85,  0.00, -0.02,  1.01, -0.12,  0.65,  0.39,  0.96,
     +    0.39,  0.38,  0.94, -0.03, -0.17,  0.00,  0.03,  0.67, -0.06,
     +    0.82, -0.03,  1.08,  0.37,  1.02, -0.11, -0.13,  1.03,  0.61,
     +    0.26,  0.18,  0.62,  0.42,  1.03,  0.72,  0.97,  0.08,  1.18,
     +    0.00,  0.69,  0.10,  0.80,  0.06,  0.82,  0.20,  0.46,  0.37,
     +    1.16,  0.93,  1.09,  0.96,  1.00,  0.80,  0.01,  0.12,  1.01,
     +    0.48,  0.79,  0.04,  0.42,  0.48, -0.18,  1.16,  0.85,  0.97,
     +    0.14,  0.40,  0.78,  1.12,  1.19,  0.68,  0.65,  0.41,  0.90,
     +    0.84, -0.11, -0.01, -0.02, -0.10,  1.04,  0.58,  0.61,  0.12,
     +   -0.02, -0.03,  0.27,  1.17,  1.02,  0.16, -0.17,  1.03,  0.13,
     +    0.04, -0.03,  0.15,  0.00, -0.01,  0.91,  1.20,  0.54, -0.14,
     +    1.03,  0.93,  0.42,  0.36, -0.10,  0.57,  0.22,  0.74,  1.15,
     +    0.40,  0.82,  0.96,  1.09,  0.42,  1.13,  0.24,  0.51,  0.60,
     +    0.06,  0.38,  0.15,  0.59,  0.76,  1.16,  0.02,  0.86,  1.14,
     +    0.37,  0.38,  0.26,  0.26,  0.07,  0.87,  0.90,  0.83,  0.09,
     +    0.03,  0.56, -0.19,  0.51,  1.07, -0.13,  0.99,  0.84,  0.22/
      DATA Y/
     +   -0.11,  1.07,  1.11, -0.17,  0.08,  0.09,  0.91,  0.17, -0.02,
     +    0.83,  1.08,  0.87,  0.46,  0.66,  0.50, -0.14,  0.78,  1.08,
```

```
     +    0.65,  0.00,  1.03,  0.06,  0.69, -0.16,  0.02,  0.59,  0.19,
     +    0.54,  0.68,  0.95,  0.30,  0.77,  0.94,  0.76,  0.56,  0.12,
     +    0.05, -0.07,  1.01,  0.61,  1.04, -0.07,  0.46,  1.07,  0.87,
     +    0.11,  0.63,  0.06,  0.53,  0.95,  0.78,  0.48,  0.45,  0.77,
     +    0.78,  0.29,  0.38,  0.85, -0.10,  1.17,  0.35,  1.14, -0.04,
     +    0.34, -0.18,  0.78,  0.17,  0.63,  0.88, -0.12,  0.58, -0.12,
     +    1.00,  0.99,  0.45,  0.86, -0.15,  0.97,  0.99,  0.90,  0.42,
     +    0.61,  0.74,  0.41,  0.44,  1.08,  1.06,  1.18,  0.89,  0.74,
     +    0.74, -0.06,  0.00,  0.99,  0.03,  1.00, -0.04,  0.24,  0.65,
     +    0.12,  0.13, -0.09, -0.05,  1.03,  1.07, -0.02,  1.18,  0.19,
     +    0.03, -0.03,  0.86,  1.12,  0.38,  0.72, -0.20, -0.08, -0.18,
     +    0.32,  0.13, -0.19,  0.93,  0.81,  0.31,  1.09, -0.03,  1.01,
     +   -0.17,  0.84, -0.11,  0.45,  0.18,  0.23,  0.81,  0.39,  1.09,
     +   -0.05,  0.58,  0.53,  0.96,  0.43,  0.48,  0.96, -0.03,  1.13,
     +    1.16,  0.16,  1.15,  0.57,  0.13,  0.71,  0.35,  1.04,  0.62,
     +    1.03,  0.98,  0.31,  0.70,  0.97,  0.87,  1.14,  0.08,  1.19,
     +    0.88,  1.00,  0.51,  0.03,  0.17,  1.01,  0.44,  0.17, -0.11/
C
      END
```

Example 5 - negative values allowed



Example 5 - negative values not allowed



Example 6 - single point mode (Fortran single precision)

```
      PROGRAM NNEX06
C
C  Code to illustrate the use of single point mode.
C
      PARAMETER(ISLIM = 171, NUMXOUT = 21, NUMYOUT = 21,
     +          IDIM=2*NUMXOUT*NUMYOUT)
C
      REAL X(ISLIM), Y(ISLIM), Z(ISLIM)
      REAL XI(NUMXOUT),YI(NUMYOUT),ZI(NUMXOUT,NUMYOUT)
C
      INTEGER I,J,IWORK(IDIM)
C
C  The data values for X and Y (defined in BLOCKDATA) are random numbers
C  in the range  -0.2 to 1.2.  They are explicitly set so that they
C  will be uniform across all compilers.
C
```

```
      COMMON /DVALUE/X,Y
C
C  Define the function values.
C
      DO 5 I=1,ISLIM
        Z(I) = (X(I)-0.25)**2 + (Y(I)-0.50)**2
    5 CONTINUE
C
C  Define the output grid.
C
      XMIN =  0.0
      XMAX =  1.0
      XINC = (XMAX-XMIN)/(NUMXOUT-1.)
      DO 20 I=1,NUMXOUT
        XI(I) = XMIN+REAL(I-1) * XINC
   20 CONTINUE
C
      YMIN =  0.0
      YMAX =  1.0
      YINC = (YMAX-YMIN)/(NUMYOUT-1.)
      DO 30 J=1,NUMYOUT
        YI(J) = YMIN+REAL(J-1) * YINC
   30 CONTINUE
C
C  Enter single point mode.
C
      CALL NNPNTINITS(ISLIM,X,Y,Z)
C
C  Calculate the interpolated values at the desired points.
C
      DO 40 I=1,NUMXOUT
        DO 50 J=1,NUMYOUT
          CALL NNPNTS(XI(I),YI(J),ZI(I,J))
   50   CONTINUE
   40 CONTINUE
C
C  Exit single point mode.
C
      CALL NNPNTEND()
C
C  Plot the interpolated surface.
C
      CALL DRWSRF(NUMXOUT,NUMYOUT,XI,YI,ZI,10.,-25.,50.,IWORK)
C
      STOP
      END
      BLOCKDATA DATAV
      COMMON /DVALUE/X,Y
      DIMENSION X(171),Y(171)
C
      DATA X/
     +    1.16,  0.47,  0.29,  0.72,  0.52,  1.12,  0.33,  0.20,  0.30,
     +    0.78,  0.92,  0.52,  0.44,  0.22, -0.10,  0.11,  0.59,  1.13,
     +    0.68,  1.11,  0.93,  0.29,  0.74,  0.43,  0.87,  0.87, -0.10,
     +    0.26,  0.85,  0.00, -0.02,  1.01, -0.12,  0.65,  0.39,  0.96,
     +    0.39,  0.38,  0.94, -0.03, -0.17,  0.00,  0.03,  0.67, -0.06,
     +    0.82, -0.03,  1.08,  0.37,  1.02, -0.11, -0.13,  1.03,  0.61,
     +    0.26,  0.18,  0.62,  0.42,  1.03,  0.72,  0.97,  0.08,  1.18,
     +    0.00,  0.69,  0.10,  0.80,  0.06,  0.82,  0.20,  0.46,  0.37,
     +    1.16,  0.93,  1.09,  0.96,  1.00,  0.80,  0.01,  0.12,  1.01,
     +    0.48,  0.79,  0.04,  0.42,  0.48, -0.18,  1.16,  0.85,  0.97,
```

```
     +    0.14,  0.40,  0.78,  1.12,  1.19,  0.68,  0.65,  0.41,  0.90,
     +    0.84, -0.11, -0.01, -0.02, -0.10,  1.04,  0.58,  0.61,  0.12,
     +   -0.02, -0.03,  0.27,  1.17,  1.02,  0.16, -0.17,  1.03,  0.13,
     +    0.04, -0.03,  0.15,  0.00, -0.01,  0.91,  1.20,  0.54, -0.14,
     +    1.03,  0.93,  0.42,  0.36, -0.10,  0.57,  0.22,  0.74,  1.15,
     +    0.40,  0.82,  0.96,  1.09,  0.42,  1.13,  0.24,  0.51,  0.60,
     +    0.06,  0.38,  0.15,  0.59,  0.76,  1.16,  0.02,  0.86,  1.14,
     +    0.37,  0.38,  0.26,  0.26,  0.07,  0.87,  0.90,  0.83,  0.09,
     +    0.03,  0.56, -0.19,  0.51,  1.07, -0.13,  0.99,  0.84,  0.22/
      DATA Y/
     +   -0.11,  1.07,  1.11, -0.17,  0.08,  0.09,  0.91,  0.17, -0.02,
     +    0.83,  1.08,  0.87,  0.46,  0.66,  0.50, -0.14,  0.78,  1.08,
     +    0.65,  0.00,  1.03,  0.06,  0.69, -0.16,  0.02,  0.59,  0.19,
     +    0.54,  0.68,  0.95,  0.30,  0.77,  0.94,  0.76,  0.56,  0.12,
     +    0.05, -0.07,  1.01,  0.61,  1.04, -0.07,  0.46,  1.07,  0.87,
     +    0.11,  0.63,  0.06,  0.53,  0.95,  0.78,  0.48,  0.45,  0.77,
     +    0.78,  0.29,  0.38,  0.85, -0.10,  1.17,  0.35,  1.14, -0.04,
     +    0.34, -0.18,  0.78,  0.17,  0.63,  0.88, -0.12,  0.58, -0.12,
     +    1.00,  0.99,  0.45,  0.86, -0.15,  0.97,  0.99,  0.90,  0.42,
     +    0.61,  0.74,  0.41,  0.44,  1.08,  1.06,  1.18,  0.89,  0.74,
     +    0.74, -0.06,  0.00,  0.99,  0.03,  1.00, -0.04,  0.24,  0.65,
     +    0.12,  0.13, -0.09, -0.05,  1.03,  1.07, -0.02,  1.18,  0.19,
     +    0.03, -0.03,  0.86,  1.12,  0.38,  0.72, -0.20, -0.08, -0.18,
     +    0.32,  0.13, -0.19,  0.93,  0.81,  0.31,  1.09, -0.03,  1.01,
     +   -0.17,  0.84, -0.11,  0.45,  0.18,  0.23,  0.81,  0.39,  1.09,
     +   -0.05,  0.58,  0.53,  0.96,  0.43,  0.48,  0.96, -0.03,  1.13,
     +    1.16,  0.16,  1.15,  0.57,  0.13,  0.71,  0.35,  1.04,  0.62,
     +    1.03,  0.98,  0.31,  0.70,  0.97,  0.87,  1.14,  0.08,  1.19,
     +    0.88,  1.00,  0.51,  0.03,  0.17,  1.01,  0.44,  0.17, -0.11/
C
      END
```

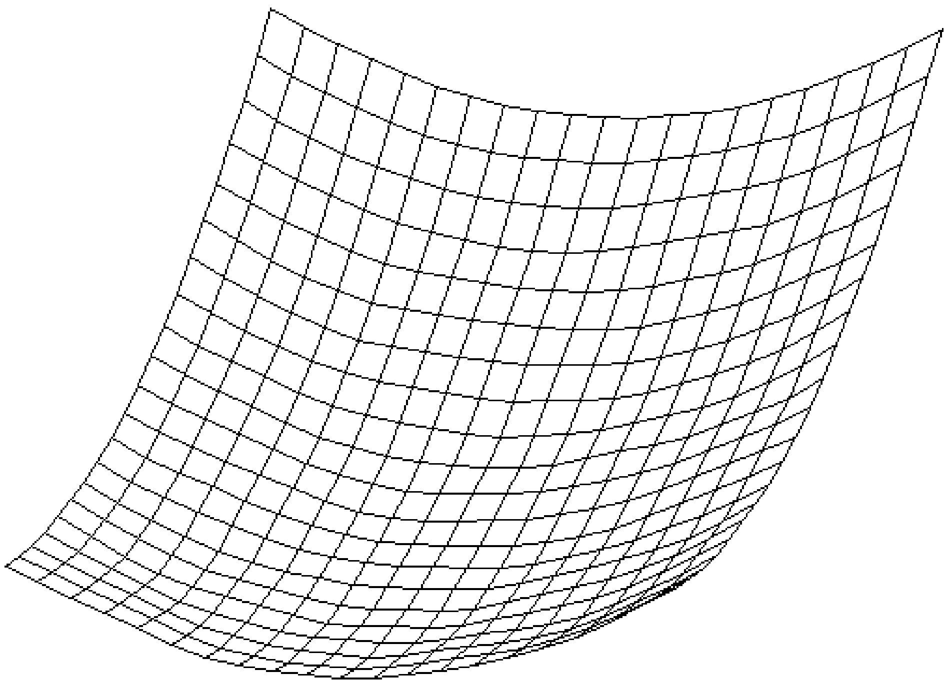
Example 6 - single point mode (C single precision)

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

extern void drwsrfc (int, int, float *, float *, float *,
              float, float, float, int *);
extern void c_nnpntinits(int, float [], float [], float []);
extern void c_nnpnts(float, float, float *);
extern void c_nnpntend();

#define NUMIN 171
#define NUMXOUT 21
#define NUMYOUT 21

main()
{
  int i, j, ier, iert, *iwk;
```

```
/*
 *  Coordinate data are defined as random numbers between
 *  -0.2 and 1.2. and are explicitly defined here for uniformity
 *  across platforms.
 */
float x[] = {
             1.16,  0.47,  0.29,  0.72,  0.52,  1.12,  0.33,  0.20,  0.30,
             0.78,  0.92,  0.52,  0.44,  0.22, -0.10,  0.11,  0.59,  1.13,
             0.68,  1.11,  0.93,  0.29,  0.74,  0.43,  0.87,  0.87, -0.10,
             0.26,  0.85,  0.00, -0.02,  1.01, -0.12,  0.65,  0.39,  0.96,
             0.39,  0.38,  0.94, -0.03, -0.17,  0.00,  0.03,  0.67, -0.06,
             0.82, -0.03,  1.08,  0.37,  1.02, -0.11, -0.13,  1.03,  0.61,
             0.26,  0.18,  0.62,  0.42,  1.03,  0.72,  0.97,  0.08,  1.18,
             0.00,  0.69,  0.10,  0.80,  0.06,  0.82,  0.20,  0.46,  0.37,
             1.16,  0.93,  1.09,  0.96,  1.00,  0.80,  0.01,  0.12,  1.01,
             0.48,  0.79,  0.04,  0.42,  0.48, -0.18,  1.16,  0.85,  0.97,
             0.14,  0.40,  0.78,  1.12,  1.19,  0.68,  0.65,  0.41,  0.90,
             0.84, -0.11, -0.01, -0.02, -0.10,  1.04,  0.58,  0.61,  0.12,
            -0.02, -0.03,  0.27,  1.17,  1.02,  0.16, -0.17,  1.03,  0.13,
             0.04, -0.03,  0.15,  0.00, -0.01,  0.91,  1.20,  0.54, -0.14,
             1.03,  0.93,  0.42,  0.36, -0.10,  0.57,  0.22,  0.74,  1.15,
             0.40,  0.82,  0.96,  1.09,  0.42,  1.13,  0.24,  0.51,  0.60,
             0.06,  0.38,  0.15,  0.59,  0.76,  1.16,  0.02,  0.86,  1.14,
             0.37,  0.38,  0.26,  0.26,  0.07,  0.87,  0.90,  0.83,  0.09,
             0.03,  0.56, -0.19,  0.51,  1.07, -0.13,  0.99,  0.84,  0.22
            };
float y[] = {
            -0.11,  1.07,  1.11, -0.17,  0.08,  0.09,  0.91,  0.17, -0.02,
             0.83,  1.08,  0.87,  0.46,  0.66,  0.50, -0.14,  0.78,  1.08,
             0.65,  0.00,  1.03,  0.06,  0.69, -0.16,  0.02,  0.59,  0.19,
             0.54,  0.68,  0.95,  0.30,  0.77,  0.94,  0.76,  0.56,  0.12,
             0.05, -0.07,  1.01,  0.61,  1.04, -0.07,  0.46,  1.07,  0.87,
             0.11,  0.63,  0.06,  0.53,  0.95,  0.78,  0.48,  0.45,  0.77,
             0.78,  0.29,  0.38,  0.85, -0.10,  1.17,  0.35,  1.14, -0.04,
             0.34, -0.18,  0.78,  0.17,  0.63,  0.88, -0.12,  0.58, -0.12,
             1.00,  0.99,  0.45,  0.86, -0.15,  0.97,  0.99,  0.90,  0.42,
             0.61,  0.74,  0.41,  0.44,  1.08,  1.06,  1.18,  0.89,  0.74,
             0.74, -0.06,  0.00,  0.99,  0.03,  1.00, -0.04,  0.24,  0.65,
             0.12,  0.13, -0.09, -0.05,  1.03,  1.07, -0.02,  1.18,  0.19,
             0.03, -0.03,  0.86,  1.12,  0.38,  0.72, -0.20, -0.08, -0.18,
             0.32,  0.13, -0.19,  0.93,  0.81,  0.31,  1.09, -0.03,  1.01,
            -0.17,  0.84, -0.11,  0.45,  0.18,  0.23,  0.81,  0.39,  1.09,
            -0.05,  0.58,  0.53,  0.96,  0.43,  0.48,  0.96, -0.03,  1.13,
             1.16,  0.16,  1.15,  0.57,  0.13,  0.71,  0.35,  1.04,  0.62,
             1.03,  0.98,  0.31,  0.70,  0.97,  0.87,  1.14,  0.08,  1.19,
             0.88,  1.00,  0.51,  0.03,  0.17,  1.01,  0.44,  0.17, -0.11
            };

  float z[NUMIN], rpnt=999.;
  float xo[NUMXOUT], yo[NUMYOUT], zo[NUMXOUT][NUMYOUT], xc, yc;

  iwk = (int *) calloc(2*NUMXOUT*NUMYOUT,sizeof(char));

  for (i = 0 ; i < NUMIN ; i++) {
    z[i] = (x[i]-0.25)*(x[i]-0.25) + (y[i]-0.50)*(y[i]-0.50);
  }

  xc = 1./(NUMXOUT-1.);
  for (i = 0 ; i < NUMXOUT ; i++) {
    xo[i] = i * xc;
  }
```

```
  yc = 1./(NUMYOUT-1.);
  for (j = 0 ; j < NUMYOUT ; j++) {
    yo[j] = j * yc;
  }

/*
 *  Enter single point mode.
 */
  c_nnpntinits(NUMIN, x, y, z);

/*
 *  Calculate the interpolated values at the desired points.
 */
  for (i = 0 ; i < NUMXOUT ; i++) {
    for (j = 0 ; j < NUMYOUT ; j++) {
      c_nnpnts(xo[i], yo[j], &rpnt);
      zo[i][j] = rpnt;
    }
  }

/*
 *  Exit single point mode.
 */
  c_nnpntend();


/*
 *  Draw the surface plot.
 */
  drwsrfc (NUMXOUT, NUMYOUT, xo, yo, &zo[0][0], 10.,-25.,50., iwk);

}
```

Plot from using single point mode



Example 7 - outlying point (Fortran single precision)

```
      PROGRAM NNEX07
C
C  Produce a plot with a single outlying point.
C
      PARAMETER(ISLIM = 171, NUMXOUT = 21, NUMYOUT = 21,
     +          IDIM=2*NUMXOUT*NUMYOUT)
C
      REAL X(ISLIM), Y(ISLIM), Z(ISLIM)
      REAL XI(NUMXOUT), YI(NUMYOUT), ZI(NUMXOUT,NUMYOUT)
C
      INTEGER I,J,IWORK(IDIM)
C
C  The data values for X and Y (defined in BLOCKDATA) are random numbers
C  in the range  -0.2 to 1.2.  They are explicitly set so that they
C  will be uniform across all compilers.
C
      COMMON /DVALUE/X,Y
C
C  Define the function values.
C
```

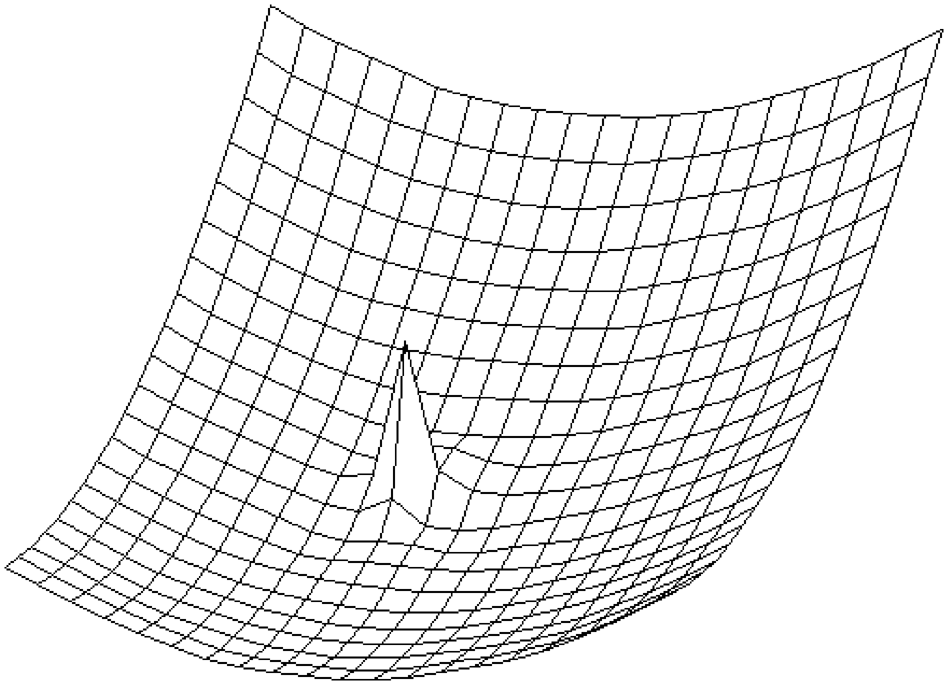
```
      DO 5 I=1,ISLIM
        Z(I) = (X(I)-0.25)**2 + (Y(I)-0.50)**2
    5 CONTINUE
C
C  Modify the first point to be an outlier.
C
      X(1) = 0.40
      Y(1) = 0.50
      Z(1) = 0.40
C
C  Define the output grid.
C
      XMIN = 0.0
      XMAX = 1.0
      XINC = (XMAX-XMIN)/(NUMXOUT-1.)
      DO 20 I=1,NUMXOUT
        XI(I) = XMIN+REAL(I-1) * XINC
   20 CONTINUE
C
      YMIN =  0.0
      YMAX =  1.0
      YINC = (YMAX-YMIN)/(NUMYOUT-1.)
      DO 30 J=1,NUMYOUT
        YI(J) = YMIN+REAL(J-1) * YINC
   30 CONTINUE
C
      CALL NNSETI('IGR - use gradient estimates',1)
      CALL NATGRIDS(ISLIM,X,Y,Z,NUMXOUT,NUMYOUT,XI,YI,ZI,IER)
C
C  Plot the interpolated surface.
C
      CALL DRWSRF(NUMXOUT,NUMYOUT,XI,YI,ZI,10.,-25.,50.,IWORK)
C
      STOP
      END
      BLOCKDATA DATAV
      COMMON /DVALUE/X,Y
      DIMENSION X(171),Y(171)
C
      DATA X/
     +    1.16,  0.47,  0.29,  0.72,  0.52,  1.12,  0.33,  0.20,  0.30,
     +    0.78,  0.92,  0.52,  0.44,  0.22, -0.10,  0.11,  0.59,  1.13,
     +    0.68,  1.11,  0.93,  0.29,  0.74,  0.43,  0.87,  0.87, -0.10,
     +    0.26,  0.85,  0.00, -0.02,  1.01, -0.12,  0.65,  0.39,  0.96,
     +    0.39,  0.38,  0.94, -0.03, -0.17,  0.00,  0.03,  0.67, -0.06,
     +    0.82, -0.03,  1.08,  0.37,  1.02, -0.11, -0.13,  1.03,  0.61,
     +    0.26,  0.18,  0.62,  0.42,  1.03,  0.72,  0.97,  0.08,  1.18,
     +    0.00,  0.69,  0.10,  0.80,  0.06,  0.82,  0.20,  0.46,  0.37,
     +    1.16,  0.93,  1.09,  0.96,  1.00,  0.80,  0.01,  0.12,  1.01,
     +    0.48,  0.79,  0.04,  0.42,  0.48, -0.18,  1.16,  0.85,  0.97,
     +    0.14,  0.40,  0.78,  1.12,  1.19,  0.68,  0.65,  0.41,  0.90,
     +    0.84, -0.11, -0.01, -0.02, -0.10,  1.04,  0.58,  0.61,  0.12,
     +   -0.02, -0.03,  0.27,  1.17,  1.02,  0.16, -0.17,  1.03,  0.13,
     +    0.04, -0.03,  0.15,  0.00, -0.01,  0.91,  1.20,  0.54, -0.14,
     +    1.03,  0.93,  0.42,  0.36, -0.10,  0.57,  0.22,  0.74,  1.15,
     +    0.40,  0.82,  0.96,  1.09,  0.42,  1.13,  0.24,  0.51,  0.60,
     +    0.06,  0.38,  0.15,  0.59,  0.76,  1.16,  0.02,  0.86,  1.14,
     +    0.37,  0.38,  0.26,  0.26,  0.07,  0.87,  0.90,  0.83,  0.09,
     +    0.03,  0.56, -0.19,  0.51,  1.07, -0.13,  0.99,  0.84,  0.22/
      DATA Y/
     +   -0.11,  1.07,  1.11, -0.17,  0.08,  0.09,  0.91,  0.17, -0.02,
```

```
     +    0.83,  1.08,  0.87,  0.46,  0.66,  0.50, -0.14,  0.78,  1.08,
     +    0.65,  0.00,  1.03,  0.06,  0.69, -0.16,  0.02,  0.59,  0.19,
     +    0.54,  0.68,  0.95,  0.30,  0.77,  0.94,  0.76,  0.56,  0.12,
     +    0.05, -0.07,  1.01,  0.61,  1.04, -0.07,  0.46,  1.07,  0.87,
     +    0.11,  0.63,  0.06,  0.53,  0.95,  0.78,  0.48,  0.45,  0.77,
     +    0.78,  0.29,  0.38,  0.85, -0.10,  1.17,  0.35,  1.14, -0.04,
     +    0.34, -0.18,  0.78,  0.17,  0.63,  0.88, -0.12,  0.58, -0.12,
     +    1.00,  0.99,  0.45,  0.86, -0.15,  0.97,  0.99,  0.90,  0.42,
     +    0.61,  0.74,  0.41,  0.44,  1.08,  1.06,  1.18,  0.89,  0.74,
     +    0.74, -0.06,  0.00,  0.99,  0.03,  1.00, -0.04,  0.24,  0.65,
     +    0.12,  0.13, -0.09, -0.05,  1.03,  1.07, -0.02,  1.18,  0.19,
     +    0.03, -0.03,  0.86,  1.12,  0.38,  0.72, -0.20, -0.08, -0.18,
     +    0.32,  0.13, -0.19,  0.93,  0.81,  0.31,  1.09, -0.03,  1.01,
     +   -0.17,  0.84, -0.11,  0.45,  0.18,  0.23,  0.81,  0.39,  1.09,
     +   -0.05,  0.58,  0.53,  0.96,  0.43,  0.48,  0.96, -0.03,  1.13,
     +    1.16,  0.16,  1.15,  0.57,  0.13,  0.71,  0.35,  1.04,  0.62,
     +    1.03,  0.98,  0.31,  0.70,  0.97,  0.87,  1.14,  0.08,  1.19,
     +    0.88,  1.00,  0.51,  0.03,  0.17,  1.01,  0.44,  0.17, -0.11/
C
      END
```

Outlying point, natural neighbor vs. curve fitting

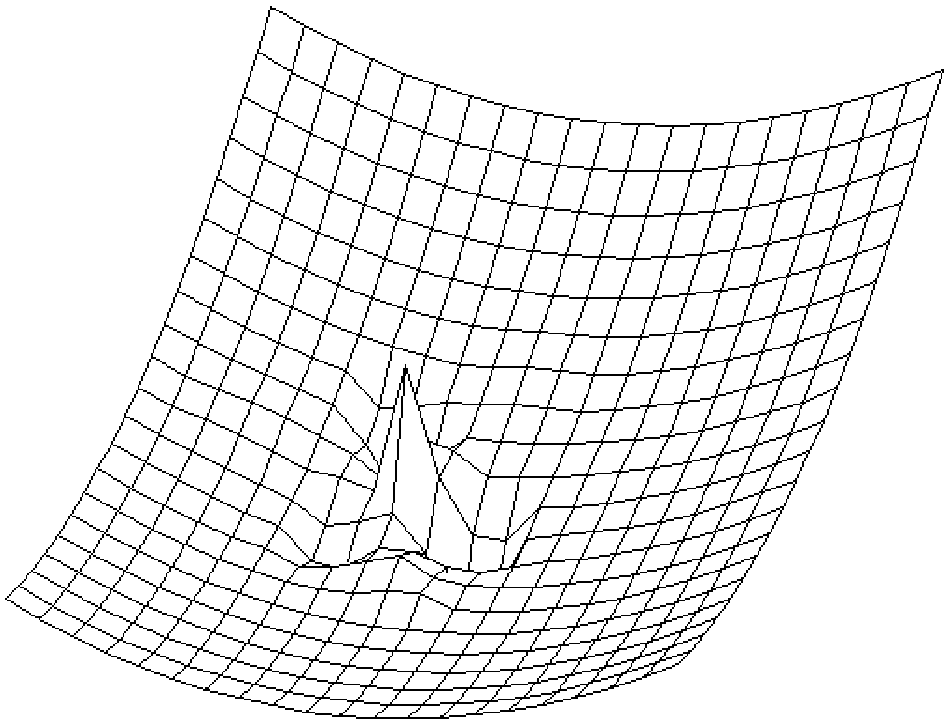
Outlying point, natural neighbor algorithm

The following plot was produced using the natural neighbor algorithm:



Outlying point, curve fitting algorithm

The following plot was produced using the curve fitting algorithm described in Akima 1 and Akima 2. The same input data were used as for the plot above. If you have access to NCAR Graphics, Akima's algorithm is available via the IDSFFT entry point (do a "man IDSFFT" for details).



Example 8 - Create file for algorithmic display tool

```
      PROGRAM NNEX08
C
      PARAMETER(ISLIM = 11, NUMXOUT = 26, NUMYOUT = 26)
C
      REAL X(ISLIM), Y(ISLIM), Z(ISLIM)
      REAL XI(NUMXOUT), YI(NUMYOUT), ZI(NUMXOUT,NUMYOUT)
C
      DATA X/0.00,0.15,0.10,0.35,0.50,0.55,0.55,0.65,0.85,0.80,1.00/
      DATA Y/0.50,0.80,0.30,0.60,0.35,1.00,0.00,0.55,0.80,0.25,0.45/
C
      DO 130 I=1,ISLIM
          Z(I) = (X(I)-0.25)**2 + (Y(I)-0.50)**2
  130 CONTINUE
C
      XINC = 1./(NUMXOUT-1.)
      DO 20 I=1,NUMXOUT
        XI(I) = REAL(I-1) * XINC
   20 CONTINUE
C
      YINC = 1./(NUMYOUT-1.)
```

```
      DO 30 J=1,NUMYOUT
        YI(J) = REAL(J-1) * YINC
   30 CONTINUE
C
      CALL NNSETI('ADF',1)
      CALL NNSETC('ALG','example8.dat')
      CALL NNSETI('IGR',0)
      CALL NATGRIDS(ISLIM,X,Y,Z,NUMXOUT,NUMYOUT,XI,YI,ZI,IER)
C
      STOP
      END
```

Output file for displaying algorithmic features

```
/*
/* Integer flags (I5 format).
/*
/*..+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
    8 - GKS workstation type (1=ncgm; 8=X11 window; 20=PostScript).
    1 - flags whether axes should be drawn.
    0 - Halfax/Grid flag (0=halfax and 1=grid)
    1 - Flags whether triangulation should be drawn.
    0 - Flags whether a blue dot should be drawn at (0.,0.) [0=no; 1=yes]
    0 - Flag to indicate whether the pseudo data should be included in the plot.
    1 - Flag indicating whether the natural neighbor circles are drawn.
    1 - Flags whether the centers of the natural neighborhood circles are drawn.
    1 - Flag indicating if Voronoi polygons should be drawn [0=no; 1=yes].
    1 - Flag indicating if the original points are to be marked.
/*
/*  Color information (3F7.3 format) as RGB triples
/*
/*..+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
  0.000  0.000  0.000 - background color
  1.000  1.000  1.000 - foreground color (used for axes)
  1.000  0.000  0.000 - circumcircle color
  0.000  1.000  0.000 - color of circumcircle centers
  0.000  1.000  1.000 - color for triangulation
  1.000  1.000  0.000 - Voronoi polygon color
  1.000  1.000  0.000 - color of vertex dots
  0.000  0.000  1.000 - color of reference dot
  0.000  0.000  1.000 - color for natural neighbor points
  1.000  1.000  1.000 - color to mark points where natural neighbors are desired
/*
/*  Scale factors (F7.3 format)
/*
/*..+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
  1.000 - scale factor for dots at vertices
  1.000 - scale factor for circumcircle centers
  2.000 - scale factor for circle lines
  2.000 - scale factor for Voronoi polygon lines
  2.000 - scale factor for tringulation lines
  1.000 - scale factor for axes lines
  1.000 - scale factor for points where natural neighbors are desired
  1.000 - scale factor for points marking natural neighbors
```

```
/*
/*  User coordinates for SET call (4E15.3 format), defaults if all zeros
/*
/*..+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
      0.000E+00      0.000E+00      0.000E+00      0.000E+00
/*
/*  Number of user input data. (I5 format)
/*
/*..+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
   11
/*
/*  User data.  The datum number occurs first (in I5 format) followed
/*  by the x,y,z values (in E15.3 format).
/*
/*..+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
    1     -8.998E-07      5.000E-01      6.250E-02
    2      1.500E-01      8.000E-01      1.000E-01
    3      1.000E-01      3.000E-01      6.250E-02
    4      3.500E-01      6.000E-01      2.000E-02
    5      5.000E-01      3.500E-01      8.500E-02
    6      5.500E-01      1.000E+00      3.400E-01
    7      5.500E-01      4.255E-06      3.400E-01
    8      6.500E-01      5.500E-01      1.625E-01
    9      8.500E-01      8.000E-01      4.500E-01
   10      8.000E-01      2.500E-01      3.650E-01
   11      1.000E+00      4.500E-01      5.650E-01
/*
/*  Pseudo data.
/*
/*..+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
   12     -1.021E+01     -1.021E+01     -5.310E+00
   13      5.099E+01     -1.021E+01      2.481E+01
   14     -1.021E+01      5.099E+01     -3.763E+00
/*
/*  The number of circumcircles (I5 format).
/*
/*..+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
   23
/*
/*  Circumcircle data.  The first three numbers are the numbers of the
/*  data (as listed above) lying on the circumcircle; the next two
/*  numbers give the center position of the circumcircle; the final
/*  number is the square of the radius of the circumcircle.
/*
/*..+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
   12    1    2     -2.296E+01      1.217E+01      6.634E+02
   12    2   14     -3.170E+01      2.039E+01      1.398E+03
   12    3    1     -6.947E+00     -3.099E+00      6.122E+01
    1    3    4      1.969E-01      4.734E-01      3.947E-02
    1    4    2      1.583E-01      6.083E-01      3.681E-02
    3    5    4      2.913E-01      3.948E-01      4.557E-02
    2    4    6      3.833E-01      8.333E-01      5.555E-02
   14    2    6     -1.148E+01      2.456E+01      7.000E+02
    3    7    5      3.224E-01      1.461E-01      7.315E-02
   12    7    3     -3.860E+00     -6.127E+00      5.699E+01
   12   13    7      2.039E+01     -3.168E+01      1.397E+03
    4    8    6      5.308E-01      7.596E-01      5.815E-02
    4    5    8      4.898E-01      5.139E-01      2.696E-02
   14    6    9      2.116E+01      3.159E+01      1.360E+03
   13   14    9      3.454E+01      3.454E+01      2.273E+03
    6    8    9      6.196E-01      7.793E-01      5.353E-02
```

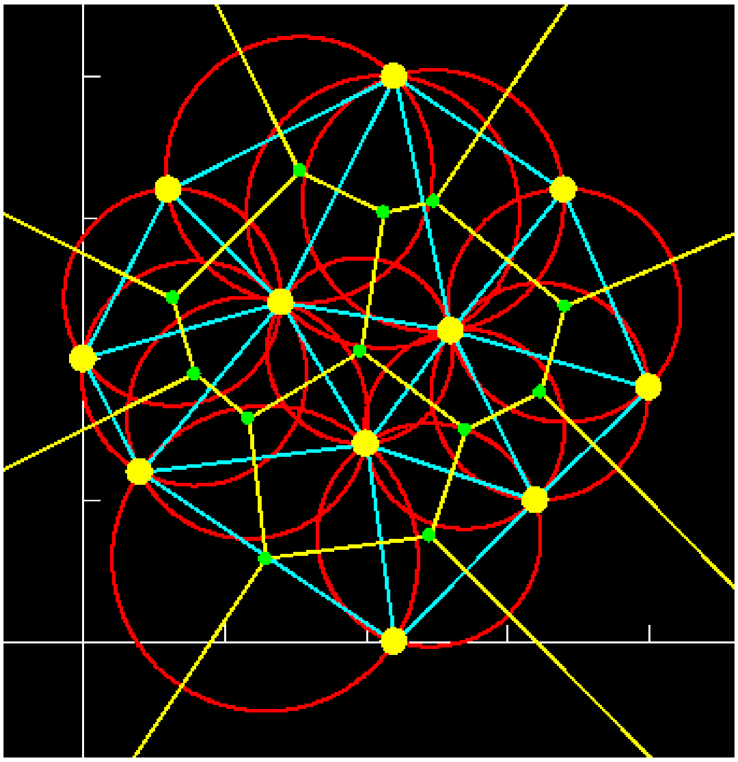
```
    5   10    8      6.750E-01      3.750E-01      3.125E-02
    5    7   10      6.125E-01      1.875E-01      3.906E-02
   13   10    7      2.243E+01     -2.163E+01      9.462E+02
    8   10   11      8.083E-01      4.417E-01      3.681E-02
    8   11    9      8.517E-01      5.936E-01      4.260E-02
   13   11   10      2.250E+01     -2.125E+01      9.335E+02
   13    9   11      2.981E+01      1.300E+01      9.875E+02
/*
/*  Number of points where natural neighbors are to be marked and
/*  a flag indicating whether just the points where first order neighbors
/*  are desired are marked (-1), whether the first  order neighbors
/*  will be marked as well (0), or both first and second order neighbors
/*  are marked (1).  The points will be marked with Xs, in the
/*  color described above. (2I5 format)
/*
/*..+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
    0    0
/*
/*  The coordinate list of points whose natural neighbors are to
/*  be displayed (using the color index as described above), should
/*  be listed here in 2E15.3 format.
/*
/*..+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
/*    0.000E-00      0.000E-00
```

Example 8 - plot algorithm defaults

The plot below was produced using the file example8.dat produced from the Example 8 source code by executing "nnalg example8.dat":

Picture elements:

- the yellow dots mark the original data points
- the red circles are the natural neighbor circumcircles
- the cyan lines form the Delaunay triangulation
- the green dots mark the circumcircle centers
- the yellow lines show the Voronoi polygons



Example 9 - plot first and second order neighbors

```
      PROGRAM NNEX09
C
C   Code to illustrate plotting neighbors with the display tool.
C
      PARAMETER(ISLIM = 131, NUMXOUT = 21, NUMYOUT = 21,
     +          IDIM=2*NUMXOUT*NUMYOUT)
C
      REAL X(ISLIM), Y(ISLIM), Z(ISLIM)
      REAL XI(NUMXOUT),YI(NUMYOUT),ZI(NUMXOUT,NUMYOUT)
C
      INTEGER I,J
C
C   The data values for Z (defined in BLOCKDATA) are pressure fields.
C
      COMMON /DVALUE/X,Y,Z
C
C   Normalize Z to be between 0. and 1.
C
      ZMAX = Z(1)
      ZMIN = Z(1)
      DO 4 I=2,ISLIM
        ZMAX = MAX(ZMAX,Z(I))
```

```
        ZMIN = MIN(ZMIN,Z(I))
    4 CONTINUE
C
      RC = 1./(ZMAX-ZMIN)
      DO 5 I=1,ISLIM
        Z(I) = RC*(Z(I)-ZMIN)
    5 CONTINUE
C
C  Define the output grid.
C
      XMIN =  0.1
      XMAX =  0.9
      XINC = (XMAX-XMIN)/(NUMXOUT-1.)
      DO 20 I=1,NUMXOUT
        XI(I) = XMIN+REAL(I-1) * XINC
   20 CONTINUE
C
      YMIN =  0.1
      YMAX =  0.5
      YINC = (YMAX-YMIN)/(NUMYOUT-1.)
      DO 30 J=1,NUMYOUT
        YI(J) = YMIN+REAL(J-1) * YINC
   30 CONTINUE
C
      CALL NNSETI('ADF',1)
      CALL NNSETC('ALG','example9.dat')
      CALL NATGRIDS(ISLIM,X,Y,Z,NUMXOUT,NUMYOUT,XI,YI,ZI,IER)
C
      STOP
      END
      BLOCKDATA DATAV
      COMMON /DVALUE/X,Y,Z
      DIMENSION X(131),Y(131),Z(131)
C
      DATA X/
     +   0.120, 0.127, 0.135, 0.140, 0.140, 0.142, 0.146, 0.153, 0.153,
     +   0.156, 0.169, 0.191, 0.200, 0.202, 0.208, 0.228, 0.228, 0.230,
     +   0.247, 0.251, 0.266, 0.277, 0.279, 0.279, 0.300, 0.326, 0.327,
     +   0.333, 0.342, 0.349, 0.350, 0.366, 0.379, 0.385, 0.386, 0.387,
     +   0.389, 0.393, 0.416, 0.431, 0.436, 0.440, 0.441, 0.449, 0.453,
     +   0.453, 0.458, 0.467, 0.476, 0.477, 0.486, 0.502, 0.503, 0.505,
     +   0.511, 0.511, 0.517, 0.524, 0.526, 0.527, 0.531, 0.550, 0.555,
     +   0.559, 0.563, 0.563, 0.567, 0.577, 0.585, 0.587, 0.602, 0.605,
     +   0.606, 0.607, 0.619, 0.626, 0.637, 0.638, 0.642, 0.642, 0.653,
     +   0.658, 0.660, 0.667, 0.670, 0.677, 0.686, 0.690, 0.694, 0.695,
     +   0.704, 0.705, 0.706, 0.718, 0.724, 0.727, 0.730, 0.738, 0.744,
     +   0.746, 0.763, 0.766, 0.770, 0.790, 0.790, 0.792, 0.793, 0.800,
     +   0.815, 0.817, 0.821, 0.822, 0.823, 0.823, 0.830, 0.833, 0.840,
     +   0.841, 0.846, 0.849, 0.849, 0.850, 0.853, 0.856, 0.865, 0.871,
     +   0.871, 0.878, 0.884, 0.097, 0.100
     +  /
      DATA Y/
     +   0.398, 0.445, 0.317, 0.288, 0.522, 0.254, 0.557, 0.505, 0.374,
     +   0.222, 0.538, 0.394, 0.209, 0.490, 0.285, 0.349, 0.440, 0.523,
     +   0.310, 0.216, 0.185, 0.416, 0.371, 0.249, 0.488, 0.382, 0.329,
     +   0.491, 0.465, 0.238, 0.168, 0.404, 0.475, 0.338, 0.199, 0.366,
     +   0.307, 0.285, 0.426, 0.168, 0.329, 0.242, 0.510, 0.481, 0.367,
     +   0.403, 0.433, 0.297, 0.137, 0.181, 0.435, 0.486, 0.337, 0.118,
     +   0.297, 0.248, 0.397, 0.165, 0.195, 0.078, 0.033, 0.336, 0.453,
     +   0.384, 0.494, 0.132, 0.253, 0.297, 0.192, 0.438, 0.143, 0.248,
     +   0.464, 0.393, 0.338, 0.362, 0.431, 0.308, 0.262, 0.412, 0.201,
```

```
     +   0.476, 0.151, 0.334, 0.441, 0.375, 0.405, 0.298, 0.177, 0.347,
     +   0.437, 0.243, 0.370, 0.219, 0.282, 0.396, 0.430, 0.306, 0.366,
     +   0.259, 0.420, 0.478, 0.189, 0.268, 0.442, 0.360, 0.314, 0.495,
     +   0.246, 0.209, 0.467, 0.339, 0.148, 0.410, 0.381, 0.505, 0.443,
     +   0.167, 0.123, 0.469, 0.413, 0.316, 0.379, 0.499, 0.546, 0.519,
     +   0.608, 0.353, 0.118, 0.423, 0.349
     +  /
      DATA Z/
     +   137.0, 161.0, 161.0, 157.0, 161.0, 174.0, 173.0, 166.0, 124.0,
     +   179.0, 166.0, 133.0, 130.0, 150.0, 139.0, 134.0, 127.0, 155.0,
     +   175.0, 143.0, 155.0, 160.0, 150.0, 185.0, 240.0, 200.0, 218.0,
     +   245.0, 242.0, 169.0, 151.0, 197.0, 253.0, 189.0, 150.0, 194.0,
     +   188.0, 184.0, 205.0, 152.0, 168.0, 137.0, 291.0, 256.0, 150.0,
     +   160.0, 174.0, 143.0, 200.0, 165.0, 173.0, 212.0, 109.0, 181.1,
     +   143.0, 160.0, 113.0, 181.0, 173.0, 179.0, 173.0, 145.0, 169.0,
     +   143.0, 197.0, 188.0, 189.0, 181.0, 193.0, 166.0, 196.0, 192.0,
     +   189.0, 168.0, 184.0, 179.0, 178.0, 197.0, 199.0, 181.0, 198.0,
     +   174.0, 193.0, 200.0, 178.0, 200.0, 197.0, 214.0, 197.0, 202.0,
     +   192.0, 209.0, 210.0, 197.0, 205.0, 210.0, 200.0, 205.0, 208.0,
     +   200.0, 206.0, 183.0, 191.0, 195.0, 212.0, 207.0, 205.0, 174.0,
     +   204.0, 203.0, 184.0, 197.0, 195.0, 186.0, 189.0, 178.0, 180.1,
     +   200.0, 183.0, 179.9, 184.0, 193.0, 193.0, 176.0, 156.0, 163.0,
     +   139.0, 191.0, 179.0, 157.0, 159.0
     +  /
C
      END
```

Modified data output file

```
/*
/* Integer flags (I5 format).
/*
/*..+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
    8 - GKS workstation type (1=ncgm; 8=X11 window; 20=PostScript).
    0 - flags whether axes should be drawn.
    0 - Halfax/Grid flag (0=halfax and 1=grid)
    1 - Flags whether triangulation should be drawn.
    0 - Flags whether a blue dot should be drawn at (0.,0.) [0=no; 1=yes]
    0 - Flag to indicate whether the pseudo data should be included in the plot.
    0 - Flag indicating whether the natural neighbor circles are drawn.
    0 - Flags whether the centers of the natural neighborhood circles are drawn.
    0 - Flag indicating if Voronoi polygons should be drawn [0=no; 1=yes].
    1 - Flag indicating if the original points are to be marked.
/*
/*  Color information (3F7.3 format) as RGB triples
/*
/*..+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
  1.000  1.000  1.000 - background color
  0.000  0.000  0.000 - foreground color (used for axes)
  1.000  0.000  0.000 - circumcircle color
  0.000  1.000  0.000 - color of circumcircle centers
  0.000  0.000  0.000 - color for triangulation
  1.000  1.000  0.000 - Voronoi polygon color
  1.000  0.650  0.000 - color of vertex dots
```

```
  0.000  0.000  1.000 - color of reference dot
  0.000  0.000  1.000 - color for natural neighbor points
  0.000  0.000  1.000 - color to mark points where natural neighbors are desired
/*
/*  Scale factors (F7.3 format)
/*
/*..+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
  0.500 - scale factor for dots at vertices
  1.000 - scale factor for circumcircle centers
  2.000 - scale factor for circle lines
  2.000 - scale factor for Voronoi polygon lines
  1.000 - scale factor for tringulation lines
  1.000 - scale factor for axes lines
  1.300 - scale factor for points where natural neighbors are desired
  0.500 - scale factor for points marking natural neighbors
/*
/*  User coordinates for SET call (4E15.3 format), defaults if all zeros
/*
/*..+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
      0.050E+00      0.950E+00     -0.150E+00      0.750E+00
/*
/*  Number of user input data. (I5 format)
/*
/*..+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
  131
/*
/*  User data.  The datum number occurs first (in I5 format) followed
/*  by the x,y,z values (in E15.3 format).
/*
/*..+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
    1      1.200E-01      3.980E-01      1.538E-01
    2      1.270E-01      4.450E-01      2.857E-01
    3      1.350E-01      3.170E-01      2.857E-01
    4      1.400E-01      2.880E-01      2.637E-01
    5      1.400E-01      5.220E-01      2.857E-01
    6      1.420E-01      2.540E-01      3.571E-01
    7      1.460E-01      5.570E-01      3.516E-01
    8      1.530E-01      5.050E-01      3.132E-01
    9      1.530E-01      3.740E-01      8.242E-02
   10      1.560E-01      2.220E-01      3.846E-01
   11      1.690E-01      5.380E-01      3.132E-01
   12      1.910E-01      3.940E-01      1.319E-01
   13      2.000E-01      2.090E-01      1.154E-01
   14      2.020E-01      4.900E-01      2.253E-01
   15      2.080E-01      2.850E-01      1.648E-01
   16      2.280E-01      3.490E-01      1.374E-01
   17      2.280E-01      4.400E-01      9.890E-02
   18      2.300E-01      5.230E-01      2.527E-01
   19      2.470E-01      3.100E-01      3.626E-01
   20      2.510E-01      2.160E-01      1.868E-01
   21      2.660E-01      1.850E-01      2.527E-01
   22      2.770E-01      4.160E-01      2.802E-01
   23      2.790E-01      3.710E-01      2.253E-01
   24      2.790E-01      2.490E-01      4.176E-01
   25      3.000E-01      4.880E-01      7.198E-01
   26      3.260E-01      3.820E-01      5.000E-01
   27      3.270E-01      3.290E-01      5.989E-01
   28      3.330E-01      4.910E-01      7.473E-01
   29      3.420E-01      4.650E-01      7.308E-01
   30      3.490E-01      2.380E-01      3.297E-01
   31      3.500E-01      1.680E-01      2.308E-01
```

32	3.660E-01	4.040E-01	4.835E-01
33	3.790E-01	4.750E-01	7.912E-01
34	3.850E-01	3.380E-01	4.396E-01
35	3.860E-01	1.990E-01	2.253E-01
36	3.870E-01	3.660E-01	4.670E-01
37	3.890E-01	3.070E-01	4.341E-01
38	3.930E-01	2.850E-01	4.121E-01
39	4.160E-01	4.260E-01	5.275E-01
40	4.310E-01	1.680E-01	2.363E-01
41	4.360E-01	3.290E-01	3.242E-01
42	4.400E-01	2.420E-01	1.538E-01
43	4.410E-01	5.100E-01	1.000E+00
44	4.490E-01	4.810E-01	8.077E-01
45	4.530E-01	3.670E-01	2.253E-01
46	4.530E-01	4.030E-01	2.802E-01
47	4.580E-01	4.330E-01	3.571E-01
48	4.670E-01	2.970E-01	1.868E-01
49	4.760E-01	1.370E-01	5.000E-01
50	4.770E-01	1.810E-01	3.077E-01
51	4.860E-01	4.350E-01	3.516E-01
52	5.020E-01	4.860E-01	5.659E-01
53	5.030E-01	3.370E-01	0.000E+00
54	5.050E-01	1.180E-01	3.962E-01
55	5.110E-01	2.970E-01	1.868E-01
56	5.110E-01	2.480E-01	2.802E-01
57	5.170E-01	3.970E-01	2.198E-02
58	5.240E-01	1.650E-01	3.956E-01
59	5.260E-01	1.950E-01	3.516E-01
60	5.270E-01	7.800E-02	3.846E-01
61	5.310E-01	3.300E-02	3.516E-01
62	5.500E-01	3.360E-01	1.978E-01
63	5.550E-01	4.530E-01	3.297E-01
64	5.590E-01	3.840E-01	1.868E-01
65	5.630E-01	4.940E-01	4.835E-01
66	5.630E-01	1.320E-01	4.341E-01
67	5.670E-01	2.530E-01	4.396E-01
68	5.770E-01	2.970E-01	3.956E-01
69	5.850E-01	1.920E-01	4.615E-01
70	5.870E-01	4.380E-01	3.132E-01
71	6.020E-01	1.430E-01	4.780E-01
72	6.050E-01	2.480E-01	4.560E-01
73	6.060E-01	4.640E-01	4.396E-01
74	6.070E-01	3.930E-01	3.242E-01
75	6.190E-01	3.380E-01	4.121E-01
76	6.260E-01	3.620E-01	3.846E-01
77	6.370E-01	4.310E-01	3.791E-01
78	6.380E-01	3.080E-01	4.835E-01
79	6.420E-01	2.620E-01	4.945E-01
80	6.420E-01	4.120E-01	3.956E-01
81	6.530E-01	2.010E-01	4.890E-01
82	6.580E-01	4.760E-01	3.571E-01
83	6.600E-01	1.510E-01	4.615E-01
84	6.670E-01	3.340E-01	5.000E-01
85	6.700E-01	4.410E-01	3.791E-01
86	6.770E-01	3.750E-01	5.000E-01
87	6.860E-01	4.050E-01	4.835E-01
88	6.900E-01	2.980E-01	5.769E-01
89	6.940E-01	1.770E-01	4.835E-01
90	6.950E-01	3.470E-01	5.110E-01
91	7.040E-01	4.370E-01	4.560E-01
92	7.050E-01	2.430E-01	5.495E-01

```
   93      7.060E-01      3.700E-01      5.549E-01
   94      7.180E-01      2.190E-01      4.835E-01
   95      7.240E-01      2.820E-01      5.275E-01
   96      7.270E-01      3.960E-01      5.549E-01
   97      7.300E-01      4.300E-01      5.000E-01
   98      7.380E-01      3.060E-01      5.275E-01
   99      7.440E-01      3.660E-01      5.440E-01
  100      7.460E-01      2.590E-01      5.000E-01
  101      7.630E-01      4.200E-01      5.330E-01
  102      7.660E-01      4.780E-01      4.066E-01
  103      7.700E-01      1.890E-01      4.505E-01
  104      7.900E-01      2.680E-01      4.725E-01
  105      7.900E-01      4.420E-01      5.659E-01
  106      7.920E-01      3.600E-01      5.385E-01
  107      7.930E-01      3.140E-01      5.275E-01
  108      8.000E-01      4.950E-01      3.571E-01
  109      8.150E-01      2.460E-01      5.220E-01
  110      8.170E-01      2.090E-01      5.165E-01
  111      8.210E-01      4.670E-01      4.121E-01
  112      8.220E-01      3.390E-01      4.835E-01
  113      8.230E-01      1.480E-01      4.725E-01
  114      8.230E-01      4.100E-01      4.231E-01
  115      8.300E-01      3.810E-01      4.396E-01
  116      8.330E-01      5.050E-01      3.791E-01
  117      8.400E-01      4.430E-01      3.907E-01
  118      8.410E-01      1.670E-01      5.000E-01
  119      8.460E-01      1.230E-01      4.066E-01
  120      8.490E-01      4.690E-01      3.896E-01
  121      8.490E-01      4.130E-01      4.121E-01
  122      8.500E-01      3.160E-01      4.615E-01
  123      8.530E-01      3.790E-01      4.615E-01
  124      8.560E-01      4.990E-01      3.681E-01
  125      8.650E-01      5.460E-01      2.582E-01
  126      8.710E-01      5.190E-01      2.967E-01
  127      8.710E-01      6.080E-01      1.648E-01
  128      8.780E-01      3.530E-01      4.505E-01
  129      8.840E-01      1.180E-01      3.846E-01
  130      9.700E-02      4.230E-01      2.637E-01
  131      1.000E-01      3.490E-01      2.747E-01
/*
/*  Pseudo data.
/*
/*..+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
  132     -7.964E+00     -6.191E+00     -2.185E+00
  133      4.040E+01     -6.191E+00      9.203E+00
  134     -7.964E+00      3.090E+01      1.094E+00
/*
/*  The number of circumcircles (I5 format).
/*
/*..+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
  263
/*
/*  Circumcircle data.  The first three numbers are the numbers of the
/*  data (as listed above) lying on the circumcircle; the next two
/*  numbers give the center position of the circumcircle; the final
/*  number is the square of the radius of the circumcircle.
/*
/*..+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
  132    7  134     -1.653E+01      1.235E+01      4.173E+02
    2    8    5      1.120E-01      4.871E-01      1.999E-03
  132   10    6     -4.754E+00     -1.908E+00      2.864E+01
```

5	8	11	1.585E-01	5.227E-01	3.438E-04
5	11	7	1.499E-01	5.383E-01	3.645E-04
1	9	12	1.565E-01	4.135E-01	1.571E-03
1	12	2	1.567E-01	4.166E-01	1.688E-03
2	14	8	1.670E-01	4.633E-01	1.936E-03
8	14	11	1.818E-01	5.114E-01	8.688E-04
3	4	15	1.750E-01	3.090E-01	1.664E-03
10	13	15	1.878E-01	2.487E-01	1.725E-03
6	10	15	1.828E-01	2.528E-01	1.669E-03
4	6	15	1.734E-01	2.729E-01	1.343E-03
9	16	12	1.879E-01	3.538E-01	1.629E-03
3	15	16	1.833E-01	3.279E-01	2.448E-03
3	16	9	1.812E-01	3.337E-01	2.419E-03
2	17	14	1.776E-01	4.456E-01	2.566E-03
2	12	17	1.775E-01	4.427E-01	2.556E-03
7	11	18	2.233E-01	6.271E-01	1.089E-02
11	14	18	1.975E-01	5.222E-01	1.059E-03
14	17	18	2.459E-01	4.811E-01	2.011E-03
15	19	16	2.143E-01	3.182E-01	1.139E-03
13	20	15	2.210E-01	2.452E-01	1.753E-03
10	21	13	-8.957E-02	-6.900E-01	8.921E-01
13	21	20	2.291E-01	1.863E-01	1.363E-03
12	22	17	2.365E-01	3.953E-01	2.070E-03
12	16	22	2.368E-01	3.940E-01	2.100E-03
16	23	22	2.398E-01	3.918E-01	1.970E-03
16	19	23	2.616E-01	3.412E-01	1.189E-03
15	20	24	2.377E-01	2.556E-01	1.747E-03
15	24	19	2.451E-01	2.701E-01	1.596E-03
20	21	24	2.867E-01	2.141E-01	1.275E-03
17	22	25	2.675E-01	4.587E-01	1.913E-03
17	25	18	2.527E-01	4.809E-01	2.286E-03
22	23	26	2.983E-01	3.944E-01	9.205E-04
19	24	27	2.928E-01	2.951E-01	2.318E-03
23	27	26	3.075E-01	3.551E-01	1.064E-03
19	27	23	2.847E-01	3.291E-01	1.788E-03
18	25	28	3.073E-01	5.902E-01	1.049E-02
22	26	29	3.203E-01	4.261E-01	1.980E-03
25	29	28	3.182E-01	4.713E-01	6.081E-04
22	29	25	3.047E-01	4.468E-01	1.719E-03
24	30	27	3.196E-01	2.790E-01	2.550E-03
21	30	24	3.087E-01	2.097E-01	2.429E-03
21	31	30	3.133E-01	2.025E-01	2.539E-03
26	32	29	3.287E-01	4.245E-01	1.816E-03
29	32	33	3.685E-01	4.402E-01	1.319E-03
28	29	33	3.566E-01	4.846E-01	5.959E-04
30	31	35	3.512E-01	2.030E-01	1.228E-03
26	36	32	3.565E-01	3.739E-01	9.942E-04
27	34	36	3.528E-01	3.544E-01	1.307E-03
26	27	36	3.518E-01	3.560E-01	1.341E-03
27	37	34	3.583E-01	3.188E-01	1.083E-03
27	38	37	3.474E-01	2.881E-01	2.090E-03
27	30	38	3.455E-01	2.853E-01	2.252E-03
30	35	38	3.920E-01	2.418E-01	1.867E-03
32	36	39	3.988E-01	3.973E-01	1.119E-03
32	39	33	3.809E-01	4.380E-01	1.375E-03
31	40	35	3.905E-01	1.574E-01	1.753E-03
34	37	41	4.091E-01	3.253E-01	7.391E-04
34	41	36	4.134E-01	3.500E-01	9.528E-04
37	38	41	4.203E-01	3.013E-01	1.012E-03
35	40	42	4.243E-01	2.064E-01	1.518E-03
35	42	38	3.963E-01	2.414E-01	1.908E-03

18	28	43	3.481E-01	7.215E-01	5.335E-02
7	18	43	3.637E-01	9.739E-01	2.212E-01
28	33	43	3.786E-01	5.481E-01	5.338E-03
33	44	43	4.133E-01	4.867E-01	1.311E-03
33	39	44	4.152E-01	4.639E-01	1.435E-03
36	41	45	4.201E-01	3.589E-01	1.147E-03
36	45	46	4.197E-01	3.850E-01	1.432E-03
36	46	39	4.180E-01	3.880E-01	1.448E-03
39	46	47	4.384E-01	4.208E-01	5.303E-04
39	47	44	4.330E-01	4.532E-01	1.029E-03
38	42	48	4.318E-01	2.802E-01	1.525E-03
38	48	41	4.298E-01	2.920E-01	1.406E-03
31	49	40	3.905E-01	6.104E-02	1.308E-02
40	49	50	4.583E-01	1.594E-01	8.170E-04
40	50	42	4.457E-01	2.038E-01	1.495E-03
46	51	47	4.734E-01	4.150E-01	5.587E-04
44	47	51	4.701E-01	4.601E-01	8.828E-04
44	51	52	4.772E-01	4.658E-01	1.025E-03
43	44	52	4.736E-01	5.034E-01	1.108E-03
41	48	53	4.698E-01	3.307E-01	1.144E-03
41	53	45	4.690E-01	3.370E-01	1.155E-03
48	55	53	4.890E-01	3.134E-01	7.529E-04
42	50	56	4.774E-01	2.229E-01	1.759E-03
48	56	55	4.890E-01	2.725E-01	1.084E-03
42	56	48	4.743E-01	2.593E-01	1.475E-03
46	57	51	4.853E-01	4.027E-01	1.040E-03
45	53	57	4.898E-01	3.717E-01	1.378E-03
45	57	46	4.836E-01	3.850E-01	1.260E-03
49	54	58	5.028E-01	1.462E-01	8.023E-04
49	58	50	4.956E-01	1.586E-01	8.488E-04
50	58	59	5.034E-01	1.814E-01	6.959E-04
50	59	56	4.939E-01	2.145E-01	1.411E-03
49	60	54	4.459E-01	5.945E-02	6.919E-03
10	61	21	-1.318E-01	-8.156E-01	1.159E+00
132	61	10	-2.268E+00	-5.055E+00	3.372E+01
49	61	60	4.313E-01	4.682E-02	1.013E-02
31	61	49	3.804E-01	1.988E-02	2.286E-02
21	61	31	2.376E-01	-1.716E-01	1.280E-01
53	62	57	5.271E-01	3.630E-01	1.256E-03
53	55	62	5.262E-01	3.208E-01	7.979E-04
51	57	63	5.232E-01	4.337E-01	1.385E-03
51	63	52	5.182E-01	4.529E-01	1.356E-03
57	64	63	5.465E-01	4.179E-01	1.305E-03
57	62	64	5.300E-01	3.646E-01	1.219E-03
43	52	65	5.165E-01	6.123E-01	1.615E-02
52	63	65	5.340E-01	4.784E-01	1.084E-03
54	60	66	5.376E-01	1.099E-01	1.131E-03
54	66	58	5.317E-01	1.345E-01	9.864E-04
56	59	67	5.410E-01	2.279E-01	1.306E-03
55	56	67	5.370E-01	2.725E-01	1.278E-03
55	67	68	5.440E-01	2.814E-01	1.334E-03
55	68	62	5.440E-01	3.030E-01	1.125E-03
58	66	69	5.594E-01	1.673E-01	1.261E-03
58	69	59	5.547E-01	1.780E-01	1.113E-03
59	69	67	5.567E-01	2.168E-01	1.417E-03
63	64	70	5.584E-01	4.186E-01	1.196E-03
60	71	66	6.025E-01	6.670E-02	5.822E-03
60	61	71	6.062E-01	6.237E-02	6.520E-03
66	71	69	5.758E-01	1.613E-01	1.024E-03
67	69	72	5.826E-01	2.244E-01	1.058E-03
67	72	68	5.887E-01	2.712E-01	8.032E-04

63	70	73	5.794E-01	4.635E-01	7.065E-04
63	73	65	5.781E-01	4.698E-01	8.137E-04
64	74	70	5.794E-01	4.077E-01	9.771E-04
62	68	75	5.847E-01	3.312E-01	1.225E-03
62	75	64	5.840E-01	3.545E-01	1.497E-03
64	75	76	5.882E-01	3.600E-01	1.431E-03
64	76	74	5.883E-01	3.602E-01	1.424E-03
70	74	77	6.102E-01	4.213E-01	8.136E-04
70	77	73	6.127E-01	4.392E-01	6.597E-04
68	78	75	6.063E-01	3.090E-01	1.004E-03
68	72	78	6.109E-01	2.838E-01	1.320E-03
72	79	78	6.130E-01	2.827E-01	1.265E-03
74	80	77	6.173E-01	4.157E-01	6.210E-04
74	76	80	6.327E-01	3.874E-01	6.910E-04
72	81	79	6.334E-01	2.290E-01	1.167E-03
69	81	72	6.169E-01	2.122E-01	1.426E-03
69	71	81	6.216E-01	1.772E-01	1.554E-03
65	73	82	6.186E-01	5.279E-01	4.245E-03
73	77	82	6.344E-01	4.596E-01	8.256E-04
71	83	81	6.276E-01	1.720E-01	1.491E-03
61	83	71	6.462E-01	3.653E-02	1.329E-02
75	78	84	6.427E-01	3.320E-01	5.963E-04
75	84	76	6.436E-01	3.438E-01	6.417E-04
77	85	82	6.483E-01	4.531E-01	6.173E-04
77	80	85	6.565E-01	4.260E-01	4.068E-04
76	84	86	6.539E-01	3.589E-01	7.906E-04
76	86	80	6.479E-01	3.825E-01	9.027E-04
80	86	87	6.620E-01	3.959E-01	6.603E-04
80	87	85	6.654E-01	4.174E-01	5.776E-04
78	88	84	6.648E-01	3.073E-01	7.198E-04
78	79	88	6.609E-01	2.868E-01	9.724E-04
81	83	89	6.667E-01	1.774E-01	7.439E-04
84	88	90	6.892E-01	3.228E-01	6.176E-04
84	90	86	6.748E-01	3.538E-01	4.537E-04
85	87	91	6.855E-01	4.263E-01	4.556E-04
82	85	91	6.904E-01	4.675E-01	1.119E-03
79	92	88	6.773E-01	2.650E-01	1.252E-03
79	81	92	6.683E-01	2.353E-01	1.407E-03
86	90	93	6.900E-01	3.635E-01	2.990E-04
86	93	87	6.939E-01	3.863E-01	4.122E-04
81	94	92	6.838E-01	2.160E-01	1.176E-03
81	89	94	6.856E-01	2.097E-01	1.137E-03
88	92	95	6.979E-01	2.706E-01	8.127E-04
87	93	96	7.047E-01	3.925E-01	5.077E-04
87	96	91	7.092E-01	4.130E-01	6.041E-04
91	96	97	7.119E-01	4.145E-01	5.698E-04
88	95	98	7.136E-01	3.041E-01	5.966E-04
88	98	90	7.109E-01	3.206E-01	9.485E-04
90	98	99	7.269E-01	3.374E-01	1.110E-03
90	99	93	7.229E-01	3.478E-01	7.775E-04
93	99	96	7.258E-01	3.755E-01	4.218E-04
92	100	95	7.225E-01	2.586E-01	5.505E-04
92	94	100	7.295E-01	2.407E-01	6.054E-04
95	100	98	7.487E-01	2.836E-01	6.150E-04
96	99	101	7.555E-01	3.923E-01	8.235E-04
96	101	97	7.425E-01	4.118E-01	4.884E-04
91	97	102	7.268E-01	4.699E-01	1.602E-03
82	91	102	7.117E-01	4.927E-01	3.165E-03
97	101	102	7.539E-01	4.495E-01	9.550E-04
94	103	100	7.549E-01	2.229E-01	1.379E-03
89	103	94	7.320E-01	1.832E-01	1.480E-03

98	100	104	7.634E-01	2.861E-01	1.038E-03
100	103	104	7.749E-01	2.298E-01	1.688E-03
101	105	102	7.617E-01	4.491E-01	8.509E-04
99	106	101	7.710E-01	3.868E-01	1.163E-03
98	104	107	7.680E-01	2.925E-01	1.084E-03
99	107	106	7.647E-01	3.364E-01	1.304E-03
98	107	99	7.620E-01	3.339E-01	1.356E-03
102	105	108	7.917E-01	4.691E-01	7.386E-04
103	109	104	7.779E-01	2.290E-01	1.665E-03
103	110	109	7.821E-01	2.257E-01	1.492E-03
105	111	108	7.941E-01	4.687E-01	7.282E-04
106	107	112	7.983E-01	3.371E-01	5.634E-04
89	113	103	7.448E-01	1.017E-01	8.251E-03
83	113	89	7.402E-01	8.128E-02	1.130E-02
103	113	110	8.029E-01	1.768E-01	1.233E-03
101	106	114	7.898E-01	3.960E-01	1.298E-03
101	114	105	7.924E-01	4.115E-01	9.378E-04
106	115	114	8.006E-01	3.893E-01	9.306E-04
106	112	115	8.157E-01	3.620E-01	5.665E-04
108	111	116	8.201E-01	4.882E-01	4.497E-04
105	114	117	8.152E-01	4.349E-01	6.826E-04
105	117	111	8.150E-01	4.427E-01	6.253E-04
109	110	118	9.067E-01	2.324E-01	8.591E-03
110	113	118	8.108E-01	1.776E-01	1.025E-03
83	119	113	7.396E-01	4.822E-02	1.691E-02
61	119	83	7.249E-01	-4.948E-02	4.441E-02
113	119	118	8.450E-01	1.452E-01	4.926E-04
111	117	120	8.356E-01	4.591E-01	2.772E-04
111	120	116	8.339E-01	4.838E-01	4.489E-04
114	121	117	8.344E-01	4.250E-01	3.555E-04
114	115	121	8.375E-01	3.982E-01	3.515E-04
117	121	120	8.879E-01	4.410E-01	2.294E-03
104	109	122	8.263E-01	2.841E-01	1.579E-03
104	122	107	8.224E-01	2.890E-01	1.491E-03
107	122	112	8.217E-01	3.101E-01	8.377E-04
109	118	122	9.206E-01	2.370E-01	1.123E-02
115	123	121	8.428E-01	3.950E-01	3.610E-04
112	123	115	8.395E-01	3.574E-01	6.467E-04
116	120	124	8.405E-01	4.868E-01	3.883E-04
108	116	125	7.963E-01	5.666E-01	5.143E-03
120	126	124	9.168E-01	4.690E-01	4.603E-03
120	121	126	9.804E-01	4.410E-01	1.805E-02
116	126	125	8.462E-01	5.277E-01	6.885E-04
116	124	126	8.491E-01	5.198E-01	4.786E-04
102	108	127	7.115E-01	6.294E-01	2.589E-02
82	102	127	7.091E-01	6.313E-01	2.674E-02
108	125	127	7.816E-01	5.854E-01	8.503E-03
65	82	127	6.548E-01	7.189E-01	5.904E-02
43	65	127	5.630E-01	9.671E-01	2.238E-01
7	43	127	4.450E-01	1.485E+00	9.500E-01
134	7	127	-6.187E-01	1.661E+01	2.582E+02
133	134	127	2.527E+01	2.416E+01	1.150E+03
125	126	127	1.007E+00	5.635E-01	2.060E-02
112	128	123	8.490E-01	3.501E-01	8.508E-04
112	122	128	8.503E-01	3.449E-01	8.343E-04
121	128	126	9.826E-01	4.406E-01	1.860E-02
121	123	128	9.031E-01	4.021E-01	3.039E-03
126	128	127	3.900E+00	5.636E-01	9.176E+00
133	127	128	2.129E+01	1.041E+00	4.173E+02
61	129	119	8.055E-01	-3.316E-01	2.083E-01
132	129	61	4.004E+00	-1.362E+01	1.984E+02

```
  132  133  129      1.622E+01     -3.075E+01      1.188E+03
  122  129  128      9.911E-01      2.383E-01      2.595E-02
  118  129  122      9.670E-01      2.342E-01      2.038E-02
  118  119  129      8.686E-01      1.479E-01      1.129E-03
  133  128  129      2.125E+01      7.557E-01      4.151E+02
    1    2  130      1.209E-01      4.219E-01      5.716E-04
    2    5  130      6.751E-02      4.946E-01      6.003E-03
    5    7  130     -1.521E-01      5.901E-01      8.998E-02
  132  130    7     -9.626E+00      4.055E+00      1.077E+02
    4  131    6      3.837E-02      2.650E-01      1.086E-02
    3  131    4      8.064E-02      2.927E-01      3.546E-03
    3    9  131      1.323E-01      3.492E-01      1.044E-03
    1  131    9      1.234E-01      3.680E-01      9.105E-04
    1  130  131      8.111E-02      3.853E-01      1.674E-03
  132  131  130     -6.400E+00      1.224E-01      4.230E+01
  132    6  131     -4.786E+00     -1.868E+00      2.879E+01
/*
/*  Number of points where natural neighbors are to be marked and
/*  a flag indicating whether just the points where first order neighbors
/*  are desired are marked (-1), whether the first  order neighbors
/*  will be marked as well (0), or both first and second order neighbors
/*  are marked (1).  The points will be marked with Xs, in the
/*  color described above. (2I5 format)
/*
/*..+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
    1   -1
/*
/*  The coordinate list of points whose natural neighbors are to
/*  be displayed (using the color index as described above), should
/*  be listed here in 2E15.3 format.
/*
/*..+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
      0.470E-00      0.310E-00
```

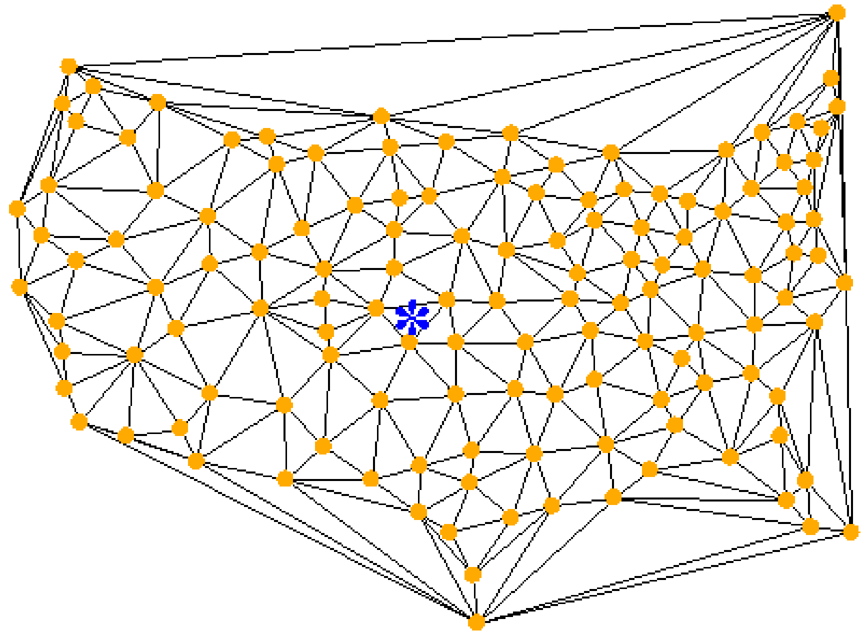
Example 9 - plotting natural neighbors

This module contains a sequence of plots. These plots were all obtained by modifying the default output file of algorithmic data produced as per the instuctions in the algorithmic display tool module and using nnalg to produce the plots. The purpose of the plots is to illustrate how to plot the natural neighbors of a given point or points. Several modifications will be made to the default output file. To fully understand what is being done, you should have read the algorithmic display tool module.

The data for this example are points in the continental U.S. with pressure readings - these data were supplied by Professor Alan Cline at the University of Texas.

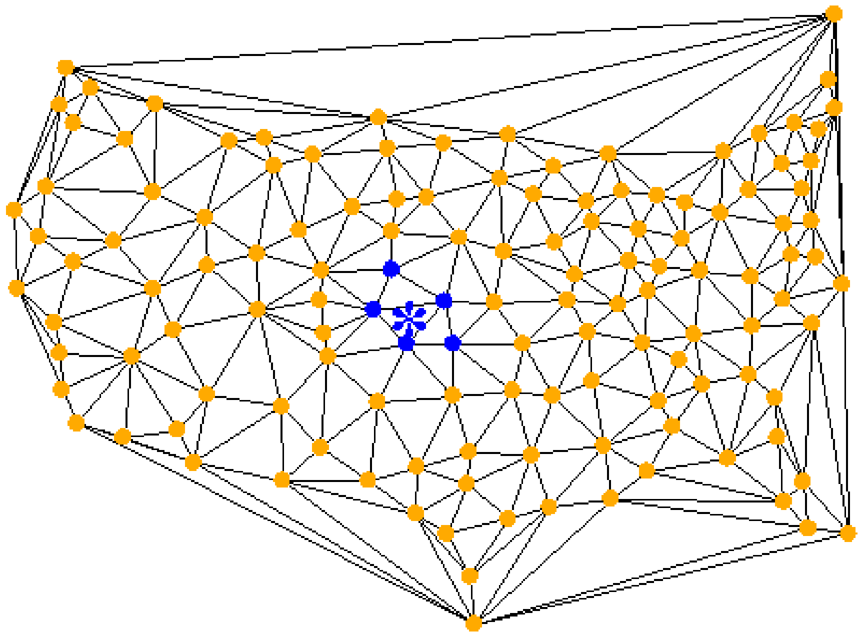
The first plot was produced using the modified output file. Several modifications were made to the original file *example9.dat* produced by running the Fortran source. For example, mods were made to suppress drawing the Voronoi polygons and the natural neighbor circumcircles and centers.

Of particular interest is the final section of the algorithmic data file where it has been indicated that we want to look at the natural neighbors of the coordinate (0.47, 0.31), but we only want to mark that point with an asterisk. Using the modified output file with nnalg produces the plot:



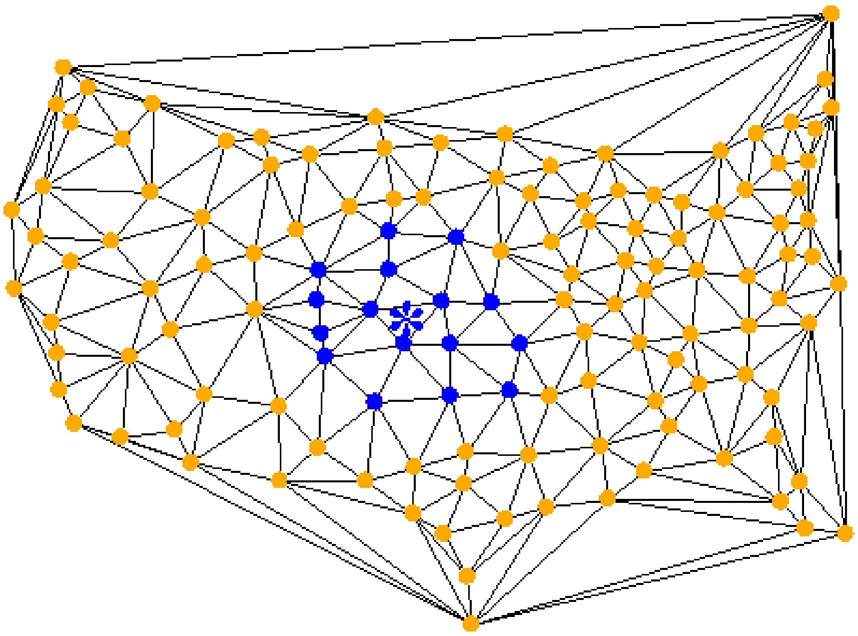
The blue asterisk marks the point at which we want to know the natural neighbors.

If we now want to see the natural neighbors of the point marked in the above plot, we change the appropriate flag, as described in the comments in the output data file, and get the plot:



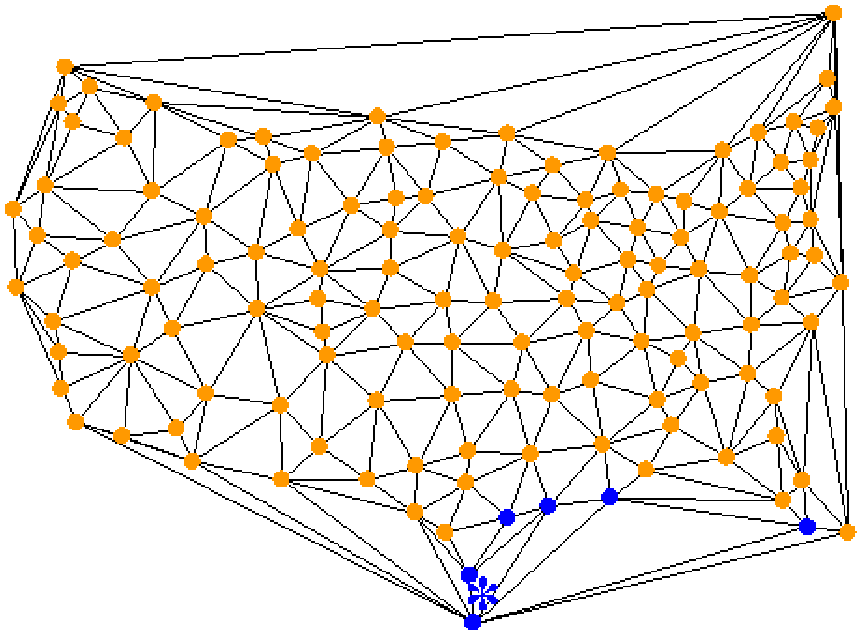
The five blue dots surrounding the asterisk indicate the natural neighbors.

If displaying the second order natural neighbors is desired, changing the flag in the data file for that gives:



The blue dots now indicate all second order natural neighbors.

Changing the interpolated point of interest to (0.540E-00, 0.050E-00) [in the southern part of Texas] and plotting the first order neighbors produces the plot:



This illustrates that for interpolated values near the convex hull of a dataset where the convex hull differs significantly from the bounding polygon, the neighbor selection may be somewhat unsatisfactory. From the above plot we see that function values in Florida can contribute to an interpolated value in Texas. However, the effect is attenuated by the fact that the weights given to distant points are small, and this greatly reduces the influence of distant points.

Natgrid References

- Akima, H., 1978, A method for bivariate interpolation and smooth surface fitting for irregularly distributed data points: ACM Trans. on Mathematical Software, v. 4, no. 2, pp. 148-159.
- Akima, H., 1978, Algorithm 526 - bivariate interpolation and smooth surface fitting for irregularly distributed data points: ACM Trans. on Mathematical Software, v. 4, no. 2, p. 160, 164.
- Sibson, R., A Brief Description of Natural Neighbor Interpolation, in *Interpreting Multivariate Data*, ed. by V. Barnett, John Wiley & Sons, New York, 1981, pp. 21-36.
- Watson, Dave, nngridr - An Implementation of Natural Neighbor Interpolation, Dave Watson Publisher, Claremont, Australia, 1994.
- Watson, Dave, Contouring: - A Guide To The Analysis and Display of Spatial Data, Pergamon Press, 1992. ISBN 0-08-040286-0

Natgrid Index

The alphabetic characters that follow entries in this index refer to citations. For example, if *entry* is followed by "A, B", then "A" and "B" are links to two references to *entry* in the Natgrid documentation modules.

A, B, C, D, E, F, G, H, I, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z

A

B

C

D

E

F

O

P

Q

R

S

T

U

V

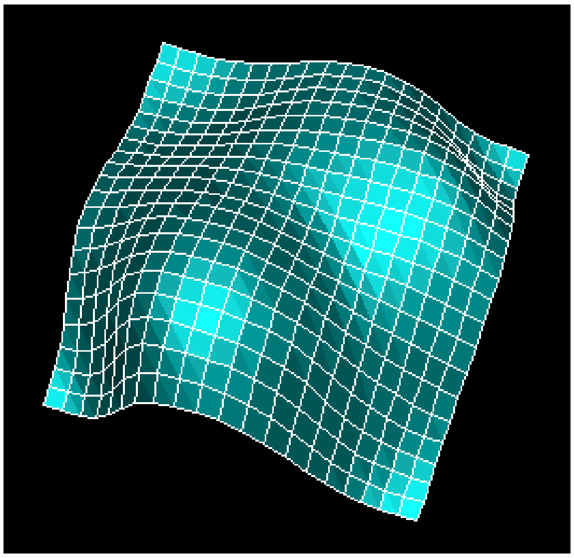
W

X

Y

Z

Dsgrid, a 3D interpolation package



Introduction (read this first)

Learning how to use the package

Dsgrid Table of contents

Credits

Introduction

Learning how to use Dsgrid

Dsgrid 3D interpolation

Dsgrid 2D interpolation

Extrapolation

Efficiency considerations

Interpolating at specified points

Error handling

Useful reference charts and tables

Examples

References

Index

Dsgrid Introduction

About Dsgrid

Dsgrid is a package in the ngmath library that implements a simple inverse distance weighted interpolation algorithm. The input for the interpolation is a set of randomly-spaced three-dimensional coordinates with functional values at those coordinates; the output is a set of interpolated function values at coordinates in a user-specified grid. The coordinates in the output grid must be monotone increasing in each coordinate direction, but need not be uniformly spaced. See the comparative analysis of the ngmath packages for details on which ngmath package is most appropriate for your problem.

General background on interpolation methods

Computational interpolation methods can be divided into two basic classes: fitted function methods and weighted average methods.

The fitted function, or fitted surface, methods fit an algebraic surface to the known data and then pick the interpolated values from the fitted surface. The weighted average methods calculate interpolated values as weighted averages of known values.

Among the weighted average methods, the simplest are known as the inverse distance weighted average methods. Dsgrid uses one of these methods.

The interpolation method used in Dsgrid

Dsgrid uses a simple weighted average method to do its interpolation. For interpolation at a coordinate, weights are computed for input coordinates, the values of such weights being inversely proportional to the distance an input coordinate is from the point where an interpolated value is desired. In the default situation in the Dsgrid package, the influence of an input datum on a calculated value at an interpolation point varies as the inverse cube of the distance between the interpolated point and the input datum. The sum of the weights is normalized to unity and an interpolated value is the sum of the products of the known functional values and the calculated weights.

The capability exists in Dsgrid to vary the power of the distances used to compute the weights. For powers of the distances greater than one, the interpolated surface is flat at the input data; the size of the flat spot increases with increasing values of the power. If the power of the distances is one, then the weights are computed simply as linear inverse distances and the areas around input data are cone shaped. If the power is less than one, then the areas around input data form a cusp at the data points. Example 2 illustrates the effects of modifying the exponent of the distances used in computing the weights. For full details on interpolation methods, consult Dave Watson's book *Contouring*.

Dsgrid functional overview

- Functionally equivalent Fortran and C interfaces are provided in both single and double precision versions.
- Interfaces exist for two-dimensional interpolation as well as 3D.
- Interfaces exist for interpolation at individual points.
- The capability exists to specify that only input data points within a specified distance from an interpolated point be considered in the calculation of weights. In the default case, all input data are used in the calculation of the weights.
- The package has a shadowing option which, when in effect, reduces the effect that a cluster of points might have in skewing interpolated values. This shadowing feature is based on original work of Bob Lackman. Example 3 illustrates the shadowing feature.

Learning how to use Dsgrid

You should now proceed to the module on learning how to use Dsgrid.

Learning how to use Dsgrid

The programmatic interfaces

There are four interfaces to choose from: Fortran or C in either double precision or single precision.

Each interface has one procedure for doing the gridding. Without worrying about understanding exactly what all the procedures do, browse the table of procedure names at this time. Notice that details on any of the procedures can be obtained by clicking on the procedure name.

A program using Dsgrid will contain a section that sets values for appropriate control parameters, after which one of the gridding procedures will be invoked. When Dsgrid is used in its default environment, you will not have to set any control parameters. Example 4 illustrates a simple use of the 3D interpolator.

To get a feeling for the available control parameters (again without worrying about understanding exactly what all of the parameters are), examine the table of control parameters at this time.

Complete details on all procedures and control parameters are presented in other modules.

Examples are provided illustrating the functionality of Dsgrid. All of the examples are in the Examples module. Source codes and example plots are presented there, as well as instructions on how to create your own executables. Browse the introductory remarks in the Examples module now.

Learning guide

To get a feel for what is in the package, quickly browse the table of contents.

Read through the following modules as appropriate for details on how to use Dsgrid.

- 3D interpolation
- 2D interpolation
- Error handling

Dsgrid credits

Acknowledgments

Author: Fred Clare

Editor: Fred Clare. Final copyedit by Brian Bevirt.

Production: Fred Clare

Special thanks: To Bob Lackman for sharing his shadowing technique, and to Dick Valent for alerting the author of the need for this package.

Copyright

Trademark

Dsgrid 3D interpolation

Simplest usage

If the default settings for the control parameters are what you want, then you can interpolate on a grid by invoking a single procedure. Example 1 presents a simple 3D interpolation with codes using each of the four programmatic interfaces.

Algorithmic details, control paramters

Given a point where an interpolated value is desired, the distance from that point to each of the known input points is calculated. Weights for each of the input points are calculated that are inversely proportional to the calculated distances raised to a power. The sum of the weights is normalized to unity and the interpolated value is the sum of the products of the function values at the input points and the weights.

The control parameter exp can be used to specify the power of the distances used in calculating the weights -- by default this value is 3.0. Example 2 illustrates the effect of varying the exponent used for the distances in the weight calculations. When the exponent is 1., cone-like surfaces are formed around the input data values; for values of the exponent greater than 1., flat spots occur around the input data values (the larger the exponent, the larger the flat spot); for values of the exponent smaller than 1., cusps are formed around the input data points.

The control parameter dmx can be used to supply a radius around an interpolation point outside of which input data points will not be used to calculate weights. If no input data points lie within the specified radius for a given interpolation point, the interpolated value is set to a special missing value. This missing value can be set using the control parameter dmv -- the default value is -9999.

Smoothness

Due to the fact that the algorithm produces cusps or flat spots around the input data points (depending on the exponent of the distances as explained above), the interpolated surfaces or isosurfaces are, in general, not particularly smooth. Example 3, which is an approximation of an analytic function defined at random data points, illustrates this. In particular, compare the Dsgrid plot with the plot of the same interpolation done using Natgrid.

Efficiency

For details on the efficiency of the algorithm, see the efficiency module.

Single point mode for 3D

Dsgrid can be used to interpolate at individual points. The Fortran procedures for this are DSPNT3S and DSPNT3D; and the equivalent C procedures are c_dspnt3s and c_dspnt3d.

You can set any control parameters you want before interpolating at individual points.

For the same number of input and output points, using the Dsgrid entries for interpolation at individual points invokes about a 30% overhead compared to using the Dsgrid entries to interpolate on a grid.

Extrapolation

In general extrapolation is not very reliable and the results so obtained are to be viewed with some lack of confidence. In order for extrapolation to be at all reliable, the original data must be very consistent.

Outside of the convex hull, the interpolated values in Dsgrid tend quickly toward a constant value. This constant is the arithmetic average of the input functional values. Example 5 illustrates this. It presents the function f(x,y) = (x-1/4)**2 + (y-1/2)**2 defined on the unit square and extrapolated to a rectangular region beyond that square.

Using the shadowing feature

Dsgrid contains an option that can be used to prevent a close grouping of input data points with similar function values from dominating the surrounding interpolation values. The control parameter shd can be used to specify whether shadowing is on or off. By default it is off, since using the shadowing feature slows the computations by a factor of about 300.

The idea of the shadowing algorithm is that, given a point where an interpolated value is desired, the weights for the input data values are decreased by a factor that depends on the extent to which one input data point is "behind" another with respect to the point of interpolation. For example, if two input data values lie on a straight line emanating from the interpolation point, then the data value that is farther from the input data point will receive a weighting factor of zero. In general an angle is computed measuring how close a data value is to any other, and the tangent of this angle is used to weight the input data value.

Example 4 illustrates how this works. In that example we have a group of points clustered about the input coordinate (0.7, 0.7). Compare the plot with the shadowing feature turned on to the plot with the shadowing feature turned off. Notice that in the latter plot, the values at the cluster of points dominate the interpolated values in the surrounding area (the large smooth mound).

Dsgrid 2D interpolation

A word of advice

The interpolation algorithm in the Natgrid package is greatly superior to that used in Dsgrid. The Natgrid package also has more options available such as smoothing parameters, prevention of extrapolation, the forcing of non-negative values, and the computation of slopes and aspects. For 2D interpolation it is advised that you use Natgrid, unless you have some specific desire to use the algorithm in Dsgrid.

Using Dsgrid for 2D interpolation

Using the 2D procedures in Dsgrid (listed in the table of procedures) is entirely analogous to using the 3D procedures. Reading the module on 3D interpolation will be sufficient in instructing you on how to use the 2D procedures.

The examples module presents several examples of using Dsgrid to interpolate in two dimensions.

Single point mode for 2D

The Fortran procedures for this are DSPNT2S and DSPNT2D; and the equivalent C procedures are c_dspnt2s and c_dspnt2d.

Dsgrid Efficiency considerations

Main time components

The time required for executing Dsgrid codes depends essentially on the number of input points and the number of output points.

For a fixed number of input points, the time varies in direct proportion to the number of output points; for a fixed number of output points, the time varies in direct proportion to the number of input points.

In the default case, all input points are used in forming the weighted average used in computing a given interpolated value. Execution time can be improved if you restrict yourself to using the number of input data values points within a fixed distance of the points where interpolated values are desired. The parameter to use to specify this fixed distance is dmx.

Detailed timing results

A table summarizing a timing analysis done on a Sun sparc Ultra-1 is provided if you want details.

General conclusions

Using Dsgrid can be expensive.

For the same number of input and output points, using the Dsgrid entries for interpolation at individual points invokes about a 30% overhead compared to using the Dsgrid entries to interpolate on a grid.

The error handling functions

Dsgrid contains a set of procedures that implement a centralized error handling capability. The package invokes the C function DSErrorHnd to produce all of its error messages. DSErrorHnd simply invokes the DSErrorLog function with the same arguments as DSErrorHnd. This allows a user to replace the error handling functionality by replacing or modifying the DSErrorHnd function with one that handles errors in an application-dependent manner. All of the error handling functions are written in C; there is no way to replace the error handler using Fortran code.

Detailed description of the error functions:

The error functions are:

- DSErrorHnd - primary error handler
- DSErrorLog - error logger
- DSErrMsg - retrieves error messages
- DSErrMax - retrieves the number of possible error messages

DSErrorHnd - Primary error handler

DSErrorHnd simply invokes the DSErrorLog function with the arguments specified in DSErrorHnd. This allows a user to replace the error handling functionality by replacing or modifying the DSErrorHnd function with one that handles errors in an application-dependent manner.

Function prototype:

```
void DSErrorHnd (int, char *, FILE *, char *);
```

Argument description:

```
-----------------------------------
                Argument | Type
-----------------------------------
```

```
void DSErrorHnd (error,    | int
                 func,     | char *
                 efile,    | FILE *
                 smsg);    | char *
-----------------------------------
```

error
: The error number, as described in the error table.

func
: A string specifying the function that is invoking the error call.

efile
: Specifies the file where the error message will be written. By default error messages are written to stderr.

smsg
: The error message; this will usually be DSErrMsg(i) where i is an error number.

DSErrorLog - error logger

Function prototype:

```
void DSErrorLog (int, char *, FILE *, char *);
```

Argument description:

```
----------------------------------
                Argument | Type
----------------------------------
void DSErrorLog (error,  | int
                 func,   | char *
                 efile,  | FILE *
                 smsg);  | char *
----------------------------------
```

error
: The error number, as described in the error table.

func
: A string specifying the function that is invoking the error call.

efile
: Specifies the file where the error message will be written. By default error messages are written to stderr.

smsg
: The error message; this will usually be DSErrMsg(i) where i is an error number.

DSErrMsg - retrieves error messages

Function prototype:

```
char *DSErrMsg(int i)
```

Return value:

DSErrMsg returns the error message associated with its integer argument.

Argument description:

Argument	Type
char *DSErrMsg (error_num);	int

error_num
An error number.

DSErrMax - retrieves the number of possible error messages

Function prototype:

```
int DSErrMax()
```

Return value:

The value returned is the total number of possible error messages.

Error table

The following table lists all of the error numbers and the associated error messages:

Error number	Error message
1	Error number out of range.
2	Insufficient data in gridded region to triangulate.
3	Array dimension out of range.
4	Parameter name not known.
5	Cannot open error file.
6	Error allocating memory for input points.
7	Fortran DOUBLE PRECISION entries are not supported on UNICOS.
9	Error allocating memory for array of distances between input points.
10	Error allocating memory for weights.
11	Error allocating memory for distance ordering vector.
12	Error allocating memory for output array.
13	Number of input points must be greater than 2.
14	No original data values within the specified distance - interpolated value set to missing value flag.

Table of Dsgrid control parameters

The following table summarizes all of the parameters used for controlling the behavior of Dsgrid.

Values for these parameters can be set by using the subroutines DSSETI, DSSETR, DSSETRD, and DSSETC for Fortran programs or the functions c_dsseti, c_dssetr, c_dssetrd, and c_dssetc for C programs.

Parameter name	parameter type	description	legal values	default value
dmv	float	a special value for data initialization and use with the **dmx** control parameter.	any	-9999.
dmx	float	in calculating weights, use only input data within the distance dmx.	> 0.	use all input data points
erf	char	error file.	any file name	stderr
exp	float	power to use for inverse distances in computing weights.	> 0.	3.0
shd	int	controls whether the shadowing feature is on.	0 = no 1 = yes	0

Overview of Dsgrid procedures

The table below summarizes all of the procedures in Dsgrid for the four user interfaces (single and double precision in both Fortran and C).

Clicking on the procedure name will link to a detailed description of that procedure.

```
Fortran single precision:
------------------------

 DSGRID2S  -  primary function for gridding 2D data.
 DSGRID3S  -  primary function for gridding 3D data.
 DSSETI    -  set INTEGER parameter values.
 DSGETI    -  retrieve values for INTEGER parameters.
 DSSETR    -  set REAL parameter values.
 DSGETR    -  retrieve values for REAL parameters.
 DSSETC    -  set CHARACTER valued parameters.
 DSGETC    -  retrieve values for CHARACTER parameters.
```

```
 DSPNT2S   -  interpolate 2D data at specified individual points.
 DSPNT3S   -  interpolate 3D data at specified individual points.
```

```
Fortran double precision (not supported on UNICOS):
--------------------------------------------------

 DSGRID2D  -  primary function for gridding 2D data.
 DSGRID3D  -  primary function for gridding 3D data.
 DSSETI    -  set INTEGER parameter values.
 DSGETI    -  retrieve values for INTEGER parameters.
 DSSETRD   -  set DOUBLE PRECISION parameter values.
 DSGETRD   -  retrieve values for DOUBLE PRECISION parameters.
 DSSETC    -  set CHARACTER valued parameters.
 DSGETC    -  retrieve values for CHARACTER parameters.
 DSPNT2D   -  interpolate 2D data at specified individual points.
 DSPNT3D   -  interpolate 3D data at specified individual points.
```

```
C single precision:
------------------

 c_dsgrid2s  -  primary function for gridding 2D data.
 c_dsgrid3s  -  primary function for gridding 3D data.
 c_dsseti    -  set int parameter values.
 c_dsgeti    -  retrieve values for int parameters.
 c_dssetr    -  set float parameter values.
 c_dsgetr    -  retrieve values for float parameters.
 c_dssetc    -  set char valued parameters.
 c_dsgetc    -  retrieve values for char parameters.
 c_dspnt2s   -  interpolate 2D data at specified individual points.
 c_dspnt3s   -  interpolate 3D data at specified individual points.
```

```
C double precision:
------------------

 c_dsgrid2d  -  primary function for gridding 2D data.
 c_dsgrid3d  -  primary function for gridding 3D data.
 c_dsseti    -  set int parameter values.
 c_dsgeti    -  retrieve values for int parameters.
 c_dssetrd   -  set double parameter values.
 c_dsgetrd   -  retrieve values for double parameters.
 c_dssetc    -  set char valued parameters.
 c_dsgetc    -  retrieve values for char parameters.
 c_dspnt2d   -  interpolate 2D data at specified individual points.
 c_dspnt3d   -  interpolate 3D data at specified individual points.
```

Detailed description of the procedures

DSGRID2S - primary 2D gridding function

DSGRID2S is the Fortran single precision subroutine that does an interpolation from 2D random data to an output grid. DSGRID2S is called after all of the desired values for control parameters have been set using the subroutines DSSETI, DSSETR, DSSETC.

```
---------------------------------------------------------------
               Argument | Type     |  Mode   | Dimension
---------------------------------------------------------------
CALL DSGRID2S (NPNTS,   | Integer  | Input   |
               X,       | Real     | Input   | NPNTS
               Y,       | Real     | Input   | NPNTS
               Z,       | Real     | Input   | NPNTS
               NUMXOUT, | Integer  | Input   |
               NUMYOUT, | Integer  | Input   |
               XO,      | Real     | Input   | NUMXOUT
               YO,      | Real     | Input   | NUMYOUT
               ZO,      | Real     | Output  | NUMXOUT x NUMYOUT
               IER)     | Integer  | Output  |
---------------------------------------------------------------
```

NPNTS
: The number of input data points.

X
: An array containing the X coordinates of the input data points.

Y
: An array containing the Y coordinates of the input data points.

Z
: An array containing the functional values of the input data points. That is, Z(L) is the value of the input function at coordinate (X(L),Y(L)), for L=1,NPNTS.

NUMXOUT
: The number of X values in the output grid.

NUMYOUT
: The number of Y values in the output grid.

XO
: An array of dimension NUMXOUT containing the X coordinates of the output data grid. The values in XO must be increasing, but need not be equally spaced.

YO
: An array of dimension NUMYOUT containing the Y coordinates of the output data grid. The values in YO must be increasing, but need not be equally spaced.

ZO
: A two-dimensional array of dimension NUMXOUT x NUMYOUT containing the interpolated functional values. ZO(I,J) is the interpolated value at grid point (XO(I),YO(J)).

IER
: An error return value. If IER is returned as 0, then no errors were detected. If IER is non-zero, then refer to the error list in the error table for details.

DSGRID3S - primary 3D gridding function

DSGRID3S is the Fortran single precision subroutine that does an interpolation from 3D random data to an output grid. DSGRID3S is called after all of the desired values for control parameters have been set using the subroutines DSSETI, DSSETR, DSSETC.

```
--------------------------------------------------------------------------
               Argument | Type     |  Mode   | Dimension
--------------------------------------------------------------------------
CALL DSGRID3S (NPNTS,   | Integer | Input   |
               X,       | Real    | Input   | NPNTS
               Y,       | Real    | Input   | NPNTS
               Z,       | Real    | Input   | NPNTS
               U,       | Real    | Input   | NPNTS
               NUMXOUT, | Integer | Input   |
               NUMYOUT, | Integer | Input   |
               NUMZOUT, | Integer | Input   |
               XO,      | Real    | Input   | NUMXOUT
               YO,      | Real    | Input   | NUMYOUT
               ZO,      | Real    | Input   | NUMZOUT
               UO,      | Real    | Output  | NUMXOUT x NUMYOUT x NUMZOUT
               IER)     | Integer | Output  |
--------------------------------------------------------------------------
```

NPNTS
: The number of input data points.

X
: An array containing the X coordinates of the input data points.

Y
: An array containing the Y coordinates of the input data points.

Z
: An array containing the Z coordinates of the input data points.

U
: An array containing the functional values of the input data points. That is, U(L) is the value of the input function at coordinate (X(L),Y(L),Z(L)), for L=1,NPNTS.

NUMXOUT
: The number of X values in the output grid.

NUMYOUT
: The number of Y values in the output grid.

NUMZOUT
: The number of Z values in the output grid.

XO
: An array of dimension NUMXOUT containing the X coordinates of the output data grid. The values in XO must be increasing, but need not be equally spaced.

YO
: An array of dimension NUMYOUT containing the Y coordinates of the output data grid. The values in YO must be increasing, but need not be equally spaced.

ZO
: An array of dimension NUMZOUT containing the Z coordinates of the output data grid. The values in ZO must be increasing, but need not be equally spaced.

UO
: A three-dimensional array of dimension NUMXOUT x NUMYOUT x NUMZOUT containing the interpolated functional values. UO(I,J,K) is the interpolated value at grid point (XO(I),YO(J),ZO(K)).

IER
: An error return value. If IER is returned as 0, then no errors were detected. If IER is non-zero, then

refer to the error list in the error table for details.

DSSETI - set INTEGER parameter values

DSSETI is used to set values for any of the control parameters that take integer values. The values set by DSSETI remain in effect until changed by subsequent calls to DSSETI.

```
-----------------------------------------------------
            Argument | Type        |  Mode  | Dimension
-----------------------------------------------------
CALL DSSETI  (PNAM,  | Character   | Input  |
              IVAL)  | Integer     | Input  |
-----------------------------------------------------
```

PNAM
    The name of the control parameter to be set.
IVAL
    The value to be assigned to the parameter.

DSGETI - retrieve values for INTEGER parameters

DSGETI is a called to obtain current values for any of the INTEGER valued control parameters.

```
-----------------------------------------------------
            Argument | Type        |  Mode  | Dimension
-----------------------------------------------------
CALL DSGETI  (PNAM,  | Character   | Input  |
              IVAL)  | Integer     | Output |
-----------------------------------------------------
```

PNAM
    The name of the control parameter whose value is to be retrieved.
IVAL
    The current value assigned to the control parameter.

DSSETR - set REAL parameter values

DSSETR is used to set values for any of the control parameters that take REAL values. The values set by DSSETR remain in effect until changed by subsequent calls to DSSETR.

```
-----------------------------------------------------
            Argument | Type        |  Mode  | Dimension
-----------------------------------------------------
CALL DSSETR  (PNAM,  | Character   | Input  |
              RVAL)  | Real        | Input  |
-----------------------------------------------------
```

PNAM

The name of the control parameter to be set.

RVAL

The value to be assigned to the parameter.

DSGETR - retrieve values for REAL parameters

DSGETR is a called to obtain current values for any of the REAL valued control parameters.

```
-----------------------------------------------------
            Argument | Type       |  Mode   | Dimension
-----------------------------------------------------
CALL DSGETR (PNAM,   | Character  | Input   |
             RVAL)   | Real       | Output  |
-----------------------------------------------------
```

PNAM

The name of the control parameter whose value is to be retrieved.

RVAL

The current value assigned to the control parameter.

DSSETC - set CHARACTER valued parameters.

```
-----------------------------------------------------
            Argument | Type       |  Mode   | Dimension
-----------------------------------------------------
CALL DSSETC (PNAM,   | Character  | Input   |
             CVAL)   | Character  | Input   |
-----------------------------------------------------
```

DSSETC is used to set values for any of the control parameters that take CHARACTER values. The values set by DSSETC remain in effect until changed by subsequent calls to DSSETC.

PNAM

The name of the control parameter to be set.

CVAL

The value to be assigned to the parameter.

DSGETC - retrieve values for CHARACTER parameters

DSGETC is a called to obtain current values for any of the CHARACTER valued control parameters.

```
-----------------------------------------------------
            Argument | Type       |  Mode   | Dimension
-----------------------------------------------------
CALL DSGETC (PNAM,   | Character  | Input   |
             CVAL)   | Character  | Output  |
-----------------------------------------------------
```

PNAM
: The name of the control parameter whose value is to be retrieved.

CVAL
: The current value assigned to the control parameter. Be sure to provide enough space in the character variable to receive the value.

DSPNT2S - interpolate 2D data at specified points

This subroutine is called to interpolate 2D data at specified points.

```
--------------------------------------------------------
              Argument | Type      |  Mode    | Dimension
--------------------------------------------------------
CALL DSPNT2S  (N,      | Integer   | Input    |
               X,      | Real      | Input    |  N
               Y,      | Real      | Input    |  N
               Z,      | Real      | Input    |  N
               M,      | Integer   | Input    |
               XO,     | Real      | Input    |  M
               YO,     | Real      | Input    |  M
               ZO,     | Real      | Output   |  M
               IER)    | Integer   | Output   |
--------------------------------------------------------
```

N
: The number of input data points.

X
: An array containing the X coordinates of the input data points.

Y
: An array containing the Y coordinates of the input data points.

Z
: An array containing the functional values of the input data points. That is, Z(L) is the value of the input function at coordinate (X(L),Y(L)), for L=1,N.

M
: The number of output data points (this may be "1").

XO
: An array of dimension M containing the X coordinates of the output data. The values in XO may be in any order.

YO
: An array of dimension M containing the Y coordinates of the output data. The values in YO may be in any order.

ZO
: An array of dimension M containing the interpolated values. ZO(I) is the interpolated value at point (XO(I),YO(I)) for I=1,M.

IER
: An error return value. If IER is returned as 0, then no errors were detected. If IER is non-zero, then refer to the error list in the error table for details.

DSPNT3S - interpolate 3D data at specified points

This subroutine is called to interpolate 3D data at specified points.

	Argument	Type	Mode	Dimension
CALL DSPNT3S	(N,	Integer	Input	
	X,	Real	Input	N
	Y,	Real	Input	N
	Z,	Real	Input	N
	U,	Real	Input	N
	M,	Integer	Input	
	XO,	Real	Input	M
	YO,	Real	Input	M
	ZO,	Real	Input	M
	UO,	Real	Output	M
	IER)	Integer	Output	

N
: The number of input data points.

X
: An array containing the X coordinates of the input data points.

Y
: An array containing the Y coordinates of the input data points.

Z
: An array containing the Z coordinates of the input data points.

U
: An array containing the functional values of the input data points. That is, U(L) is the value of the input function at coordinate (X(L),Y(L),Z(L)), for L=1,N.

M
: The number of output data points (this may be "1").

XO
: An array of dimension M containing the X coordinates of the output data. The values in XO may be in any order.

YO
: An array of dimension M containing the Y coordinates of the output data. The values in YO may be in any order.

ZO
: An array of dimension M containing the Z coordinates of the output data. The values in ZO may be in any order.

UO
: An array of dimension M containing the interpolated values. UO(I) is the interpolated value at point (XO(I),YO(I),ZO(I)) for I=1,M.

IER
: An error return value. If IER is returned as 0, then no errors were detected. If IER is non-zero, then refer to the error list in the error table for details.

DSGRID2D - primary 2D gridding function

DSGRID2D is the Fortran double precision subroutine that does an interpolation from 2D random data to an output grid. DSGRID2D is called after all of the desired values for control parameters have been set using the subroutines DSSETI, DSSETRD, DSSETC.

```
--------------------------------------------------------------------------
               Argument |            Type     |  Mode   | Dimension
--------------------------------------------------------------------------
CALL DSGRID2D (NPNTS,   | Integer             | Input   |
               X,       | Double precision    | Input   | NPNTS
               Y,       | Double precision    | Input   | NPNTS
               Z,       | Double precision    | Input   | NPNTS
               NUMXOUT, | Integer             | Input   |
               NUMYOUT, | Integer             | Input   |
               XO,      | Double precision    | Input   | NUMXOUT
               YO,      | Double precision    | Input   | NUMYOUT
               ZO,      | Double precision    | Output  | NUMXOUT x NUMYOUT
               IER)     | Integer             | Output  |
--------------------------------------------------------------------------
```

NPNTS
: The number of input data points.

X
: An array containing the X coordinates of the input data points.

Y
: An array containing the Y coordinates of the input data points.

Z
: An array containing the functional values of the input data points. That is, Z(L) is the value of the input function at coordinate (X(L),Y(L)), for L=1,NPNTS.

NUMXOUT
: The number of X values in the output grid.

NUMYOUT
: The number of Y values in the output grid.

XO
: An array of dimension NUMXOUT containing the X coordinates of the output data grid. The values in XO must be increasing, but need not be equally spaced.

YO
: An array of dimension NUMYOUT containing the Y coordinates of the output data grid. The values in YO must be increasing, but need not be equally spaced.

ZO
: A two-dimensional array of dimension NUMXOUT x NUMYOUT containing the interpolated functional values. ZO(I,J) is the interpolated value at grid point (XO(I),YO(J)).

IER
: An error return value. If IER is returned as 0, then no errors were detected. If IER is non-zero, then refer to the error list in the error table for details.

DSGRID3D - primary 3D gridding function

DSGRID3D is the Fortran double precision subroutine that does an interpolation from 3D random data to an output grid. DSGRID3D is called after all of the desired values for control parameters have been set using the subroutines DSSETI, DSSETRD, DSSETC.

```
--------------------------------------------------------------------------------
               Argument | Type               |  Mode  | Dimension
--------------------------------------------------------------------------------
CALL DSGRID3D (NPNTS,   | Integer            | Input  |
               X,       | Double precision   | Input  | NPNTS
               Y,       | Double precision   | Input  | NPNTS
               Z,       | Double precision   | Input  | NPNTS
               U,       | Double precision   | Input  | NPNTS
               NUMXOUT, | Integer            | Input  |
               NUMYOUT, | Integer            | Input  |
               NUMZOUT, | Integer            | Input  |
               XO,      | Double precision   | Input  | NUMXOUT
               YO,      | Double precision   | Input  | NUMYOUT
               ZO,      | Double precision   | Input  | NUMZOUT
               UO,      | Double precision   | Output | NUMXOUT x NUMYOUT x NUMZOUT
               IER)     | Integer            | Output |
--------------------------------------------------------------------------------
```

NPNTS
: The number of input data points.

X
: An array containing the X coordinates of the input data points.

Y
: An array containing the Y coordinates of the input data points.

Z
: An array containing the Z coordinates of the input data points.

U
: An array containing the functional values of the input data points. That is, U(L) is the value of the input function at coordinate (X(L),Y(L),Z(L)), for L=1,NPNTS.

NUMXOUT
: The number of X values in the output grid.

NUMYOUT
: The number of Y values in the output grid.

NUMZOUT
: The number of Z values in the output grid.

XO
: An array of dimension NUMXOUT containing the X coordinates of the output data grid. The values in XO must be increasing, but need not be equally spaced.

YO
: An array of dimension NUMYOUT containing the Y coordinates of the output data grid. The values in YO must be increasing, but need not be equally spaced.

ZO
: An array of dimension NUMZOUT containing the Z coordinates of the output data grid. The values in ZO must be increasing, but need not be equally spaced.

UO
: A three-dimensional array of dimension NUMXOUT x NUMYOUT x NUMZOUT containing the interpolated functional values. UO(I,J,K) is the interpolated value at grid point (XO(I),YO(J),ZO(K)).

IER
: An error return value. If IER is returned as 0, then no errors were detected. If IER is non-zero, then

refer to the error list in the error table for details.

DSSETRD - set DOUBLE PRECISION parameter values

DSSETRD is used to set values for any of the control parameters that take DOUBLE PRECISION values. The values set by DSSETRD remain in effect until changed by subsequent calls to DSSETRD.

Argument	Type	Mode	Dimension
CALL DSSETRD (PNAM,	Character	Input	
RVAL)	Real	Input	

PNAM
  The name of the control parameter to be set.
RVAL
  The value to be assigned to the parameter.

DSGETRD - retrieve values for DOUBLE PRECISION parameters

DSGETRD is a called to obtain current values for any of the DOUBLE PRECISION valued control parameters.

Argument	Type	Mode	Dimension
CALL DSGETRD (PNAM,	Character	Input	
RVAL)	DOUBLE precision	Output	

PNAM
  The name of the control parameter whose value is to be retrieved.
RVAL
  The current value assigned to the control parameter.

DSPNT2D - interpolate 2D double precision data at specified points

This subroutine is called to interpolate 2D double precision data at specified points.

Argument	Type	Mode	Dimension
CALL DSPNT2D (N,	Integer	Input	

X,	Double precision	Input	N
Y,	Double precision	Input	N
Z,	Double precision	Input	N
M,	Integer	Input	
XO,	Double precision	Input	M
YO,	Double precision	Input	M
ZO,	Double precision	Output	M
IER)	Integer	Output	

N
: The number of input data points.

X
: An array containing the X coordinates of the input data points.

Y
: An array containing the Y coordinates of the input data points.

Z
: An array containing the functional values of the input data points. That is, Z(L) is the value of the input function at coordinate (X(L),Y(L)), for L=1,N.

M
: The number of output data points (this may be "1").

XO
: An array of dimension M containing the X coordinates of the output data. The values in XO may be in any order.

YO
: An array of dimension M containing the Y coordinates of the output data. The values in YO may be in any order.

ZO
: An array of dimension M containing the interpolated values. ZO(I) is the interpolated value at point (XO(I),YO(I)) for I=1,M.

IER
: An error return value. If IER is returned as 0, then no errors were detected. If IER is non-zero, then refer to the error list in the error table for details.

DSPNT3D - interpolate 3D double precision data at specified points

---

This subroutine is called to interpolate 3D double precision data at specified points.

Argument	Type	Mode	Dimension
CALL DSPNT3D (N,	Integer	Input	
X,	Double precision	Input	N
Y,	Double precision	Input	N
Z,	Double precision	Input	N
U,	Double precision	Input	N
M,	Integer	Input	
XO,	Double precision	Input	M
YO,	Double precision	Input	M
ZO,	Double precision	Input	M

```
            UO,       | Double precision | Output | M
            IER)      | Integer          | Output |
---------------------------------------------------------------
```

N
: The number of input data points.

X
: An array containing the X coordinates of the input data points.

Y
: An array containing the Y coordinates of the input data points.

Z
: An array containing the Z coordinates of the input data points.

U
: An array containing the functional values of the input data points. That is, U(L) is the value of the input function at coordinate (X(L),Y(L),Z(L)), for L=1,N.

M
: The number of output data points (this may be "1").

XO
: An array of dimension M containing the X coordinates of the output data. The values in XO may be in any order.

YO
: An array of dimension M containing the Y coordinates of the output data. The values in YO may be in any order.

ZO
: An array of dimension M containing the Z coordinates of the output data. The values in ZO may be in any order.

UO
: An array of dimension M containing the interpolated values. UO(I) is the interpolated value at point (XO(I),YO(I),ZO(I)) for I=1,M.

IER
: An error return value. If IER is returned as 0, then no errors were detected. If IER is non-zero, then refer to the error list in the error table for details.

c_dsgrid2s - Primary gridding function

---

c_dsgrid2s is the C single precision function that does an interpolation from 2D random data to an output grid. c_dsgrid2s is called after all of the desired values for control parameters have been set using the function c_dsseti, c_dssetr, c_dssetc.

Function prototype:

```
float *c_dsgrid2s(int, float [], float [], float [],
                  int, int, float [], float [], int *);
```

Return value:

c_dsgrid2s returns a pointer to a linear array of data that is the interpolated grid stored in row-major order. That is, if *out* is declared as

```
  float *out;
```

and we set:

```
  out = c_dsgrid2s(npnts, x, y, z, numxout, numyout, xo, yo, &ier);
```

then out[i*numyout+j] is the interpolated function value at coordinate point (xo[i],yo[j]) for 0 <= i < numxout and 0 <= j < numyout. The space for *out* is allocated internal to *c_dsgrid2s* and is numxout*numyout floats in size.

Argument description:

```
-------------------------------------------------
                   Argument | Type      |  Size
-------------------------------------------------
float *c_dsgrid2s (npnts,   | int       |
                   x,       | float []  | npnts
                   y,       | float []  | npnts
                   z,       | float []  | npnts
                   numxout, | int       |
                   numyout, | int       |
                   xi,      | float []  | numxout
                   yi,      | float []  | numyout
                   ier      | int *     |
                   );
-------------------------------------------------
```

npnts
: The number of input data points.

x
: An array of size npnts containing the X coordinates of the input data points.

y
: An array of size npnts containing the Y coordinates of the input data points.

z
: An array of size npnts containing the functional values of the input data points. That is, z[n] is the value of the input function at coordinate (x[n],y[n]), for 0 <= n < npnts.

numxout
: The number of X values in the output grid.

numyout
: The number of Y values in the output grid.

xi
: An array of size numxout containing the X coordinates of the output data grid. The values in xi must be increasing, but need not be equally spaced.

yi
: An array of size numyout containing the Y coordinates of the output data grid. The values in yi must be increasing, but need not be equally spaced.

ier
: An error return value. If *ier is returned as 0, then no errors were detected. If *ier is non-zero, then refer to the error list in the error table for details.

c_dsgrid3s - Primary gridding function

c_dsgrid3s is the C single precision function that does an interpolation from 3D random data to an output grid. c_dsgrid3s is called after all of the desired values for control parameters have been set using the function c_dsseti, c_dssetr, c_dssetc.

Function prototype:

```
float    *c_dsgrid3s(int, float [], float [], float [], float [],
                     int, int, int, float [], float [], float [], int *);
```

Return value:

c_dsgrid3s returns a pointer to a linear array of data that is the interpolated grid stored in row-major order. That is, if *out* is declared as

```
  float *out;
```

and we set:

```
  out = c_dsgrid3s(n, x, y, z, u, nx, ny, nz, xo, yo, zo, &ier);
```

then out[nz*ny*i + nz*j + k] is the interpolated function value at coordinate point (xo[i],yo[j],zo[k]) for 0 <= i < nx, 0 <= j < ny, and 0 <= k < nz. The space for *out* is allocated internal to *c_dsgrid3s* and is nx*ny*nz floats in size.

Argument description:

```
-------------------------------------------------
                  Argument | Type       |  Size
-------------------------------------------------
float *c_dsgrid3s (npnts,  | int        |
                   x,      | float []   | npnts
                   y,      | float []   | npnts
                   z,      | float []   | npnts
                   u,      | float []   | npnts
                   numxout,| int        |
                   numyout,| int        |
                   numzout,| int        |
                   xo,     | float []   | numxout
                   yo,     | float []   | numyout
                   zo,     | float []   | numzout
                   ier     | int *      |
                   );
-------------------------------------------------
```

npnts
: The number of input data points.

x
: An array of size npnts containing the X coordinates of the input data points.

y
: An array of size npnts containing the Y coordinates of the input data points.

z
: An array of size npnts containing the Z coordinates of the input data points.

u
: An array of size npnts containing the functional values of the input data points. That is, z[n] is the value of the input function at coordinate (x[n],y[n],z[n]), for 0 <= n < npnts.

numxout
: The number of X values in the output grid.

numyout
: The number of Y values in the output grid.

numzout
: The number of Z values in the output grid.

xo
: An array of size numxout containing the X coordinates of the output data grid. The values in xo must be increasing, but need not be equally spaced.

yo
: An array of size numyout containing the Y coordinates of the output data grid. The values in yo must be increasing, but need not be equally spaced.

zo
: An array of size numyout containing the Z coordinates of the output data grid. The values in zo must be increasing, but need not be equally spaced.

ier
: An error return value. If *ier is returned as 0, then no errors were detected. If *ier is non-zero, then refer to the error list in the error table for details.

c_dsseti - Set int valued parameters

---

c_dsseti is used to set values for any of the control parameters that take int values. The values set by c_dsseti remain in effect until changed by subsequent calls to c_dsseti.

Function prototype:

```
  void c_dsseti(char *, int);
```

Argument description:

```
-------------------------------------------
              Argument | Type      |  Size
-------------------------------------------
void c_dsseti (pnam,   | char *    |
               ival);  | int       |
-------------------------------------------
```

pnam
: The name of the control parameter to be assigned an int value.

ival
: The value to be assigned to the control parameter whose name is pointed to by pnam.

c_dsgeti - Retrieve an int valued parameter

---

c_dsgeti is a called to obtain current values for any of the int valued control parameters.

Function prototype:

```
void c_dsgeti(char *, int);
```

Argument description:

```
-------------------------------------------
              Argument | Type      |  Size
-------------------------------------------
void c_dsgeti  (pnam,  | char *    |
                ival); | int   *   |
-------------------------------------------
```

pnam
: The name of the control parameter whose value is to be retrieved.

ival
: *ival will be the value currently assigned to the control parameter whose name is pointed to by pnam.

c_dssetr - Set float valued parameters

---

c_dssetr is used to set values for any of the control parameters that take float values. The values set by c_dssetr remain in effect until changed by subsequent calls to c_dssetr.

Function prototype:

```
void c_dssetr(char *, float);
```

Argument description:

```
-------------------------------------------
              Argument | Type      |  Size
-------------------------------------------
void c_dssetr  (pnam,  | char *    |
                fval); | float     |
-------------------------------------------
```

pnam
: The name of the control parameter to be assigned a float value.

fval
: The value to be assigned to the control parameter whose name is pointed to by pnam.

c_dsgetr - Retrieve a float valued parameter

---

c_dsgetr is a called to obtain current values for any of the float valued control parameters.

Function prototype:

```
void c_dsgetr(char *, float *);
```

Argument description:

```
-------------------------------------------
              Argument | Type     |  Size
-------------------------------------------
void c_dsgetr (pnam,   | char *   |
               fval);  | float *  |
-------------------------------------------
```

pnam
: The name of the control parameter whose value is to be retrieved.

fval
: *fval will be the value currently assigned to the control parameter whose name is pointed to by pnam.

c_dssetrd - Set double precision valued parameters

---

c_dssetrd is used to set values for any of the control parameters that take double precision values. The values set by c_dssetrd remain in effect until changed by subsequent calls to c_dssetrd.

Function prototype:

```
void c_dssetrd(char *, double);
```

Argument description:

```
-------------------------------------------
              Argument | Type     |  Size
-------------------------------------------
void c_dssetrd (pnam,  | char *   |
                dval); | double   |
-------------------------------------------
```

pnam
: The name of the control parameter to be assigned a double precision value.

dval
: The value to be assigned to the control parameter whose name is pointed to by pnam.

c_dsgetrd - Retrieve a double precision valued parameter

---

c_dsgetrd is a called to obtain current values for any of the double precision valued control parameters.

Function prototype:

```
void c_dsgetrd(char *, double *);
```

Argument description:

```
--------------------------------------------
              Argument | Type       |  Size
--------------------------------------------
void c_dsgetrd (pnam,  | char *     |
                dval); | double *   |
--------------------------------------------
```

pnam
: The name of the control parameter whose value is to be retrieved.

dval
: *dval will be the value currently assigned to the control parameter whose name is pointed to by pnam.

c_dssetc - Set char valued parameters

---

c_dssetc is used to set values for any of the control parameters that take string values. The values set by c_dssetc remain in effect until changed by subsequent calls to c_dssetc.

Function prototype:

```
void c_dssetc(char *, char *);
```

Argument description:

```
-------------------------------------------
              Argument | Type      |  Size
-------------------------------------------
void c_dssetc (pnam,   | char *    |
               cval);  | char *    |
-------------------------------------------
```

pnam
: The name of the control parameter to be assigned a char value.

cval
: The value to be assigned to the control parameter whose name is pointed to by pnam.

c_dsgetc - Retrieve a char valued parameter

---

c_dsgetc is a called to obtain current values for any of the string valued control parameters.

Function prototype:

```
void c_dsgetc(char *, char *);
```

Argument description:

```
-------------------------------------------
              Argument | Type      |  Size
-------------------------------------------
```

```
void c_dsgetc (pnam,     | char *  |
               cval);    | char *  |
-------------------------------------------
```

pnam

The name of the control parameter whose value is to be retrieved.

cval

cval will point to a string containing the returned value. The user is required to reserve enough space to store the string.

c_dspnt2s - Interpolate at specified points

This function is called to interpolate single precision 2D data at specified points.

Function prototype:

```
   void c_dspnt2s(int, float [], float [], float [],
                  int, float [], float [], float *, int *);
```

Argument description:

```
-----------------------------------------------
               Argument | Type       |  Size
-----------------------------------------------
void c_dspnt2s (n,      | int        |
                x,      | float []   | n
                y,      | float []   | n
                z,      | float []   | n
                m,      | int        |
                xo,     | float []   | m
                yo,     | float []   | m
                zo,     | float *    |
                ier     | int *      |
                );      |            |
-----------------------------------------------
```

n

The number of input data points.

x

An array containing the X coordinates of the input data points.

y

An array containing the Y coordinates of the input data points.

z

An array containing the functional values of the input data points. That is, z[l] is the value of the input function at coordinate (x[l],y[l]), for l=1,n.

m

The number of output data points (this may be "1").

xo

An array of dimension m containing the X coordinates of the output data. The values in xo may be in any order.

yo

An array of dimension m containing the Y coordinates of the output data. The values in yo may be in any order.

zo

A pointer to a float. Space for m floats is allocated for zo by dspnt2s. zo[i] is the interpolated value at point (xo[i],yo[i]) for i=1,m.

ier

An error return value. If *ier is returned as 0, then no errors were detected. If *ier is non-zero, then refer to the error list in the error table for details.

c_dspnt3s - Interpolate at specified points

This function is called to interpolate single precision 3D data at specified points.

Function prototype:

```
   void c_dspnt3s(int, float [], float [], float [], float[]
                  int, float [], float [], float [], float *, int *);
```

Argument description:

```
----------------------------------------------
               Argument | Type       |  Size
----------------------------------------------
void c_dspnt3s (n,       | int        |
                x,       | float []   |  n
                y,       | float []   |  n
                z,       | float []   |  n
                u,       | float []   |  n
                m,       | int        |
                xo,      | float []   |  m
                yo,      | float []   |  m
                zo,      | float []   |  m
                uo,      | float *    |
                ier      | int *      |
                );       |            |
----------------------------------------------
```

n

The number of input data points.

x

An array containing the X coordinates of the input data points.

y

An array containing the Y coordinates of the input data points.

z

An array containing the Z coordinates of the input data points.

u

An array containing the functional values of the input data points. That is, u[l] is the value of the input function at coordinate (x[l],y[l],z[l]), for l=1,n.

m

The number of output data points (this may be "1").

xo

An array of dimension m containing the X coordinates of the output data. The values in xo may be in any order.

yo

An array of dimension m containing the Y coordinates of the output data. The values in yo may be in any order.

zo

An array of dimension m containing the Z coordinates of the output data. The values in yo may be in any order.

uo

A pointer to a float. Space for m floats is allocated for uo by dspnt3s. zo[i] is the interpolated value at point (xo[i],yo[i],zo[i]) for i=1,m.

ier

An error return value. If *ier is returned as 0, then no errors were detected. If *ier is non-zero, then refer to the error list in the error table for details.

c_dsgrid2d - Primary 2D gridding function

---

c_dsgrid2d is the C double precision function that does an interpolation from 2D random data to an output grid. c_dsgrid2d is called after all of the desired values for control parameters have been set using the function c_dsseti, c_dssetrd, c_dssetc.

Function prototype:

```
double *c_dsgrid2d(int, double [], double [], double [],
                   int, int, double [], double [], int *);
```

Return value:

c_dsgrid2d returns a pointer to a linear array of data that is the interpolated grid stored in row-major order. That is, if *out* is declared as

```
double *out;
```

and we set:

```
out = c_dsgrid2d(npnts, x, y, z, numxout, numyout, xo, yo, &ier);
```

then out[i*numyout+j] is the interpolated function value at coordinate point (xo[i],yo[j]) for 0 <= i < numxout and 0 <= j < numyout. The space for *out* is allocated internal to *c_dsgrid2d* and is numxout*numyout doubles in size.

Argument description:

```
-------------------------------------------------
                   Argument | Type        |  Size
-------------------------------------------------
double *c_dsgrid2d (npnts,  | int         |
                    x,      | double []   | npnts
                    y,      | double []   | npnts
                    z,      | double []   | npnts
```

```
                    numxout,| int          |
                    numyout,| int          |
                    xi,     | double []    | numxout
                    yi,     | double []    | numyout
                    ier     | int *        |
                    );      |              |
-------------------------------------------------
```

npnts
: The number of input data points.

x
: An array of size npnts containing the X coordinates of the input data points.

y
: An array of size npnts containing the Y coordinates of the input data points.

z
: An array of size npnts containing the functional values of the input data points. That is, z[n] is the value of the input function at coordinate (x[n],y[n]), for 0 <= n < npnts.

numxout
: The number of X values in the output grid.

numyout
: The number of Y values in the output grid.

xi
: An array of size numxout containing the X coordinates of the output data grid. The values in xi must be increasing, but need not be equally spaced.

yi
: An array of size numyout containing the Y coordinates of the output data grid. The values in yi must be increasing, but need not be equally spaced.

ier
: An error return value. If *ier is returned as 0, then no errors were detected. If *ier is non-zero, then refer to the error list in the error table for details.

c_dsgrid3d - Primary 3D gridding function

---

c_dsgrid3d is the C double precision function that does an interpolation from 3D random data to an output grid. c_dsgrid3d is called after all of the desired values for control parameters have been set using the function c_dsseti, c_dssetrd, c_dssetc.

Function prototype:

```
double   *c_dsgrid3d(int, double [], double [], double [], double [],
                     int, int, int, double [], double [], double [], int *);
```

Return value:

c_dsgrid3d returns a pointer to a linear array of data that is the interpolated grid stored in row-major order. That is, if *out* is declared as

```
  double *out;
```

and we set:

```
out = c_dsgrid3d(n, x, y, z, u, nx, ny, nz, xo, yo, zo, &ier);
```

then out[nz*ny*i + nz*j + k] is the interpolated function value at coordinate point (xo[i],yo[j],zo[k]) for 0 <= i < nx, 0 <= j < ny, and 0 <= k < nz. The space for *out* is allocated internal to *c_dsgrid3d* and is nx*ny*nz doubles in size.

Argument description:

```
--------------------------------------------------
                    Argument   | Type        |  Size
--------------------------------------------------
double *c_dsgrid3d (npnts,     | int         |
                    x,         | double []   | npnts
                    y,         | double []   | npnts
                    z,         | double []   | npnts
                    u,         | double []   | npnts
                    numxout,   | int         |
                    numyout,   | int         |
                    numzout,   | int         |
                    xo,        | double []   | numxout
                    yo,        | double []   | numyout
                    zo,        | double []   | numzout
                    ier        | int *       |
                    );         |             |
--------------------------------------------------
```

npnts
: The number of input data points.

x
: An array of size npnts containing the X coordinates of the input data points.

y
: An array of size npnts containing the Y coordinates of the input data points.

z
: An array of size npnts containing the Z coordinates of the input data points.

u
: An array of size npnts containing the functional values of the input data points. That is, z[n] is the value of the input function at coordinate (x[n],y[n],z[n]), for 0 <= n < npnts.

numxout
: The number of X values in the output grid.

numyout
: The number of Y values in the output grid.

numzout
: The number of Z values in the output grid.

xo
: An array of size numxout containing the X coordinates of the output data grid. The values in xo must be increasing, but need not be equally spaced.

yo
: An array of size numyout containing the Y coordinates of the output data grid. The values in yo must be increasing, but need not be equally spaced.

zo

An array of size numzout containing the Z coordinates of the output data grid. The values in zo must be increasing, but need not be equally spaced.

ier

An error return value. If *ier is returned as 0, then no errors were detected. If *ier is non-zero, then refer to the error list in the error table for details.

c_dspnt2d - Interpolate at specified points

This function is called to interpolate double precision 2D data at specified points.

Function prototype:

```
   void c_dspnt2d(int, double [], double [], double [],
                  int, double [], double [], double *, int *);
```

Argument description:

```
-----------------------------------------------
               Argument |  Type       |  Size
-----------------------------------------------
void c_dspnt2d (n,      |  int        |
                x,      |  double []  |  n
                y,      |  double []  |  n
                z,      |  double []  |  n
                m,      |  int        |
                xo,     |  double []  |  m
                yo,     |  double []  |  m
                zo,     |  double *   |
                ier     |  int *      |
                );      |             |
-----------------------------------------------
```

n

The number of input data points.

x

An array containing the X coordinates of the input data points.

y

An array containing the Y coordinates of the input data points.

z

An array containing the functional values of the input data points. That is, z[l] is the value of the input function at coordinate (x[l],y[l]), for l=1,n.

m

The number of output data points (this may be "1").

xo

An array of dimension m containing the X coordinates of the output data. The values in xo may be in any order.

yo

An array of dimension m containing the Y coordinates of the output data. The values in yo may be in any order.

zo

A pointer to a float. Space for m floats is allocated for zo by dspnt2s. zo[i] is the interpolated value at point (xo[i],yo[i]) for i=1,m.

ier

An error return value. If *ier is returned as 0, then no errors were detected. If *ier is non-zero, then refer to the error list in the error table for details.

c_dspnt3d - Interpolate at specified points

This function is called to interpolate double precision 3D data at specified points.

Function prototype:

```
   void c_dspnt3d(int, double [], double [], double [], double[]
                  int, double [], double [], double [], double *, int *);
```

Argument description:

```
-----------------------------------------------
               Argument | Type        |  Size
-----------------------------------------------
void c_dspnt3d (n,      | int         |
                x,      | double []   | n
                y,      | double []   | n
                z,      | double []   | n
                u,      | double []   | n
                m,      | int         |
                xo,     | double []   | m
                yo,     | double []   | m
                zo,     | double []   | m
                uo,     | double *    |
                ier     | int *       |
                );      |             |
-----------------------------------------------
```

n

The number of input data points.

x

An array containing the X coordinates of the input data points.

y

An array containing the Y coordinates of the input data points.

z

An array containing the Z coordinates of the input data points.

u

An array containing the functional values of the input data points. That is, u[l] is the value of the input function at coordinate (x[l],y[l],z[l]), for l=1,n.

m

The number of output data points (this may be "1").

xo

An array of dimension m containing the X coordinates of the output data. The values in xo may be in any order.

yo

An array of dimension m containing the Y coordinates of the output data. The values in yo may be in any order.

zo

An array of dimension m containing the Z coordinates of the output data. The values in zo may be in any order.

uo

A pointer to a float. Space for m floats is allocated for uo by dspnt3s. uo[i] is the interpolated value at point (xo[i],yo[i],zo[i]) for i=1,m.

ier

An error return value. If *ier is returned as 0, then no errors were detected. If *ier is non-zero, then refer to the error list in the error table for details.

Dsgrid timing table

The following execution times were obtained on a Sun sparc Ultra-1 running SunOS 5.5.1 using version SC4.0 of the C compiler:

For interpolating on a grid:

Number of input points	Number of output points	Times for interpolation (in secs.)	Time per point (secs.)
50	500	0.03	.00006
50	1000	0.06	.00006
50	2000	0.12	.00006
50	4000	0.24	.00006
50	8000	0.48	.00006
100	500	0.07	.00012
100	1000	0.12	.00012
100	2000	0.25	.00012
100	4000	0.48	.00012
100	8000	0.96	.00012
200	500	0.13	.00025
200	1000	0.25	.00025
200	2000	0.49	.00025
200	4000	0.96	.00025
200	8000	1.92	.00025
400	500	0.24	.00050
400	1000	0.50	.00050
400	2000	0.98	.00050
400	4000	1.92	.00050
400	8000	3.84	.00050
800	500	0.49	.00100
800	1000	0.99	.00100
800	2000	1.96	.00100

800	4000	3.84	.00100
800	8000	7.66	.00100
1600	500	0.98	.00200
1600	1000	2.05	.00200
1600	2000	4.05	.00200
1600	4000	7.88	.00200
1600	8000	15.72	.00200

For interpolating in single point mode:

Number of input points	Time per point for interpolation (in secs.)
50	0.00011
100	0.00019
200	0.00035
400	0.00067
800	0.00134
1600	0.00265

Dsgrid Examples

Overview

This module contains source codes for all of the examples referenced by the other modules in the documentation. If you are using NCAR Graphics, all of the examples are available using the example tool *ncargex*. Links to all of the example plots are provided. Most of the examples use Dsgrid to interpolate in two dimensions, since it is much easier to illustrate the principles with such examples. The primary use of Dsgrid, however, is intended to be for 3D interpolation, since in most cases the Natgrid package is superior for 2D interpolations.

The following list summarizes the examples:

- Example 1 -- a simple 3D interpolation
- Example 2 -- modifying the exponent applied to the distances (2D examples)
- Example 3 -- smoothness (2D example)
- Example 4 -- using the shadowing feature (2D example)
- Example 5 -- extrapolation (2D example)
- Example 6 -- single point mode

The plotting subroutines used by the Fortran examples are TDEZ2D and TDEZ3D from the Tdapck package of NCAR Graphics. TDEZ2D is a simplified procedure for drawing a surface and TDEZ3D is a simplified procedure for drawing an isosurface. The C examples use the C functions, c_tdez2d.c and c_tdez3d.c from Tdpack - these are equivalent to the Fortran procedures. If you are not using NCAR Graphics and want to draw the plots, you will have to substitute appropriate calls from the graphics package you are using.

Creating your own executable

If you have access to NCAR Graphics, you can create executables for all of the examples in one of two ways. The easiest is to use the ncargex utility, which is documented with a man page. The other way is to use either *ncargf77* (for Fortran) or *ncargcc* (for C) to compile and execute the source codes.

If you are using NCAR Graphics and the C interface to Dsgrid and you are compiling using *ncargcc*, then function prototypes for all Dsgrid C functions are contained in a C header file. This can be accessed by inserting the line:

```
#include <ncarg/ngmath.h>
```

into your code.

If you do not have access to NCAR Graphics, replace the calls for drawing the plots with calls to the appropriate procedures at your site.

Example 1 -- a simple 3D interpolation

- Fortran single precision
- Fortran double precision
- C single precision
- C double precision
- Sample plot

Example 2 -- varying the distance exponent

- Fortran single precision
- C single precision
- Sample plot for exponent of 0.5
- Sample plot for exponent of 1.0
- Sample plot for exponent of 5.0

Example 3 -- smoothness

- Fortran single precision
- C single precision
- Sample plot of interpolated analytic function

Example 4 -- using the shadowing feature

- Fortran single precision
- C single precision
- Shadowing feature turned off
- Shadowing feature turned on

Example 5 -- extrapolation

- Fortran single precision
- C single precision
- Sample plot for extrapolation

Example 6 -- single point mode

- Fortran single precision
- C single precision
- Sample plot for single point mode

Example 1 - a simple 3D interpolation (Fortran single precision)

```
      PROGRAM DSEX01
C
      PARAMETER (NUM=1000, NX=21, NY=21, NZ=21)
C
C Define error file, Fortran unit number, and workstation type,
C and workstation ID.
C
      PARAMETER (IERRF=6, LUNIT=2, IWTYPE=1, IWKID=1)
C
      REAL XI(NUM), YI(NUM), ZI(NUM), U(NUM)
      REAL XO(NX),  YO(NY),  ZO(NZ),  OUTPUT(NX,NY,NZ)
      INTEGER I, J, K, IER
      DATA XMIN,YMIN,ZMIN,XMAX,YMAX,ZMAX / -2., -2., -2., 2., 2., 2./
C
      DO 10 I=1,NUM
        XI(I) = XMIN+(XMAX-XMIN)*DSRND1()
        YI(I) = YMIN+(YMAX-YMIN)*DSRND1()
        ZI(I) = ZMIN+(ZMAX-ZMIN)*DSRND1()
         U(I) = XI(I)**2 + YI(I)**2 + ZI(I)**2
   10 CONTINUE
C
C  Create the output grid.
C
      DO 102 I=1,NX
        XO(I) = XMIN+(REAL(I-1)/REAL(NX-1))*(XMAX-XMIN)
  102 CONTINUE
C
```

```
      DO 103 J =1,NY
        YO(J)= YMIN+(REAL(J-1)/REAL(NY-1))*(YMAX-YMIN)
  103 CONTINUE
C
      DO 104 K=1,NZ
        ZO(K) = ZMIN+(REAL(K-1)/REAL(NZ-1))*(ZMAX-ZMIN)
  104 CONTINUE
C
C  Interpolate.
C
      CALL DSGRID3S(NUM,XI,YI,ZI,U,NX,NY,NZ,XO,YO,ZO,OUTPUT,IER)
      IF (IER .NE. 0) THEN
        WRITE(6,520) IER
  520   FORMAT(' Error ',I3,' returned from DSGRID3S')
        STOP
      ENDIF
C
C  Plot an isosurface.
C
C
C Open GKS and define the foreground and background color.
C
      CALL GOPKS (IERRF, ISZDM)
      CALL GOPWK (IWKID, LUNIT, IWTYPE)
      CALL GACWK (IWKID)
C
      CALL TDEZ3D(NX, NY, NZ, XO, YO, ZO, OUTPUT, 3.0,
     +             2., -35., 65., 6)
      CALL FRAME()
C
C Close GKS.
C
      CALL GDAWK (IWKID)
      CALL GCLWK (IWKID)
      CALL GCLKS
C
      STOP
      END
      REAL FUNCTION DSRND1()
C
      DATA ISEED/1/
      SAVE ISEED
C
      ISEED = ISEED*1103515245 + 12345
      IT = IAND(ISHIFT(ISEED,-16),32767)
C
      DSRND1 = REAL(IT)/32767.
C
      RETURN
      END
```

Example 1 - a simple 3D interpolation (Fortran double precision)

```
      PROGRAM DSEX01
C
      PARAMETER (NUM=1000, NX=21, NY=21, NZ=21)
C
C Define error file, Fortran unit number, and workstation type,
C and workstation ID.
C
      PARAMETER (IERRF=6, LUNIT=2, IWTYPE=1, IWKID=1)
C
      DOUBLE PRECISION XI(NUM), YI(NUM), ZI(NUM), U(NUM)
      DOUBLE PRECISION XO(NX),  YO(NY),  ZO(NZ),  OUTPUT(NX,NY,NZ)
      DOUBLE PRECISION XMIN,    YMIN,    ZMIN
      DOUBLE PRECISION XMAX,    YMAX,    ZMAX
      REAL             XP(NX),  YP(NY),  ZP(NZ),  OUTP(NX,NY,NZ)
      INTEGER I, J, K, IER
C
      DATA XMIN,YMIN,ZMIN/ -2.D0, -2.D0, -2.D0/
      DATA XMAX,YMAX,ZMAX/  2.D0,  2.D0,  2.D0/
C
C  Create random data in three space and define a function.
C  To get this to work on your system, you may have to insert
C  the correct random number generator for your compiler.
C
      DO 10 I=1,NUM
        XI(I) = XMIN+(XMAX-XMIN)*DBLE(DSRND1D())
        YI(I) = YMIN+(YMAX-YMIN)*DBLE(DSRND1D())
        ZI(I) = ZMIN+(ZMAX-ZMIN)*DBLE(DSRND1D())
         U(I) = XI(I)**2 + YI(I)**2 + ZI(I)**2
   10 CONTINUE
C
C  Create the output grid.
C
      DO 102 I=1,NX
        XO(I) = XMIN+(DBLE(I-1)/DBLE(NX-1))*(XMAX-XMIN)
  102 CONTINUE
C
      DO 103 J =1,NY
        YO(J)= YMIN+(DBLE(J-1)/DBLE(NY-1))*(YMAX-YMIN)
  103 CONTINUE
C
      DO 104 K=1,NZ
        ZO(K) = ZMIN+(DBLE(K-1)/DBLE(NZ-1))*(ZMAX-ZMIN)
  104 CONTINUE
C
C  Interpolate.
C
      CALL DSGRID3D(NUM,XI,YI,ZI,U,NX,NY,NZ,XO,YO,ZO,OUTPUT,IER)
      IF (IER .NE. 0) THEN
        WRITE(6,520) IER
  520   FORMAT(' Error ',I3,' returned from DSGRID3D')
        STOP
      ENDIF
C
C  Pass single precision arrays for plotting an isosurface.
C
      DO 105 I=1,NX
        XP(I) = XO(I)
        DO 106 J=1,NY
          YP(J) = YO(J)
          DO 107 K=1,NZ
```

```
            ZP(K) = ZO(K)
            OUTP(I,J,K) = OUTPUT(I,J,K)
  107     CONTINUE
  106   CONTINUE
  105 CONTINUE
C
C  Draw plot.
C
      CALL GOPKS (IERRF, ISZDM)
      CALL GOPWK (IWKID, LUNIT, IWTYPE)
      CALL GACWK (IWKID)
      CALL TDEZ3D(NX, NY, NZ, XP, YP, ZP, OUTP, 3.0, 0., 0., 0., 6)
      CALL GDAWK (IWKID)
      CALL GCLWK (IWKID)
      CALL GCLKS
C
      STOP
      END
      REAL FUNCTION DSRND1D()
C
      DATA ISEED/1/
      SAVE ISEED
C
      ISEED = ISEED*1103515245 + 12345
      IT = IAND(ISHIFT(ISEED,-16),32767)
C
      DSRND1D = REAL(IT)/32767.
C
      RETURN
      END
```

Example 1 - a simple 3D interpolation (C single precision)

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <ncarg/ncargC.h>
#include <ncarg/gks.h>
#include <ncarg/ngmath.h>

#define NUM 1000
#define NX    21
#define NY    21
#define NZ    21
#define IWTYPE 1
#define WKID   1

main()
{
  int  i, j, k, ier;
  float xi[NUM], yi[NUM], zi[NUM], u[NUM];
  float xo[NX], yo[NY], zo[NZ], *output, outr[NZ][NY][NX];
```

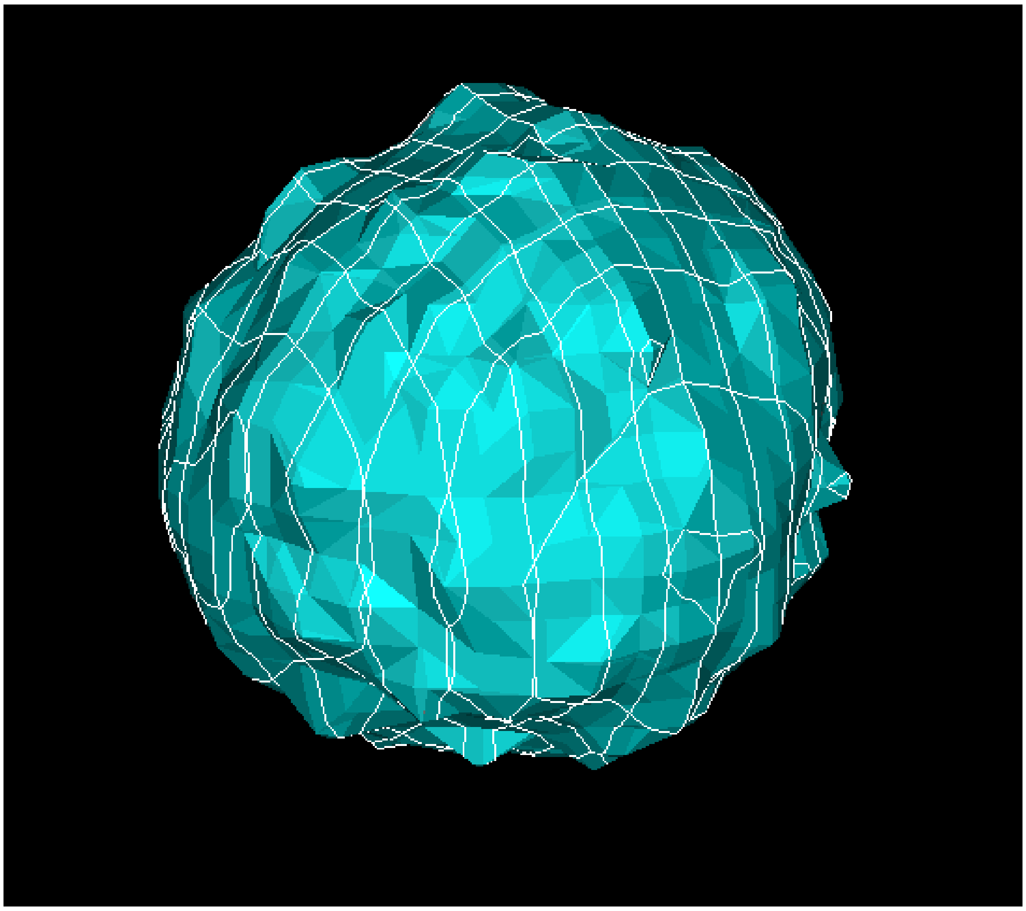
```
  float xmin = -2.0, ymin = -2.0, zmin = -2.0;
  float xmax =  2.0, ymax =  2.0, zmax =  2.0;

/*
 *  Create random data in three space and define a function.
 */
  for (i = 0; i < NUM; i++) {
    xi[i] = xmin+(xmax-xmin)*((float) rand() / (float) RAND_MAX);
    yi[i] = ymin+(ymax-ymin)*((float) rand() / (float) RAND_MAX);
    zi[i] = zmin+(zmax-zmin)*((float) rand() / (float) RAND_MAX);
     u[i] = xi[i]*xi[i] + yi[i]*yi[i] + zi[i]*zi[i];
  }

/*
 *  Create the output grid.
 */
  for (i = 0; i < NX; i++) {
    xo[i] = xmin + ( (float) i / (float) (NX-1)) * (xmax-xmin);
  }
  for (j = 0; j < NY; j++) {
    yo[j] = ymin + ( (float) j / (float) (NY-1)) * (ymax-ymin);
  }
  for (k = 0; k < NZ; k++) {
    zo[k] = zmin + ( (float) k / (float) (NZ-1)) * (zmax-zmin);
  }

/*
 *  Interpolate.
 */
  output = c_dsgrid3s(NUM, xi, yi, zi, u, NX, NY, NZ, xo, yo, zo, &ier);
  if (ier != 0) {
    printf(" Error %d returned from c_dsgrid3s\n",ier);
    exit(1);
  }

/*
 *  Plot an isosurface.
 */
  gopen_gks ("stdout",0);
  gopen_ws (WKID, NULL, IWTYPE);
  gactivate_ws(WKID);

/*
 *  Reorder the array elements for c_tdez3d, since c_dsgrid3s
 *  returns an array in column dominate order.
 */
  for (i = 0; i < NX; i++) {
    for (j = 0; j < NY; j++) {
      for (k = 0; k < NZ; k++) {
        outr[k][j][i] = output[i*NY*NZ + j*NY + k];
      }
    }
  }
  c_tdez3d(NX, NY, NZ, xo, yo, zo, &outr[0][0][0], 3.0, 2., -35., 65., 6);
  c_frame();

  gdeactivate_ws(WKID);
  gclose_ws(WKID);
  gclose_gks();
}
```

Example 1 - a simple 3D interpolation (C double precision)

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <ncarg/ncargC.h>
#include <ncarg/gks.h>
#include <ncarg/ngmath.h>

#define NUM 1000
#define NX    21
#define NY    21
#define NZ    21
#define IWTYPE 1
#define WKID   1

main()
{
  int  i, j, k, ier;
  double xi[NUM], yi[NUM], zi[NUM], u[NUM];
  double xo[NX], yo[NY], zo[NZ], *output;
  float  xp[NX], yp[NY], zp[NZ], outp[NZ][NY][NX];
  double xmin = -2.0, ymin = -2.0, zmin = -2.0;
  double xmax =  2.0, ymax =  2.0, zmax =  2.0;

/*
 *  Create random data in three space and define a function.
 */
  for (i = 0; i < NUM; i++) {
    xi[i] = xmin+(xmax-xmin)*((double) rand() / (double) RAND_MAX);
    yi[i] = ymin+(ymax-ymin)*((double) rand() / (double) RAND_MAX);
    zi[i] = zmin+(zmax-zmin)*((double) rand() / (double) RAND_MAX);
     u[i] = xi[i]*xi[i] + yi[i]*yi[i] + zi[i]*zi[i];
  }

/*
 *  Create the output grid.
 */
  for (i = 0; i < NX; i++) {
    xo[i] = xmin + ( (double) i / (double) (NX-1)) * (xmax-xmin);
  }
  for (j = 0; j < NY; j++) {
    yo[j] = ymin + ( (double) j / (double) (NY-1)) * (ymax-ymin);
  }
  for (k = 0; k < NZ; k++) {
    zo[k] = zmin + ( (double) k / (double) (NZ-1)) * (zmax-zmin);
  }

/*
 *  Interpolate.
 */
  output = c_dsgrid3d(NUM, xi, yi, zi, u, NX, NY, NZ, xo, yo, zo, &ier);
```

```
  if (ier != 0) {
    printf(" Error %d returned from c_dsgrid3s\n",ier);
    exit(1);
  }

/*
 *  Plot an isosurface, converting to single precision arrays first.
 *  Also, rearrange the output array, since dsgrid3d returns its
 *  array in column dominate order.
 */
  for (i = 0; i < NX; i++) {
    xp[i] = xo[i];
    for (j = 0; j < NY; j++) {
      yp[i] = yo[i];
      for (k = 0; k < NZ; k++) {
        zp[i] = zo[i];
        outp[k][j][i] = output[i*NY*NZ + j*NY + k];
      }
    }
  }

  gopen_gks ("stdout",0);
  gopen_ws (WKID, NULL, IWTYPE);
  gactivate_ws(WKID);

  c_tdez3d(NX, NY, NZ, xp, yp, zp, &outp[0][0][0], 3.0, 2., -35., 65., 6);

  gdeactivate_ws(WKID);
  gclose_ws(WKID);
  gclose_gks();
}
```

Plot for example 1 - a simple 3D interpolation



Example 2 - varying the exponent of the distances (Fortran single precision)

```
      PROGRAM DSEX02
C
      PARAMETER (NUM=6, NX=61, NY=61, IWDIM=2*NX*NY)
C
C Define error file, Fortran unit number, and workstation type,
C and workstation ID.
C
      PARAMETER (IERRF=6, LUNIT=2, IWTYPE=1, IWKID=1)
C
      REAL XI(NUM), YI(NUM), ZI(NUM)
      REAL XO(NX), YO(NY), OUTPUT(NX,NY)
      REAL XINC, YINC
C
C  Input data points and values.
C
      DATA XI/0.00, 1.00, 0.00, 1.00, 0.40, 0.75/
      DATA YI/0.00, 0.00, 1.00, 1.00, 0.20, 0.65/
```

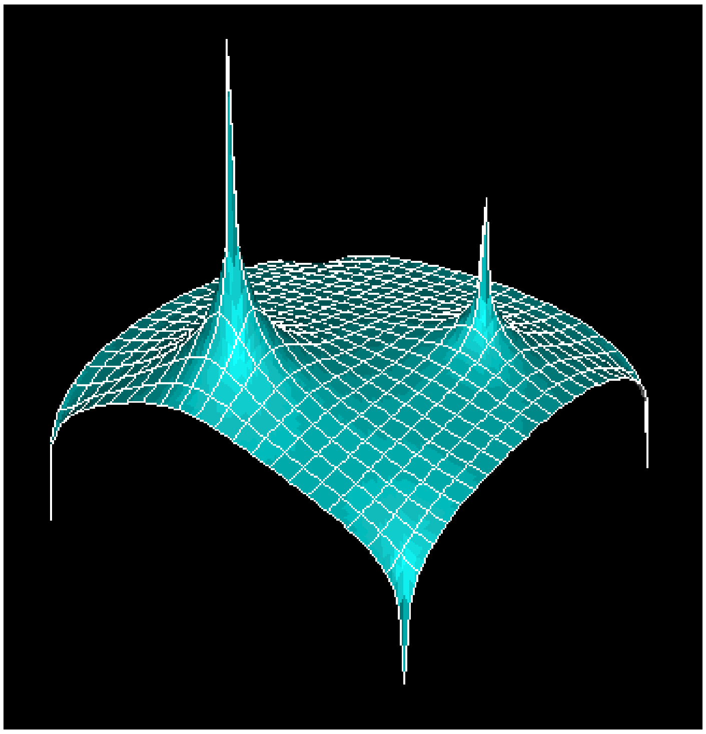
```
      DATA ZI/0.00, 0.00, 0.00, 0.00, 1.25, 0.80/
      DATA RHO, THETA, PHI/3.0, -45., 55./
C
C  Specify the output grid.
C
      XINC = 1./ REAL(NX-1)
      YINC = 1./ REAL(NY-1)
      DO 30 I=1,NX
        XO(I) = REAL((I-1)*XINC)
        DO 40 J=1,NY
          YO(J) = REAL((J-1)*YINC)
   40   CONTINUE
   30 CONTINUE
C
C  Exponent equals 0.5
C
      CALL DSSETR('EXP',0.5)
      CALL DSGRID2S(NUM, XI, YI, ZI, NX, NY, XO, YO, OUTPUT, IER)
      IF (IER .NE. 0) THEN
        WRITE(6,520) IER
  520   FORMAT(' Error ',I3,' returned from DSGRID2S')
        STOP
      ENDIF
C
      CALL GOPKS (IERRF, ISZDM)
      CALL GOPWK (IWKID, LUNIT, IWTYPE)
      CALL GACWK (IWKID)
      CALL TDEZ2D(NX, NY, XO, YO, OUTPUT, RHO, THETA, PHI, 6)
      CALL FRAME()
C
C  Exponent equals 1.0
C
      CALL DSSETR('EXP',1.0)
      CALL DSGRID2S(NUM, XI, YI, ZI, NX, NY, XO, YO, OUTPUT, IER)
      IF (IER .NE. 0) THEN
        WRITE(6,520) IER
        STOP
      ENDIF
      CALL TDEZ2D(NX, NY, XO, YO, OUTPUT, RHO, THETA, PHI, 6)
      CALL FRAME()
C
C  Exponent equals 10.0
C
      CALL DSSETR('EXP',5.0)
      CALL DSGRID2S(NUM, XI, YI, ZI, NX, NY, XO, YO, OUTPUT, IER)
      IF (IER .NE. 0) THEN
        WRITE(6,520) IER
        STOP
      ENDIF
C
      CALL TDEZ2D(NX, NY, XO, YO, OUTPUT, RHO, THETA, PHI, 6)
      CALL FRAME()
C
      CALL GDAWK (IWKID)
      CALL GCLWK (IWKID)
      CALL GCLKS
C
      STOP
      END
```

Example 2 - varying the exponent of the distances (C single precision)

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <ncarg/ncargC.h>
#include <ncarg/gks.h>
#include <ncarg/ngmath.h>

#define NUM    6
#define NX    61
#define NY    61
#define IWTYPE 1
#define WKID   1

main()
{
  int  i, j, k, ier;
  float xi[] = {0.00, 1.00, 0.00, 1.00, 0.40, 0.75};
  float yi[] = {0.00, 0.00, 1.00, 1.00, 0.20, 0.65};
  float zi[] = {0.00, 0.00, 0.00, 0.00, 1.25, 0.80};
  float xo[NX], yo[NY], xinc, yinc, *output, outr[NY][NX];
  float rho =  3., theta = -45., phi =  55.;

/*
 *  Create the output grid.
 */
  xinc = 1./ (float) (NX-1);
  yinc = 1./ (float) (NY-1);
  for (i = 0; i < NX; i++) {
    xo[i] = xinc * (float) i;
  }
  for (j = 0; j < NY; j++) {
    yo[j] = yinc * (float) j;
  }

/*
 *  Exponent equals 0.5
 */
  c_dssetr("exp", 0.5);
  output = c_dsgrid2s(NUM, xi, yi, zi, NX, NY, xo, yo, &ier);
  if (ier != 0) {
    printf(" Error %d returned from c_dsgrid2s\n",ier);
    exit(1);
  }
  gopen_gks ("stdout",0);
  gopen_ws (WKID, NULL, IWTYPE);
  gactivate_ws(WKID);
/*
 *  Reverse the array indices for plotting with tdez2d, since
 *  c_dsgrid returns its array in column dominate order.
 */
  for (i = 0; i < NX; i++) {
```

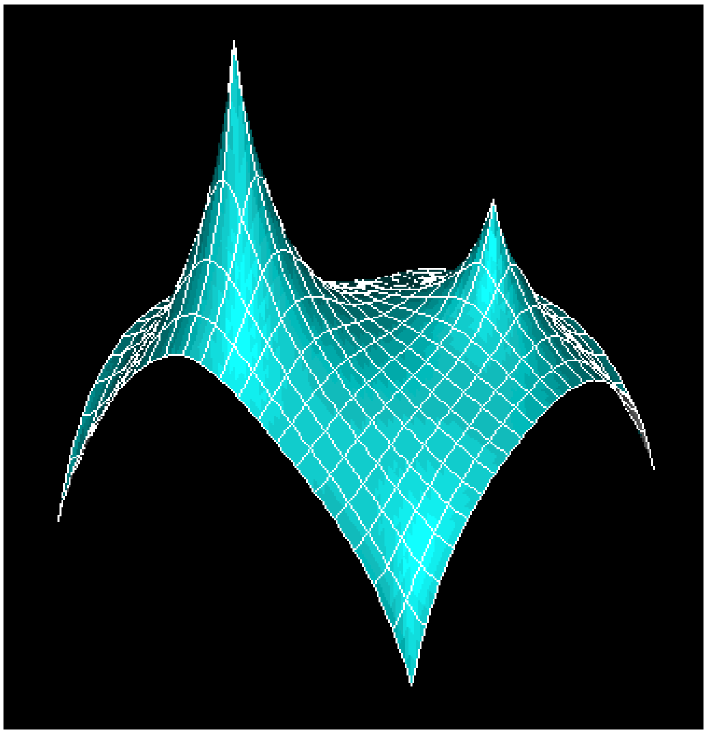
```
    for (j = 0; j < NY; j++) {
      outr[j][i] = output[i*NY+j];
    }
  }
  c_tdez2d(NX, NY, xo, yo, &outr[0][0], rho, theta, phi, 6);
  c_frame();

/*
 *  Exponent equals 1.0
 */
  c_dssetr("exp", 1.0);
  output = c_dsgrid2s(NUM, xi, yi, zi, NX, NY, xo, yo, &ier);
  if (ier != 0) {
    printf(" Error %d returned from c_dsgrid2s\n",ier);
    exit(1);
  }
/*
 *  Reverse the array indices for plotting with tdez2d, since
 *  c_dsgrid returns its array in column dominate order.
 */
  for (i = 0; i < NX; i++) {
    for (j = 0; j < NY; j++) {
      outr[j][i] = output[i*NY+j];
    }
  }
  c_tdez2d(NX, NY, xo, yo, &outr[0][0], rho, theta, phi, 6);
  c_frame();

/*
 *  Exponent equals 5.0
 */
  c_dssetr("exp", 5.0);
  output = c_dsgrid2s(NUM, xi, yi, zi, NX, NY, xo, yo, &ier);
  if (ier != 0) {
    printf(" Error %d returned from c_dsgrid2s\n",ier);
    exit(1);
  }
/*
 *  Reverse the array indices for plotting with tdez2d, since
 *  c_dsgrid returns its array in column dominate order.
 */
  for (i = 0; i < NX; i++) {
    for (j = 0; j < NY; j++) {
      outr[j][i] = output[i*NY+j];
    }
  }
  c_tdez2d(NX, NY, xo, yo, &outr[0][0], rho, theta, phi, 6);
  c_frame();

  gdeactivate_ws(WKID);
  gclose_ws(WKID);
  gclose_gks();
}
```

Plot for example 2 - distance exponent equals 0.5



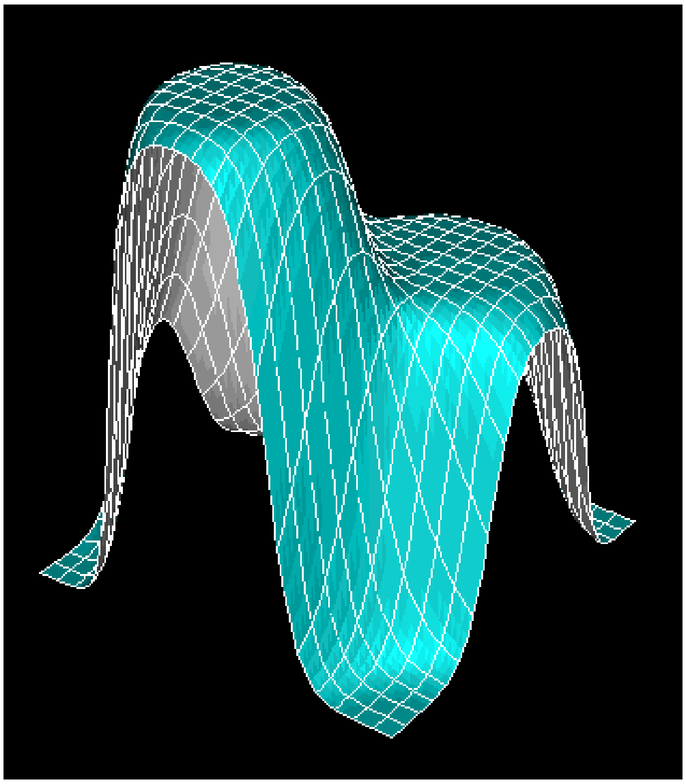
Notice the cusp-like regions around the input data points.

Plot for example 2 - distance exponent equals 1.0



Notice the cone-shaped regions around the input data points.

Plot for example 2 - distance exponent equals 5.0



Notice the flat regions around the input data points.

Example 3 - smoothness (Fortran single precision)

```
      PROGRAM DSEX03
C
      PARAMETER (NUM=171, NX=21, NY=21)
C
C Define error file, Fortran unit number, and workstation type,
C and workstation ID.
C
      PARAMETER (IERRF=6, LUNIT=2, IWTYPE=1, IWKID=1)
C
      REAL XI(NUM), YI(NUM), ZI(NUM)
      REAL XO(NX),  YO(NY),  OUTPUT(NX,NY)
      DATA XMININ,YMININ,XMAXIN,YMAXIN/ -0.2, -0.2, 1.2, 1.2/
      DATA XMINOT,YMINOT,XMAXOT,YMAXOT/  0.0,  0.0, 1.0, 1.0/
      DATA THETA, PHI, RHO/-54., 32., 3./
C
C  Create random data in two-space and define a function.
C  To get this to work on your system, you may have to insert
C  the correct random number generator for your compiler.
C
```

```
      DO 10 I=1,NUM
        XI(I) = XMININ+(XMAXIN-XMININ)*DSRND3()
        YI(I) = YMININ+(YMAXIN-YMININ)*DSRND3()
        ZI(I) = (XI(I)-0.25)**2 + (YI(I)-0.50)**2
   10 CONTINUE
C
C  Create the output grid.
C
      DO 102 I=1,NX
        XO(I) = XMINOT+(REAL(I-1)/REAL(NX-1))*(XMAXOT-XMINOT)
  102 CONTINUE
      DO 103 J =1,NY
        YO(J)= YMINOT+(REAL(J-1)/REAL(NY-1))*(YMAXOT-YMINOT)
  103 CONTINUE
C
C  Interpolate.
C
      CALL DSGRID2S(NUM, XI, YI, ZI, NX, NY, XO, YO, OUTPUT, IER)
      IF (IER .NE. 0) THEN
        WRITE(6,520) IER
  520   FORMAT(' Error ',I3,' returned from DSGRID2S')
        STOP
      ENDIF
C
C  Plot a surface.
C
      CALL GOPKS (IERRF, ISZDM)
      CALL GOPWK (IWKID, LUNIT, IWTYPE)
      CALL GACWK (IWKID)
      CALL TDEZ2D(NX, NY, XO, YO, OUTPUT, RHO, THETA, PHI, 6)
      CALL FRAME()
      CALL GDAWK (IWKID)
      CALL GCLWK (IWKID)
      CALL GCLKS
C
      STOP
      END
      REAL FUNCTION DSRND3()
C
      DATA ISEED/1/
      SAVE ISEED
C
      ISEED = ISEED*1103515245 + 12345
      IT = IAND(ISHIFT(ISEED,-16),32767)
C
      DSRND3 = REAL(IT)/32767.
C
      RETURN
      END
```

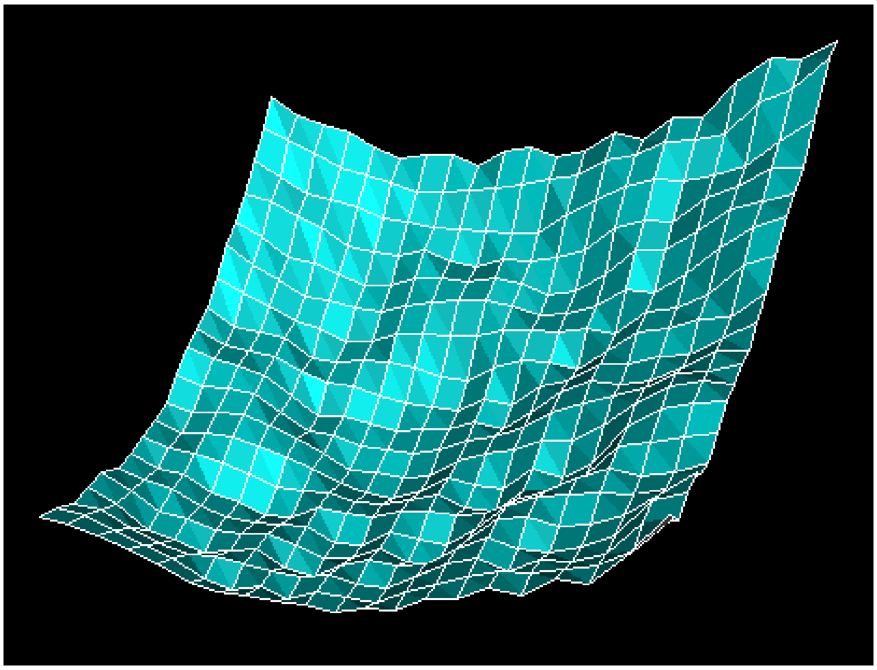
Example 3 - smoothness (C single precision)

```
#include <stdio.h>
#include <stdlib.h>
```

```
#include <math.h>
#include <ncarg/ncargC.h>
#include <ncarg/gks.h>
#include <ncarg/ngmath.h>

#define NUM  171
#define NX    21
#define NY    21
#define IWTYPE 1
#define WKID   1

main()
{
  int  i, j, k, ier;
  float xi[NUM], yi[NUM], zi[NUM];
  float xo[NX], yo[NY], *output, outr[NY][NX];
  float xminin = -0.2, yminin = -0.2, xmaxin = 1.2, ymaxin = 1.2;
  float xminot =  0.0, yminot =  0.0, xmaxot = 1.0, ymaxot = 1.0;
  float rho = 3., theta = -54., phi = 32.;

/*
 *  Create random data in three space and define a function.
 */
  for (i = 0; i < NUM; i++) {
    xi[i] = xminin+(xmaxin-xminin)*((float) rand() / (float) RAND_MAX);
    yi[i] = yminin+(ymaxin-yminin)*((float) rand() / (float) RAND_MAX);
    zi[i] = (xi[i]-0.25)*(xi[i]-0.25) + (yi[i]-0.50)*(yi[i]-0.50);
  }

/*
 *  Create the output grid.
 */
  for (i = 0; i < NX; i++) {
    xo[i] = xminot + ( (float) i / (float) (NX-1)) * (xmaxot-xminot);
  }
  for (j = 0; j < NY; j++) {
    yo[j] = yminot + ( (float) j / (float) (NY-1)) * (ymaxot-yminot);
  }

/*
 *  Interpolate.
 */
  output = c_dsgrid2s(NUM, xi, yi, zi, NX, NY, xo, yo, &ier);
  if (ier != 0) {
    printf(" Error %d returned from c_dsgrid2s\n",ier);
    exit(1);
  }
  gopen_gks ("stdout",0);
  gopen_ws (WKID, NULL, IWTYPE);
  gactivate_ws(WKID);

/*
 *  Reverse the array indices for plotting with tdez2d, since
 *  c_dsgrid returns its array in column dominate order.
 */
  for (i = 0; i < NX; i++) {
    for (j = 0; j < NY; j++) {
      outr[j][i] = output[i*NY+j];
    }
  }
  c_tdez2d(NX, NY, xo, yo, &outr[0][0], rho, theta, phi, 6);
```

```
  c_frame();

  gdeactivate_ws(WKID);
  gclose_ws(WKID);
  gclose_gks();
}
```

Plot for example 3 - smoothness (or lack thereof)



Example 4 - using the shadowing feature (Fortran single precision)

```
      PROGRAM DSEX04
C
      PARAMETER (NUM=16, NX=21, NY=21)
C
C Define error file, Fortran unit number, and workstation type,
C and workstation ID.
C
      PARAMETER (IERRF=6, LUNIT=2, IWTYPE=1, IWKID=1)
C
      REAL XI(NUM), YI(NUM), ZI(NUM)
      REAL XO(NX), YO(NY), OUTPUT(NX,NY)
```

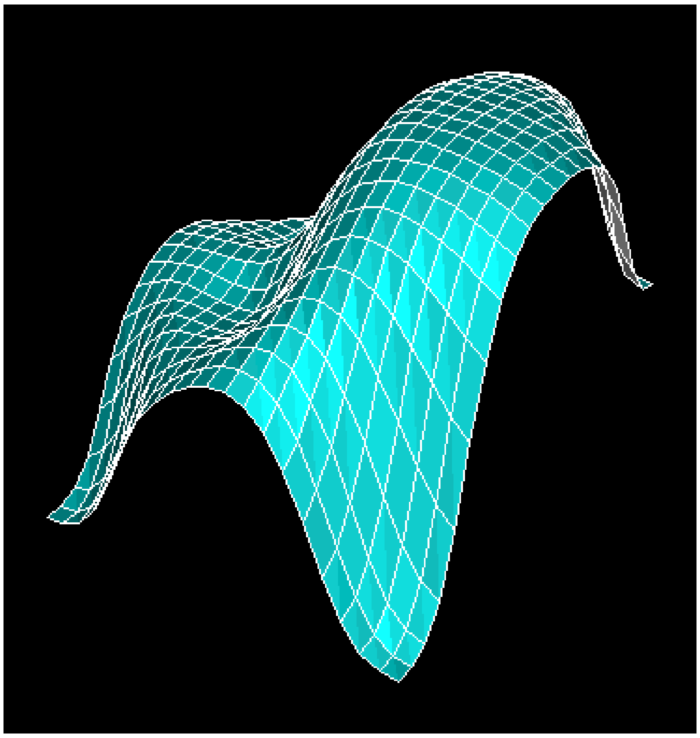
```
      REAL XINC, YINC
C
C  Input data points and values.
C
      DATA XI/0.00, 1.00, 0.00, 1.00, 0.30, 0.30, 0.30, 0.69,
     +        0.71, 0.71, 0.69, 0.70, 0.70, 0.70, 0.69, 0.71/
      DATA YI/0.00, 0.00, 1.00, 1.00, 0.70, 0.30, 0.70, 0.69,
     +        0.69, 0.71, 0.71, 0.70, 0.69, 0.71, 0.70, 0.70/
      DATA ZI/0.00, 0.00, 0.00, 0.50, 0.50, 0.50, 0.50, 1.00,
     +        1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00/
      DATA RHO, THETA, PHI/3.0, -45., 55./
C
C  Specify the output grid.
C
      XINC = 1./ REAL(NX-1)
      YINC = 1./ REAL(NY-1)
      DO 30 I=1,NX
        XO(I) = REAL((I-1)*XINC)
        DO 40 J=1,NY
          YO(J) = REAL((J-1)*YINC)
   40   CONTINUE
   30 CONTINUE
C
C  Set shadowing flag.
C
      CALL DSSETI('SHD',1)
      CALL DSGRID2S(NUM, XI, YI, ZI, NX, NY, XO, YO, OUTPUT, IER)
      IF (IER .NE. 0) THEN
        WRITE(6,520) IER
  520   FORMAT(' Error ',I3,' returned from DSGRID2S')
        STOP
      ENDIF
      CALL GOPKS (IERRF, ISZDM)
      CALL GOPWK (IWKID, LUNIT, IWTYPE)
      CALL GACWK (IWKID)
      CALL TDEZ2D(NX, NY, XO, YO, OUTPUT, RHO, THETA, PHI, 6)
      CALL FRAME()
      CALL GDAWK (IWKID)
      CALL GCLWK (IWKID)
      CALL GCLKS
C
      STOP
      END
```

Example 4 - using the shadowing feature (C single precision)

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <ncarg/ncargC.h>
#include <ncarg/gks.h>
#include <ncarg/ngmath.h>
```

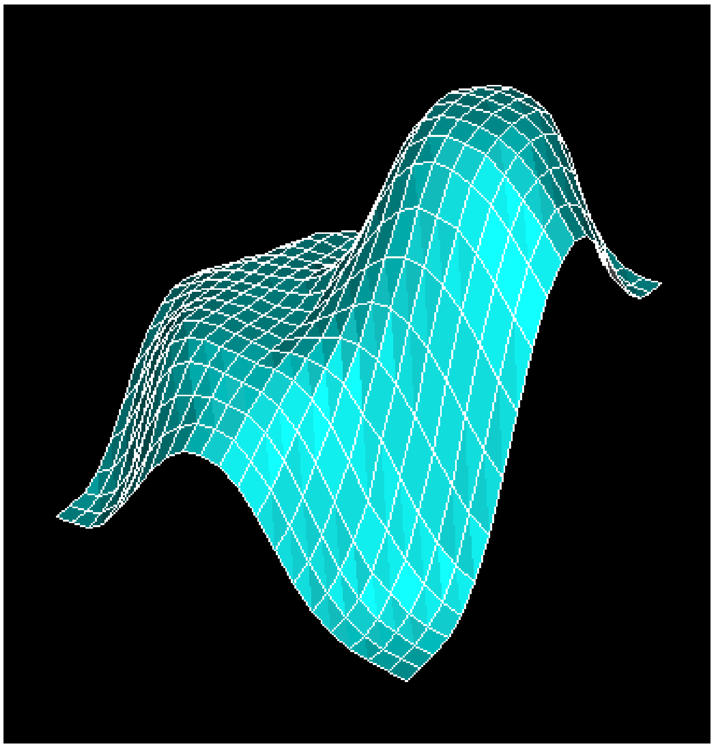
```
#define NX    21
#define NY    21
#define IWTYPE 1
#define WKID   1

main()
{
  int  i, j, k, ier;
  float xi[] = {0.00, 1.00, 0.00, 1.00, 0.30, 0.30, 0.30, 0.69,
                0.71, 0.71, 0.69, 0.70, 0.70, 0.70, 0.69, 0.71};
  float yi[] = {0.00, 0.00, 1.00, 1.00, 0.70, 0.30, 0.70, 0.69,
                0.71, 0.71, 0.69, 0.70, 0.70, 0.70, 0.69, 0.71};
  float zi[] = {0.00, 0.00, 0.00, 0.50, 0.50, 0.50, 0.50, 1.00,
                1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00};
  float xo[NX], yo[NY], xinc, yinc, *output, outr[NY][NX];
  float rho =  3., theta = -45., phi =  55.;

/*
 *  Create the output grid.
 */
  xinc = 1./ (float) (NX-1);
  yinc = 1./ (float) (NY-1);
  for (i = 0; i < NX; i++) {
    xo[i] = xinc * (float) i;
  }
  for (j = 0; j < NY; j++) {
    yo[j] = yinc * (float) j;
  }

/*
 *  Set the shadowing flag.
 */
  c_dsseti("shd", 1);
  output = c_dsgrid2s(sizeof(xi)/sizeof(xi[0]), xi, yi, zi,
                                             NX, NY, xo, yo, &ier);
  if (ier != 0) {
    printf(" Error %d returned from c_dsgrid2s\n",ier);
    exit(1);
  }
  gopen_gks ("stdout",0);
  gopen_ws (WKID, NULL, IWTYPE);
  gactivate_ws(WKID);

/*
 *  Reverse the array indices for plotting with tdez2d, since
 *  c_dsgrid returns its array in column dominate order.
 */
  for (i = 0; i < NX; i++) {
    for (j = 0; j < NY; j++) {
      outr[j][i] = output[i*NY+j];
    }
  }
  c_tdez2d(NX, NY, xo, yo, &outr[0][0], rho, theta, phi, 6);
  c_frame();

  gdeactivate_ws(WKID);
  gclose_ws(WKID);
  gclose_gks();
}
```

Plot for example 4 - shadowing feature off



Notice the dominant flat mound.

Plot for example 4 - shadowing feature on



Notice the sharply defined raised mound.

Example 5 - extrapolation (Fortran single precision)

```
      PROGRAM DSEX05
C
      PARAMETER (NUM=5000, NX=21, NY=21)
C
C Define error file, Fortran unit number, and workstation type,
C and workstation ID.
C
      PARAMETER (IERRF=6, LUNIT=2, IWTYPE=1, IWKID=1)
C
      REAL XI(NUM), YI(NUM), ZI(NUM)
      REAL XO(NX),  YO(NY),  OUTPUT(NX,NY)
      DATA XMININ,YMININ,XMAXIN,YMAXIN/  0.00,  0.00, 1.00, 1.00/
      DATA XMINOT,YMINOT,XMAXOT,YMAXOT/ -0.21, -0.21, 1.21, 1.21/
      DATA RHO, THETA, PHI/3., -70., 40./
C
C  Create random data in two-space and define a function.
C  To get this to work on your system, you may have to insert
C  the correct random number generator for your compiler.
C
```

```
      DO 10 I=1,NUM
        XI(I) = XMININ+(XMAXIN-XMININ)*DSRND5()
        YI(I) = YMININ+(YMAXIN-YMININ)*DSRND5()
        ZI(I) = (XI(I)-0.25)**2 + (YI(I)-0.50)**2
   10 CONTINUE
C
C  Create the output grid.
C
      DO 102 I=1,NX
        XO(I) = XMINOT+(REAL(I-1)/REAL(NX-1))*(XMAXOT-XMINOT)
  102 CONTINUE
      DO 103 J =1,NY
        YO(J)= YMINOT+(REAL(J-1)/REAL(NY-1))*(YMAXOT-YMINOT)
  103 CONTINUE
C
C  Interpolate.
C
      CALL DSGRID2S(NUM, XI, YI, ZI, NX, NY, XO, YO, OUTPUT, IER)
      IF (IER .NE. 0) THEN
        WRITE(6,520) IER
  520   FORMAT(' Error ',I3,' returned from DSGRID2S')
        STOP
      ENDIF
C
C  Plot a surface.
C
      CALL GOPKS (IERRF, ISZDM)
      CALL GOPWK (IWKID, LUNIT, IWTYPE)
      CALL GACWK (IWKID)
      CALL TDEZ2D(NX, NY, XO, YO, OUTPUT, RHO, THETA, PHI, 6)
      CALL FRAME()
      CALL GDAWK (IWKID)
      CALL GCLWK (IWKID)
      CALL GCLKS
C
      STOP
      END
      REAL FUNCTION DSRND5()
C
      DATA ISEED/1/
      SAVE ISEED
C
      ISEED = ISEED*1103515245 + 12345
      IT = IAND(ISHIFT(ISEED,-16),32767)
C
      DSRND5 = REAL(IT)/32767.
C
      RETURN
      END
```

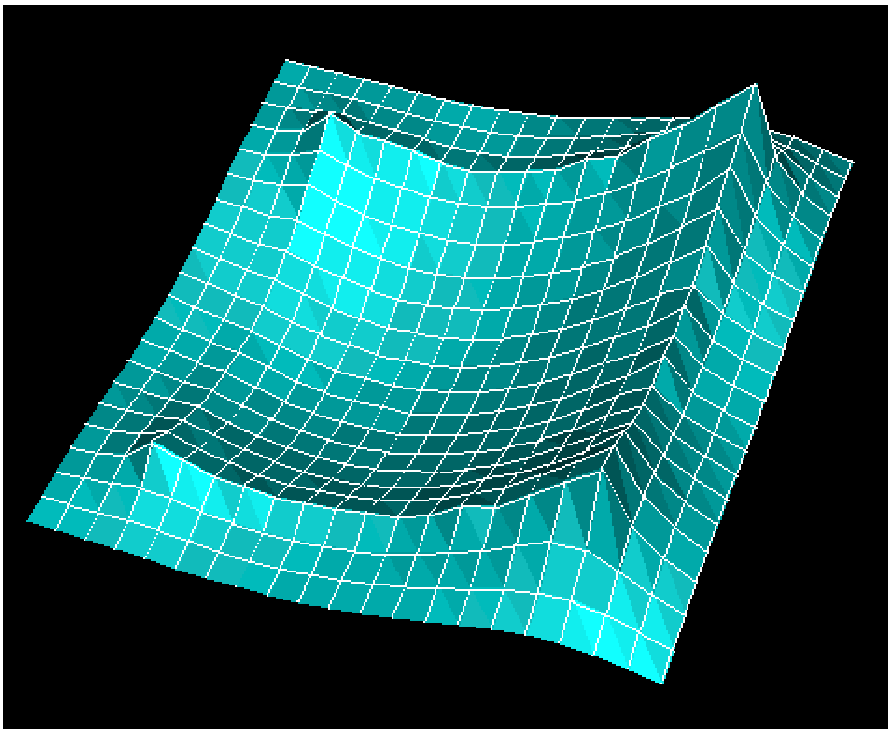
Example 5 - extrapolation (C single precision)

```
#include <stdio.h>
#include <stddef.h>
```

```
#include <stdlib.h>
#include <ncarg/ncargC.h>
#include <ncarg/gks.h>
#include <ncarg/ngmath.h>

#define NUM 5000
#define NX    21
#define NY    21
#define IWTYPE 1
#define WKID   1

main()
{
  int  i, j, k, ier;
  float xi[NUM], yi[NUM], zi[NUM];
  float xo[NX], yo[NY], *output, outr[NY][NX];
  float xminin =  0.00, yminin =  0.00, xmaxin = 1.00, ymaxin = 1.00;
  float xminot = -0.21, yminot = -0.21, xmaxot = 1.21, ymaxot = 1.21;
  float rho = 3., theta = -70., phi = 40.;

/*
 *  Create random data in three space and define a function.
 */
  for (i = 0; i < NUM; i++) {
    xi[i] = xminin+(xmaxin-xminin)*((float) rand() / (float) RAND_MAX);
    yi[i] = yminin+(ymaxin-yminin)*((float) rand() / (float) RAND_MAX);
    zi[i] = (xi[i]-0.25)*(xi[i]-0.25) + (yi[i]-0.50)*(yi[i]-0.50);
  }

/*
 *  Create the output grid.
 */
  for (i = 0; i < NX; i++) {
    xo[i] = xminot + ( (float) i / (float) (NX-1)) * (xmaxot-xminot);
  }
  for (j = 0; j < NY; j++) {
    yo[j] = yminot + ( (float) j / (float) (NY-1)) * (ymaxot-yminot);
  }

/*
 *  Interpolate.
 */
  output = c_dsgrid2s(NUM, xi, yi, zi, NX, NY, xo, yo, &ier);
  if (ier != 0) {
    printf(" Error %d returned from c_dsgrid2s\n",ier);
    exit(1);
  }
  gopen_gks ("stdout",0);
  gopen_ws (WKID, NULL, IWTYPE);
  gactivate_ws(WKID);

/*
 *  Reverse the array indices for plotting with tdez2d, since
 *  c_dsgrid returns its array in column dominate order.
 */
  for (i = 0; i < NX; i++) {
    for (j = 0; j < NY; j++) {
      outr[j][i] = output[i*NY+j];
    }
  }
  c_tdez2d(NX, NY, xo, yo, &outr[0][0], rho, theta, phi, 6);
```

```
  c_frame();

  gdeactivate_ws(WKID);
  gclose_ws(WKID);
  gclose_gks();
}
```

Plot for example 5 - extrapolation



Example 6 - 3D interpolation using single point mode (Fortran single precision)

```
      PROGRAM DSEX06
C
      PARAMETER (NUM=1000, NX=21, NY=21, NZ=21)
C
C Define error file, Fortran unit number, and workstation type,
C and workstation ID.
C
      PARAMETER (IERRF=6, LUNIT=2, IWTYPE=1, IWKID=1)
C
```

```
      REAL XI(NUM), YI(NUM), ZI(NUM), U(NUM)
      REAL XO(NX),  YO(NY),  ZO(NZ),  OUTPUT(NX,NY,NZ)
      INTEGER I, J, K, IER
      DATA XMIN,YMIN,ZMIN,XMAX,YMAX,ZMAX / -2., -2., -2., 2., 2., 2./
C
C  3D example.
C
      DO 10 I=1,NUM
        XI(I) = XMIN+(XMAX-XMIN)*DSRND6()
        YI(I) = YMIN+(YMAX-YMIN)*DSRND6()
        ZI(I) = ZMIN+(ZMAX-ZMIN)*DSRND6()
         U(I) = XI(I)**2 + YI(I)**2 + ZI(I)**2
   10 CONTINUE
C
C  Create the output grid.
C
      DO 102 I=1,NX
        XO(I) = XMIN+(REAL(I-1)/REAL(NX-1))*(XMAX-XMIN)
  102 CONTINUE
C
      DO 103 J =1,NY
        YO(J)= YMIN+(REAL(J-1)/REAL(NY-1))*(YMAX-YMIN)
  103 CONTINUE
C
      DO 104 K=1,NZ
        ZO(K) = ZMIN+(REAL(K-1)/REAL(NZ-1))*(ZMAX-ZMIN)
  104 CONTINUE
C
C  Interpolate.
C
      DO 110 I=1,NX
        DO 111 J=1,NY
          DO 112 K=1,NZ
            CALL DSPNT3S (NUM,XI,YI,ZI,U,1,XO(I),YO(J),ZO(K),
     +                    OUTPUT(I,J,K),IER)
            IF (IER .NE. 0) THEN
              WRITE(6,520) IER
  520         FORMAT(' Error ',I3,' returned from DSPNT3S')
              STOP
            ENDIF
  112     CONTINUE
  111   CONTINUE
  110 CONTINUE
C
C  Plot an isosurface.
C
C Open GKS and define the foreground and background color.
C
      CALL GOPKS (IERRF, ISZDM)
      CALL GOPWK (IWKID, LUNIT, IWTYPE)
      CALL GACWK (IWKID)
C
      CALL TDEZ3D(NX, NY, NZ, XO, YO, ZO, OUTPUT, 3.0,
     +            2., -35., 65., 6)
      CALL FRAME()
C
C  2D Example.
C
      CALL GDAWK (IWKID)
      CALL GCLWK (IWKID)
      CALL GCLKS
```

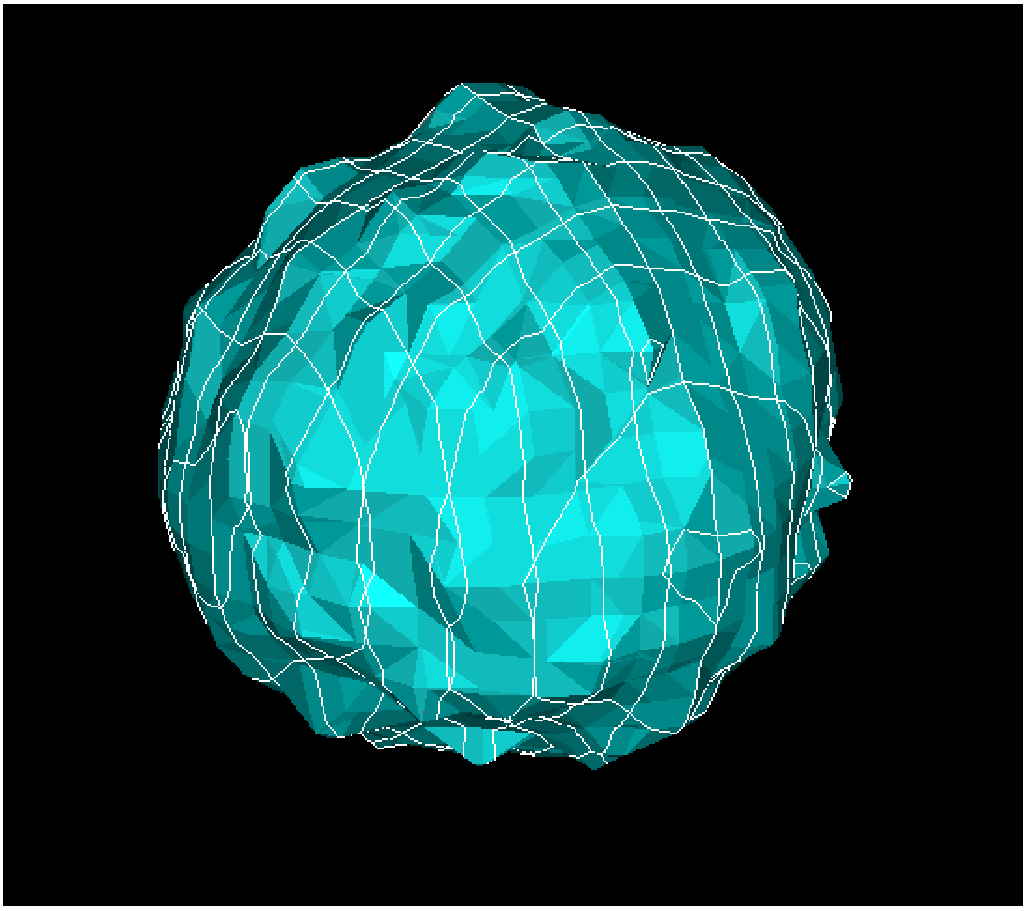
```
C
      STOP
      END
      REAL FUNCTION DSRND6()
C
      DATA ISEED/1/
      SAVE ISEED
C
      ISEED = ISEED*1103515245 + 12345
      IT = IAND(ISHIFT(ISEED,-16),32767)
C
      DSRND6 = REAL(IT)/32767.
C
      RETURN
      END
```

Example 6 - a simple 3D interpolation (C single precision)

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <ncarg/ncargC.h>
#include <ncarg/gks.h>
#include <ncarg/ngmath.h>

#define NUM 1000
#define NX    21
#define NY    21
#define NZ    21
#define IWTYPE 1
#define WKID   1

main()
{
  int  i, j, k, ier;
  float xi[NUM], yi[NUM], zi[NUM], u[NUM];
  float xo[NX], yo[NY], zo[NZ], output[NX][NY][NZ], outr[NZ][NY][NX];
  float xmin = -2.0, ymin = -2.0, zmin = -2.0;
  float xmax =  2.0, ymax =  2.0, zmax =  2.0;

/*
 *  Create random data in three space and define a function.
 */
  for (i = 0; i < NUM; i++) {
    xi[i] = xmin+(xmax-xmin)*((float) rand() / (float) RAND_MAX);
    yi[i] = ymin+(ymax-ymin)*((float) rand() / (float) RAND_MAX);
    zi[i] = zmin+(zmax-zmin)*((float) rand() / (float) RAND_MAX);
     u[i] = xi[i]*xi[i] + yi[i]*yi[i] + zi[i]*zi[i];
  }

/*
 *  Create the output grid.
```

```
 */
  for (i = 0; i < NX; i++) {
    xo[i] = xmin + ( (float) i / (float) (NX-1)) * (xmax-xmin);
  }
  for (j = 0; j < NY; j++) {
    yo[j] = ymin + ( (float) j / (float) (NY-1)) * (ymax-ymin);
  }
  for (k = 0; k < NZ; k++) {
    zo[k] = zmin + ( (float) k / (float) (NZ-1)) * (zmax-zmin);
  }

/*
 *  Interpolate a single point at a time.
 */
  for (i = 0; i < NX; i++) {
    for (j = 0; j < NY; j++) {
      for (k = 0; k < NZ; k++) {
        c_dspnt3s(NUM,xi,yi,zi,u,1,&xo[i],&yo[j],&zo[k],
                  &output[i][j][k],&ier);
        if (ier != 0) {
          printf(" Error %d returned from c_dspnt3s\n",ier);
          exit(1);
        }
      }
    }
  }

/*
 *  Plot an isosurface.
 */
  gopen_gks ("stdout",0);
  gopen_ws (WKID, NULL, IWTYPE);
  gactivate_ws(WKID);

/*
 *  Reorder the array elements for c_tdez3d, since c_dsgrid3s
 *  returns an array in column dominate order.
 */
  for (i = 0; i < NX; i++) {
    for (j = 0; j < NY; j++) {
      for (k = 0; k < NZ; k++) {
        outr[k][j][i] = output[i][j][k];
      }
    }
  }
  c_tdez3d(NX, NY, NZ, xo, yo, zo, &outr[0][0][0], 3.0, 2., -35., 65., 6);
  c_frame();

  gdeactivate_ws(WKID);
  gclose_ws(WKID);
  gclose_gks();
}
```

Plot for example 6 - single point mode



Dsgrid References

- Watson, Dave, Contouring: - A Guide To The Analysis and Display of Spatial Data, Pergamon Press, 1992. ISBN 0-08-040286-0

Dsgrid Index

The alphabetic characters that follow entries in this index refer to citations. For example, if *Entry* is followed by "A, B", then "A" and "B" are links to two references to *Entry* in the Dsgrid documentation modules.

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

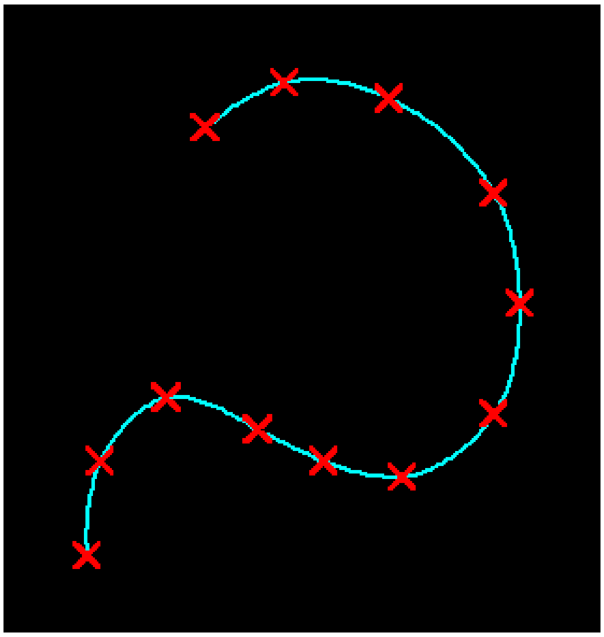
Q

R

S

T

Fitgrid, a 1D and 2D interpolation package



Introduction (read this first)

Learning how to use the package

Fitgrid Table of contents

Credits

Introduction

Learning how to use Fitgrid

Useful reference charts and tables

Examples

References

Fitgrid credits

Acknowledgments

Author: Fred Clare

Editor: Fred Clare. Final copyedit by Brian Bevirt.

Production: Fred Clare

Special thanks: Fitgrid is based on Professor Alan Cline's package Fitpack, an early version of which is available from netlib. An expanded version of Fitpack, as well as complete hardcopy documentation, can be obtained from: Pleasant Valley Software, 8603 Altus Cove, Austin TX 78759. Dr. Cline has our great gratitude for allowing us to incorporate his early version of Fitpack into the ngmath library.

Copyright

Trademark

Fitgrid Introduction

Fitgrid functionality

Fitgrid is a package in the ngmath library for interpolating one-dimensional and two-dimensional data. The package has the following functionality:

- Interpolating single-valued functions in one dimension.
- Interpolating parametric curves in the plane.
- Computing an interpolatory surface through a rectangular grid of functional values.
- Interpolating periodic functions in one dimension.
- Finding integrals and derivatives of interpolated functions.

See the comparative analysis of the ngmath packages for details on which ngmath package is most appropriate for your problem.

Fitgrid algorithm

The functions in Fitgrid utilize splines under tension for the interpolations. Splines under tension are controlled by a scalar tension factor that can be adjusted to provide a smooth transition from cubic spline interpolation to linear interpolation. To do its interpolations, Fitgrid fits a tension spline to the input data and then picks its interpolated values from the fitted spline function.

The documentation for Natgrid contains a general discussion of interpolation algorithms.

Fitgrid history

Fitgrid is based on Alan Cline's package Fitpack, an early version of which is available from netlib. Fitpack uses splines under tension to interpolate in one and two dimensions. An expanded version of Fitpack, as well as complete hardcopy documentation, can be obtained from: Pleasant Valley Software, 8603 Altus Cove, Austin TX 78759. Dr. Cline has our great gratitude for allowing us to incorporate his early version of Fitpack into the ngmath library.

Learning how to use Fitgrid

You should now proceed to the module on learning how to use Fitgrid.

Learning how to use Fitgrid

Briefly browse the table of contents to get a general feel for what is in the Fitgrid package.

Fitgrid can be accessed by using any one of three interfaces: Fortran, C, and NCL.

The Fortran interface

This interface is identical to the original Fitpack interface. Examine the procedures overview module for a summary of the Fortran entries. From there you can click on the desired procedure name for details on how to call it correctly. Also, see the examples section for learning how to use the Fortran interface.

The C interface

This interface has been added to the original Fitpack package. The C interface differs from the Fortran interface in that many of the arguments to the original procedures have been converted into user-settable global parameters. The idea is to set values for the appropriate parameters and then invoke the desired procedure. For a list of all parameters, see the parameters module. The default values for the parameters are listed there. The functions c_ftseti, c_ftsetr, c_ftsetc, c_ftsetfa are used for setting parameter values.

Examine the procedures overview module for a summary of the C entries. From there you can click on the desired function name for details on how to call it correctly. The documentation for each C function indicates which parameters are appropriate for that function. Also, see the examples section for learning how to use the C interface.

The NCL interface

The NCL interface is much simpler than the Fortran or C interfaces. The basic way to proceed is the same as for the C interface: set values for global parameters and invoke the desired NCL function or procedure. The parameters are listed in the parameters module.

Examine the NCL Fitgrid documentation to learn how to use the NCL interface. Note that there is only a single procedure needed for setting values for the parameters. Also, see the examples section for learning how to use the NCL interface.

Table of Fitgrid control parameters

The following table summarizes all of the parameters used for controlling the behavior of Fitgrid. These parameters apply only to the C and NCL interfaces.

For C, the values for these parameters can be set by using the procedures c_ftseti, c_ftsetr, c_ftsetc, and c_ftsetfa. For NCL programs, the parameters can be set using the NCL procedure ftsetp.

Parameter name	parameter type	description	legal values	default value
eps	float	smoothing parameter for ftcurvs and ftcurvps	0. <= eps and 1. >= eps	none
sig	float	the value of the tension factor	>= 0.	1.
sl1	float	the slope of the curve at the first point (see parameter **sf1**).	any float	0.
sln	float	the slope of the curve at the final point (see parameter **sf1**).	any float	0.

sf1	int	controls the use of parameters **sl1** and **sl2**	0=use both 1=use **sl1** only 2=use **sl2** only 3=use neither	3
sf2	int	controls the use of parameters **smt** and **eps**	0=use neither 1=use both 2=use **smt** only 3=use **eps** only	3
smt	float	smoothing parameter for ftcurvs and ftcurvps	> 0.	none
zx1	float array	X partials at left for ftsurf	any floats	none
zxm	float array	X partials at right for ftsurf	any floats	none
zy1	float array	Y partials at bottom for ftsurf	any floats	none
zyn	float array	Y partials at top for ftsurf	any floats	none
z11	float	X-Y partial at lower left for ftsurf	any float	none
zm1	float	X-Y partial at lower right for ftsurf	any float	none
z1n	float	X-Y partial at upper left for ftsurf	any float	none
zmn	float	X-Y partial at upper right for ftsurf	any float	none
df1	int	flag indicating if **zx1** is user supplied	0 = yes 1 = no	1
df2	int	flag indicating if **zxm** is user supplied	0 = yes 1 = no	1
df3	int	flag indicating if **zy1** is user supplied	0 = yes 1 = no	1
df4	int	flag indicating if **zyn** is user supplied	0 = yes 1 = no	1
df5	int	flag indicating if **z11** is user supplied	0 = yes 1 = no	1
df6	int	flag indicating if **zm1** is user supplied	0 = yes 1 = no	1
df7	int	flag indicating if **z1n** is user supplied	0 = yes 1 = no	1
df8	int	flag indicating if	0 = yes	1

```
          |            | zmn is user supplied| 1 = no        |
------------------------------------------------------------------------------
```

Detailed description of Fitgrid procedures

CURV1 - calculate values for CURV2 usage

This subroutine calculates certain values that are used by CURV2 in order to compute an interpolatory spline under tension through a sequence of functional values. The actual computation of the interpolated values must be done using CURV2 .

```
---------------------------------------------------------------
            Argument | Type     |  Mode   | Dimension
---------------------------------------------------------------
CALL CURV1  (N,      | Integer  | Input   |
            X,       | Real     | Input   | N
            Y,       | Real     | Input   | N
            SLP1,    | Real     | Input   |
            SLPN,    | Real     | Input   |
            ISLPSW,  | Integer  | Input   |
            YP,      | Real     | Output  | N
            TEMP,    | Real     | Input   | N
            SIGMA,   | Real     | Input   |
            IER)     | Integer  | Output  |
---------------------------------------------------------------
```

N
: The number of input data values. (N > 1)

X
: An array containing the abscissae for the input function.

Y
: An array containing the functional values of the input data -- (Y(K) is the functional value at X(K) for K=1,N).

SLP1
: A user-specified value for the desired slope at X(1). See ISLPSW below if you want to have a value calculated internally.

SLPN
: A user-specified value for the desired slope at X(N). See ISLPSW below if you want to have a value calculated internally.

ISLPSW
: A switch to indicate whether the slopes at the end points should be calculated internally.
  - = 0 if SLP1 and SLPN are user-specified.
  - = 1 if SLP1 is user-specified, but SLPN is internally calculated.
  - = 2 if SLPN is user-specified, but SLP1 is internally calculated.
  - = 3 if SLP1 and SLPN are internally calculated.

YP
: Contains values for the second derivative (CURV1 computes these).

TEMP
: Scratch space.

SIGMA
: Tension factor. Values near zero result in a cubic spline; large values (e.g. 50) result in nearly a polygonal line. A typical value is 1.

IER
: An error return value. If IER is returned as 0, then no errors were detected.
  = 1 if N is less than 2.
  = 2 if X values are not strictly increasing.

CURV2 - interpolate at a specified point

This function interpolates a value at a specified point using a spline under tension. CURV1 must be called prior to calling CURV2, and the values thus obtained used as input to CURV2. CURV2 returns the interpolated function value at the X-coordinate specified.

```
--------------------------------------------------------------
                Argument  | Type      |  Mode   | Dimension
--------------------------------------------------------------
FUNCTION CURV2 (T,        | Real      | Input   |
                N,        | Integer   | Input   |
                X,        | Real      | Input   | N
                Y,        | Real      | Input   | N
                YP,       | Real      | Input   | N
                SIGMA)    | Real      | Input   |
--------------------------------------------------------------
```

T
: The abscissa for which an interpolated function value is desired.

N
: The number of input data values. (N > 1)

X
: An array containing the abscissae for the input function.

Y
: An array containing the functional values of the input data -- Y(K) is the functional value at X(K) for K=1,N.

YP
: Contains values for the second derivative (as calculated by CURV1).

SIGMA
: Tension factor. Values near zero result in a cubic spline; large values (e.g. 50) result in nearly a polygonal line. A typical value is 1.

CURVD - calculate a derivative at a specified point

This function calculates a derivative at a specified point using a spline under tension. CURV1 must be called prior to calling CURVD, and the values thus obtained used as input to CURVD. The derivative value is returned.

```
-------------------------------------------------------------
                Argument | Type      |  Mode   | Dimension
-------------------------------------------------------------
FUNCTION CURVD (T,       | Real      | Input   |
                N,       | Integer   | Input   |
                X,       | Real      | Input   | N
                Y,       | Real      | Input   | N
                YP,      | Real      | Input   | N
                SIGMA)   | Real      | Input   |
-------------------------------------------------------------
```

T
: The abscissa for which an interpolated function value is desired.

N
: The number of input data values. (N > 1)

X
: An array containing the abscissae for the input function.

Y
: An array containing the functional values of the input function -- Y(K) is the functional value at X(K) for K=1,N.

YP
: Contains values for the second derivative (as calculated by CURV1).

SIGMA
: Tension factor. Values near zero result in a cubic spline; large values (e.g. 50) result in nearly a polygonal line. A typical value is 1.

CURVI - integrate a spline curve

This function calculates an integral between two specified limits. CURV1 must be called prior to calling CURVI, and the values thus obtained used as input to CURVI. The value of the integral is returned.

```
-------------------------------------------------------------
                Argument | Type      |  Mode   | Dimension
-------------------------------------------------------------
FUNCTION CURVI (XL,      | Real      | Input   |
                XU,      | Real      | Input   |
                N,       | Integer   | Input   |
                X,       | Real      | Input   | N
                Y,       | Real      | Input   | N
                YP,      | Real      | Input   | N
                SIGMA)   | Real      | Input   |
-------------------------------------------------------------
```

XL
: The lower limit of the integration.

XR
: The upper limit of the integration.

N
: The number of input data values. (N > 1)

X
: An array containing the abscissae for the input function.

Y
: An array containing the functional values of the input function -- Y(K) is the functional value at X(K) for K=1,N.

YP
: Contains values for the second derivative (as calculated by CURV1).

SIGMA
: Tension factor. Values near zero result in a cubic spline; large values (e.g. 50) result in nearly a polygonal line. A typical value is 1.

CURVP1 - calculate values for CURVP2 usage

This subroutine calculates certain values that are used by CURVP2 in order to compute an interpolatory spline under tension through a sequence of functional values for a periodic function. The actual computation of the interpolated values must be done using CURVP2 .

```
-------------------------------------------------------------
             Argument  | Type     |  Mode   | Dimension
-------------------------------------------------------------
CALL CURVP1  (N,       | Integer  | Input   |
             X,        | Real     | Input   | N
             Y,        | Real     | Input   | N
             P,        | Real     | Input   |
             YP,       | Real     | Output  | N
             TEMP,     | Real     | Input   | N x 2
             SIGMA,    | Real     | Input   |
             IER)      | Integer  | Output  |
-------------------------------------------------------------
```

N
: The number of input data values. (N > 1)

X
: An array containing the abscissae for the input function.

Y
: An array containing the functional values of the input function -- Y(K) is the functional value at X(K) for K=1,N.

P
: The period of the function. P must be *greater than* X(N)-X(1).

YP
: Contains values for the second derivative (these are calculated by CURVP1).

TEMP
: Scratch space.

SIGMA
: Tension factor. Values near zero result in a cubic spline; large values (e.g. 50) result in nearly a polygonal line. A typical value is 1.

IER
: An error return value. If IER is returned as 0, then no errors were detected.
: = 1 if N is less than 2.

= 2 if P is less than or equal to X(N)-X(1).
= 3 if the X values are not strictly increasing.

CURVP2 - interpolate a periodic function at a specified point

---

This function interpolates a value at a specified point using a spline under tension. CURVP1 must be called prior to calling CURVP2, and the values thus obtained used as input to CURVP2. The interpolated value is returned.

```
----------------------------------------------------------------
                 Argument | Type      |  Mode  | Dimension
----------------------------------------------------------------
FUNCTION CURVP2 (T,       | Real      | Input  |
                 N,       | Integer   | Input  |
                 X,       | Real      | Input  | N
                 Y,       | Real      | Input  | N
                 P,       | Real      | Input  |
                 YP,      | Real      | Input  | N
                 SIGMA)   | Real      | Input  |
----------------------------------------------------------------
```

T
: The abscissa for which an interpolated function value is desired.

N
: The number of input data values. (N > 1)

X
: An array containing the abscissae for the input function.

Y
: An array containing the functional values for the input function -- Y(K) is the functional value at X(K) for K=1,N.

P
: The period of the function.

YP
: Contains values for the second derivative (as computed by CURVP1).

SIGMA
: Tension factor. Values near zero result in a cubic spline; large values (e.g. 50) result in nearly a polygonal line. A typical value is 1.

CURVPI - integrate a periodic spline curve

This function calculates an integral between two specified limits. CURVP1 must be called prior to calling CURVPI, and the values thus obtained used as input to CURVPI. The value of the integral is returned.

```
----------------------------------------------------------------
                Argument | Type      |  Mode  | Dimension
----------------------------------------------------------------
FUNCTION CURVI (XL,      | Real      | Input  |
                XU,      | Real      | Input  |
```

N,	Integer	Input	
X,	Real	Input	N
Y,	Real	Input	N
P,	Real	Input	
YP,	Real	Input	N
SIGMA)	Real	Input	

XL
: The lower limit of the integration.

XR
: The upper limit of the integration.

N
: The number of input data values. (N > 1)

X
: An array containing the abscissae for the input function.

Y
: An array containing the functional values of the input function -- Y(K) is the functional value at X(K) for K=1,N.

P
: The period of the function.

YP
: Contains values for the second derivative (as calculated by CURVP1).

SIGMA
: Tension factor. Values near zero result in a cubic spline; large values (e.g. 50) result in nearly a polygonal line. A typical value is 1.

CURVS - calculate values for a smoothing spline

This subroutine calculates certain values that are used by CURV2 in order to compute an interpolatory smoothing spline under tension through a sequence of functional values. The actual computation of the interpolated values must be done using CURV2.

Three parameters are used to control the degree of smoothness -- D, S, and EPS.

The parameter D is a value indicating the degree of confidence in the accuracy of the input function values -- it should be an approximation of the standard deviation of error. Effectively the value of D controls how close the smoothed curve comes to the input data points. If D is small then the interpolated curve will pass close to the input data. The larger the value of D, the more freedom the smooth curve has in how close it comes to the input data values.

S is a more subtle global smoothing parameter. S must be non-negative. For small values of S, the interpolated curve approximates the tension spline and for larger values of S, the curve is smoother. A reasonable value for EPS is REAL(N).

EPS controls the precision to which S is interpreted; EPS must be between 0. and 1. inclusive. A reasonable value for S is SQRT(2./REAL(N)).

Argument	Type	Mode	Dimension
CALL CURVS (N,	Integer	Input	
X,	Real	Input	N
Y,	Real	Input	N
D,	Real	Input	N
ISW,	Integer	Input	
S,	Real	Input	
EPS,	Real	Input	
YS,	Real	Output	N
YSP,	Real	Output	N
SIGMA,	Real	Input	
TEMP,	Real	Input	N x 9
IER)	Integer	Output	

N
: The number of input data values. (N > 1)

X
: An array containing the abscissae for the input function.

Y
: An array containing the functional values for the input function -- Y(K) is the functional value at X(K) for K=1,N.

D
: A user-specified value containing the observed weights. D may be either an array or a scalar, depending on the value of ISW (as described below).

ISW
: A switch for interpreting the value of D. If ISW=0, then D is an array of length N (D contains an individual error estimate for each input data value); if ISW=1, then D is a scalar that serves as an error estimate for every single data item.

S
: Contains the value for smoothing. S must be non-negative. Larger values for S yield greater smoothing. A reasonable value is REAL(N).

EPS
: Contains a tolerance value for the relative precision to which S should be interpreted. EPS must be between 0. and 1. inclusive. A reasonable value is SQRT(2./REAL(N)).

YS
: An array of length N.

YSP
: An array of length N.

SIGMA
: Tension factor. Values near zero result in a cubic spline; large values (e.g. 50) result in nearly a polygonal line. A typical value is 1.

TEMP
: Scratch space.

IER
: An error return value. If IER is returned as 0, then no errors were detected.

    = 1 if N is less than 2.
    = 2 if S is negative.
    = 3 if EPS is negative or greater than 1.
    = 4 if X values are not strictly increasing.
    = 5 if D is negative.

CURVPS - calculate values for a smoothing spline

This subroutine calculates certain values that are used by CURVP2 in order to compute an interpolatory smoothing spline under tension through a sequence of functional values for a periodic function. The actual computation of the interpolated values must be done using CURVP2.

Three parameters are used to control the degree of smoothness -- D, S, and EPS.

The parameter D is a value indicating the degree of confidence in the accuracy of the input function values -- it should be an approximation of the standard deviation of error. Effectively the value of D controls how close the smoothed curve comes to the input data points. If D is small then the interpolated curve will pass close to the input data. The larger the value of D, the more freedom the smooth curve has in how close it comes to the input data values.

S is a more subtle global smoothing parameter. S must be non-negative. For small values of S, the curve approximates the tension spline and for larger values of S, the curve is smoother. A reasonable value for S is REAL(N).

EPS controls the precision to which S is interpreted; EPS must be between 0. and 1. inclusive. A reasonable value for EPS is SQRT(2./REAL(N)).

```
-------------------------------------------------------------------
              Argument | Type      |  Mode   | Dimension
-------------------------------------------------------------------
CALL CURVPS (N,        | Integer   | Input   |
              X,       | Real      | Input   | N
              Y,       | Real      | Input   | N
              P,       | Real      | Input   |
              D,       | Real      | Input   | N
              ISW,     | Integer   | Input   |
              S,       | Real      | Input   |
              EPS,     | Real      | Input   |
              YS,      | Real      | Output  | N
              YSP,     | Real      | Output  | N
              SIGMA,   | Real      | Input   |
              TEMP,    | Real      | Input   | N x 11
              IER)     | Integer   | Output  |
-------------------------------------------------------------------
```

N
: The number of input data values. (N > 1)

X
: An array containing the abscissae for the input function.

Y
: An array containing the functional values of the input function -- Y(K) is the functional value at X(K) for K=1,N.

P
: The period of the function. P must be *greater than* X(N)-X(1).

D
: A user-specified value containing the observed weights. D may either be an array or a scalar,

depending on the value of ISW.

ISW
: A switch for interpreting the value of D. If ISW=0, then D is an array of length N (an error estimate for each input data value); if ISW=1, then D is a scalar that serves as an error estimate for every single data item.

S
: Contains the value for smoothing. S must be non-negative. Larger values for S yield greater smoothing. A reasonable value is REAL(N).

EPS
: Contains a tolerance value for the relative precision to which S should be interpreted. EPS must be between 0. and 1. inclusive. A reasonable value is SQRT(2./REAL(N)).

YS
: An array of length N.

YSP
: An array of length N.

SIGMA
: Tension factor. Values near zero result in a cubic spline; large values (e.g. 50) result in nearly a polygonal line. A typical value is 1.

TEMP
: Scratch space.

IER
: An error return value. If IER is returned as 0, then no errors were detected.
    - = 1 if N is less than 2.
    - = 2 if S is negative.
    - = 3 if EPS is negative or greater than 1.
    - = 4 if X values are not strictly increasing.
    - = 5 if D is negative.
    - = 6 if P is less than or equal to X(N)-X(1).

KURV1 - calculate values for KURV2 usage

This subroutine calculates certain values that are used by KURV2 in order to compute an interpolatory spline under tension through a sequence of points in the plane. The actual computation of the interpolated values must be done using KURV2 .

	Argument	Type	Mode	Dimension
CALL KURV1	(N,	Integer	Input	
	X,	Real	Input	N
	Y,	Real	Input	N
	SLP1,	Real	Input	
	SLPN,	Real	Input	
	ISLPSW,	Integer	Input	
	XP,	Real	Output	N
	YP,	Real	Output	N
	TEMP,	Real	Input	N
	S,	Real	Output	N
	SIGMA,	Real	Input	
	IER)	Integer	Output	

N
: The number of input data values. (N > 1)

X
: An array containing the X values of the points.

Y
: An array containing the Y values of the points. Adjacent pairs of points must be distinct.

SLP1
: A value (in radians) containing the slope at (X(1),Y(1)). The angles are measured counter-clockwise from the X axis and the positive sense of the curve is assumed to be that moving from point 1 to point N. A value for SLP1 may be omitted as indicated by the switch ISLPSW.

SLPN
: A value (in radians) containing the slope at (X(N),Y(N)). The angles are measured counter-clockwise from the X axis and the positive sense of the curve is assumed to be that moving from point 1 to point N. A value for SLP2 may be omitted as indicated by the switch ISLPSW.

ISLPSW
: A switch to indicate whether the slopes at the end points should be calculated internally.
  = 0 if SLP1 and SLPN are user-specified.
  = 1 if SLP1 is user-specified, but SLPN calculated internally.
  = 2 if SLPN is user-specified, but SLP1 calculated internally.
  = 3 if SLP1 and SLPN are internally calculated.

XP
: An array of length N.

YP
: An array of length N.

TEMP
: Scratch space.

S
: An array of length N. On output, S contains the polygonal arclengths of the curve.

SIGMA
: Tension factor. Values near zero result in a cubic spline; large values (e.g. 50) result in nearly a polygonal line. A typical value is 1.

IER
: An error return value. If IER is returned as 0, then no errors were detected.
  = 1 if N is less than 2.
  = 2 if a pair of adjacent points coincide.

KURV2 - does interpolation for parametric curves

The interpolated curve is parameterized by mapping points in the interval [0.,1.] onto the interpolated curve. The resulting curve has a parametric representation both of whose components are splines under tension and functions of the polygonal arc length. The value 0. is mapped onto (X[1],Y[1]) and the value 1. is mapped onto (X[N],Y[N]). KURV1 must be called before calling KURV2.

```
-------------------------------------------------------------
            Argument | Type     |  Mode  | Dimension
-------------------------------------------------------------
CALL KURV2  (T,      | Real     | Input  |
            XS,      | Real     | Output |
            YS,      | Real     | Output |
            N,       | Integer  | Input  |
            X,       | Real     | Input  | N
            Y,       | Real     | Input  | N
            XP,      | Real     | Input  | N
            YP,      | Real     | Input  | N
            S,       | Real     | Input  | N
            SIGMA)   | Real     | Input  |
-------------------------------------------------------------
```

T
: A value to be mapped to a point on the curve. The interval [0.,1.] is mapped onto the curve such that 0. is mapped to (X(1),Y(1)) and 1. is mapped to (X(N),Y(N)). Values outside of [0.,1.] result in extrapolation.

XS
: Contains the X coordinate of the interpolated point that T maps to.

YS
: Contains the Y coordinate of the interpolated point that T maps to.

N
: The number of input data points. (N > 1)

X
: An array containing the X values of the input points.

Y
: An array containing the Y values of the input points. Adjacent pairs of points must be distinct.

XP
: An array of length N as computed by KURV1.

YP
: An array of length N as computed by KURV1.

S
: An array of length N containing the polygonal arc lengths of the curve.

SIGMA
: Tension factor. Values near zero result in a cubic spline; large values (e.g. 50) result in nearly a polygonal line. A typical value is 1.

KURVD - interpolation and derivatives for parametric curves

This subroutine behaves like KURV2 except that in addition it returns the first and second derivatives of the component functions in the parameterization. The interval [0.,1.] maps onto a curve in the plane. The resulting curve has a parametric representation both of whose components are splines under tension and functions of the polygonal arc length. KURV1 must be called before calling KURVD.

```
-------------------------------------------------------------
            Argument | Type     |  Mode  | Dimension
-------------------------------------------------------------
CALL KURVD  (T,      | Real     | Input  |
            XS,      | Real     | Output |
```

```
            YS,        | Real     | Output |
            XST,       | Real     | Output |
            YST,       | Real     | Output |
            XSTT,      | Real     | Output |
            YSTT,      | Real     | Output |
            N,         | Integer  | Input  |
            X,         | Real     | Input  | N
            Y,         | Real     | Input  | N
            XP,        | Real     | Input  | N
            YP,        | Real     | Input  | N
            S,         | Real     | Input  | N
            SIGMA)     | Real     | Input  |
-----------------------------------------------------------------
```

T
: A value to be mapped to a point on the curve. The interval [0.,1.] is mapped onto the curve such that 0. is mapped to (X(1),Y(1)) and 1. is mapped to (X(N),Y(N)). Values outside of [0.,1.] result in extrapolation.

XS
: Contains the X coordinate of the point that T maps to.

YS
: Contains the Y coordinate of the point that T maps to.

XST
: Contains the first derivatives of the X component with respect to T.

YST
: Contains the first derivatives of the Y component with respect to T.

XSTT
: Contains the second derivatives of the X component with respect to T.

YSTT
: Contains the second derivatives of the X component with respect to T.

N
: The number of input data points. (N > 1)

X
: An array containing the X values of the input points.

Y
: An array containing the Y values of the input points. Adjacent pairs of points must be distinct.

XP
: An array of length N as computed by KURV1.

YP
: An array of length N as computed by KURV1.

S
: An array of length N containing the polygonal arc lengths of the curve, as computed by KURV1.

SIGMA
: Tension factor. Values near zero result in a cubic spline; large values (e.g. 50) result in nearly a polygonal line. A typical value is 1.

KURVP1 - calculate values for KURVP2 usage

This subroutine calculates certain values that are used by KURVP2 in order to compute an interpolatory spline under tension through a sequence of points in the plane that form a closed curve. The actual computation of the interpolated values must be done using KURVP2 .

```
---------------------------------------------------------------
             Argument | Type     |  Mode   | Dimension
---------------------------------------------------------------
CALL KURVP1  (N,      | Integer  | Input   |
             X,       | Real     | Input   | N
             Y,       | Real     | Input   | N
             XP,      | Real     | Output  | N
             YP,      | Real     | Output  | N
             TEMP,    | Real     | Input   | N x 2
             S,       | Real     | Output  | N
             SIGMA,   | Real     | Input   |
             IER)     | Integer  | Output  |
---------------------------------------------------------------
```

N
: The number of input data points. (N > 1)

X
: An array containing the X values of the points.

Y
: An array containing the Y values of the points. Adjacent pairs of points must be distinct.

XP
: An array of length N.

YP
: An array of length N.

TEMP
: Scratch space.

S
: An array of length N.

SIGMA
: Tension factor. Values near zero result in a cubic spline; large values (e.g. 50) result in nearly a polygonal line. A typical value is 1.

IER
: An error return value. If IER is returned as 0, then no errors were detected.
  = 1 if N is less than 2.
  = 2 if a pair of adjacent points coincide.

KURVP2 - does interpolation for closed parametric curves

The interpolated curve is parameterized by mapping points in the interval [0.,1.] onto an interpolated closed curve. The resulting curve has a parametric representation both of whose components are splines under tension and functions of the polygonal arc length. The value 0. is mapped onto (X[1],Y[1]) and the value 1. is also mapped onto (X[1],Y[1]), since the interpolated curve is closed. KURVP1 must be called before calling KURVP2.

```
---------------------------------------------------------------
             Argument | Type     |  Mode   | Dimension
---------------------------------------------------------------
CALL KURVP2  (T,      | Real     | Input   |
             XS,      | Real     | Output  |
```

```
             YS,        | Real     | Output |
             N,         | Integer  | Input  |
             X,         | Real     | Input  | N
             Y,         | Real     | Input  | N
             XP,        | Real     | Input  | N
             YP,        | Real     | Input  | N
             S,         | Real     | Input  | N
             SIGMA)     | Real     | Input  |
---------------------------------------------------------------
```

T
: A value to be mapped to a point on the curve. The interval [0.,1.] is mapped onto the curve such that 0. is mapped to (X(1),Y(1)) and 1. is mapped to (X(1),Y(1)), since the interpolated curve is closed. Any interval [TT,TT+1.] maps onto the entire curve.

XS
: Contains the X coordinate of the point that T maps to.

YS
: Contains the Y coordinate of the point that T maps to.

N
: The number of input data points. (N > 1)

X
: An array containing the X values of the points.

Y
: An array containing the Y values of the points. Adjacent pairs of points must be distinct.

XP
: An array of length N as computed by KURV1.

YP
: An array of length N as computed by KURV1.

S
: An array of length N containing the polygonal arc lengths of the curve, as computed by KURV1.

SIGMA
: Tension factor. Values near zero result in a cubic spline; large values (e.g. 50) result in nearly a polygonal line. A typical value is 1.

KURVPD - interpolation and derivatives for closed parametric curves

---

This subroutine behaves like KURVP2 except that in addition it returns the first and second derivatives of the component functions in the parameterization. The interval [0.,1.] maps onto a closed curve in the plane. The resulting curve has a parametric representation both of whose components are splines under tension and functions of the polygonal arc length. KURVP1 must be called before calling KURVPD.

```
---------------------------------------------------------------
             Argument   | Type     |  Mode  | Dimension
---------------------------------------------------------------
CALL KURVPD  (T,        | Real     | Input  |
             XS,        | Real     | Output |
             YS,        | Real     | Output |
             XST,       | Real     | Output |
             YST,       | Real     | Output |
```

```
          XSTT,      | Real     | Output  |
          YSTT,      | Real     | Output  |
          N,         | Integer  | Input   |
          X,         | Real     | Input   | N
          Y,         | Real     | Input   | N
          XP,        | Real     | Input   | N
          YP,        | Real     | Input   | N
          S,         | Real     | Input   | N
          SIGMA)     | Real     | Input   |
---------------------------------------------------------------
```

T
: A value to be mapped to a point on the curve. The interval [0.,1.] is mapped onto the curve such that 0. is mapped to (X(1),Y(1)) and 1. is mapped to (X(1),Y(1)). Any interval [TT,TT+1.] maps onto the entire closed curve.

XS
: Contains the X coordinate of the point that T maps to.

YS
: Contains the Y coordinate of the point that T maps to.

XST
: Contains the first derivatives of the X component with respect to T.

YST
: Contains the first derivatives of the Y component with respect to T.

XSTT
: Contains the second derivatives of the X component with respect to T.

YSTT
: Contains the second derivatives of the X component with respect to T.

N
: The number of input data points. (N > 1)

X
: An array containing the X values of the input points.

Y
: An array containing the Y values of the input points. Adjacent pairs of points must be distinct.

XP
: An array of length N as computed by KURVP1.

YP
: An array of length N as computed by KURVP1.

S
: An array of length N containing the polygonal arc lengths of the curve, as computed by KURVP1.

SIGMA
: Tension factor. Values near zero result in a cubic spline; large values (e.g. 50) result in nearly a polygonal line. A typical value is 1.

SURF1 - calculate values for SURF2 usage

This subroutine calculates certain values that are used by SURF2 in order to compute an interpolatory surface passing through a rectangular grid of function values. The surface computed is a tensor product of splines under tension. To calculate actual interpolated values, SURF2 must be called.

Argument	Type	Mode	Dimension
CALL SURF1 (M,	Integer	Input	
N,	Integer	Input	
X,	Real	Input	M
Y,	Real	Input	N
Z,	Real	Input	IZ x N
IZ,	Integer	Input	
ZX1,	Real	Input	N
ZXM,	Real	Input	N
ZY1,	Real	Input	M
ZYN,	Real	Input	M
ZXY11,	Real	Input	
ZXYM1,	Real	Input	
ZXY1N,	Real	Input	
ZXYMN,	Real	Input	
ISLPSW,	Integer	Input	
ZP,	Real	Output	M x N x 3
TEMP,	Real	Input	N + N + M
SIGMA,	Real	Input	
IER)	Integer	Output	

M
: The number of grid lines in the X direction. (M > 1)

N
: The number of grid lines in the Y direction. (N > 1)

X
: An array containing M X coordinates for grid lines in the X direction. These values must be strictly increasing.

Y
: An array containing N Y coordinates for grid lines in the Y direction. These values must be strictly increasing.

Z
: An array containing M x N functional values at the grid points; Z(I,J) contains the functional value at (X(I),Y(J)) for I=1,M and J=1,N.

IZ
: The row dimension of the matrix Z (IZ is greater than or equal to M).

ZX1
: An array containing N X-partial derivatives of the function along the line X(1), that is ZX1(J) is the X-partial derivative at point (X(1),Y(J)) for J=1,N. This parameter may be defaulted by setting ISLPSW appropriately.

ZXM
: An array containing N X-partial derivatives of the function along the line X(M), that is ZXM(J) is the X-partial derivative at point (X(M),Y(J)) for J=1,N. This parameter may be defaulted by setting ISLPSW appropriately.

ZY1
: An array containing M Y-partial derivatives of the function along the line Y(1), that is ZY1(I) is the Y-partial derivative at point (X(I),Y(1)) for I=1,M. This parameter may be defaulted by setting ISLPSW appropriately.

ZYN
: An array containing M Y-partial derivatives of the function along the line Y(N), that is ZY1(I) is

the Y-partial derivative at point (X(I),Y(N)) for I=1,M. This parameter may be defaulted by setting ISLPSW appropriately.

ZXY11
: The X-Y-partial derivative at (X(1),Y(1). This parameter may be defaulted by setting ISLPSW appropriately.

ZXYM1
: The X-Y-partial derivative at (X(M),Y(1). This parameter may be defaulted by setting ISLPSW appropriately.

ZXY1N
: The X-Y-partial derivative at (X(1),Y(N). This parameter may be defaulted by setting ISLPSW appropriately.

ZXYMN
: The X-Y-partial derivative at (X(M),Y(N). This parameter may be defaulted by setting ISLPSW appropriately.

ISLPSW
: A switch to indicate which boundary derivatives are user supplied and which should be estimated internally. Where

    I1 = 0 if ZX1 is user-supplied and 1 otherwise.
    I2 = 0 if ZXM is user-supplied and 1 otherwise.
    I3 = 0 if ZY1 is user-supplied and 1 otherwise.
    I4 = 0 if ZYN is user-supplied and 1 otherwise.
    I5 = 0 if ZXY11 is user-supplied and 1 otherwise.
    I6 = 0 if ZXYM1 is user-supplied and 1 otherwise.
    I7 = 0 if ZXY1N is user-supplied and 1 otherwise.
    I8 = 0 if ZXYMN is user-supplied and 1 otherwise.

  set ISLPSW = I1 + 2*I2 + 4*I3 +8*I4 +16*I5 +32*I6 + 64*I7 + 128*I8. Then, for example, if ISLPSW=0, then all derivative information is user-supplied and if ISLPSW=255, then all derivative information is to be internally estimated.

ZP
: An array of size M x N x 3. On output this contains partial derivatives of the surface at the given nodes. ZP is used by SURF2.

TEMP
: Scratch space.

SIGMA
: Tension factor. Values near zero result in a cubic spline; large values (e.g. 50) result in nearly a polygonal line. A typical value is 1.

IER
: An error return value. If IER is returned as 0, then no errors were detected.
    = 1 if N is less than 2.
    = 2 if X or Y values are not strictly increasing.

SURF2 - 2D interpolation for gridded data

This function interpolates a surface value at a specified coordinate using bi-splines under tension. SURF1 must be called before invoking SURF2. The desired interpolated value is returned as the value of the function.

```
-------------------------------------------------------------
                Argument  | Type     |  Mode   | Dimension
-------------------------------------------------------------
FUNCTION SURF2 (XX,       | Real     | Input   |
                YY,       | Real     | Input   |
                M,        | Integer  | Input   |
                N,        | Integer  | Input   |
                X,        | Real     | Input   | M
                Y,        | Real     | Input   | N
                Z,        | Real     | Input   | IZ x N
                IZ,       | Integer  | Input   |
                ZP,       | Real     | Input   | M x N x 3
                SIGMA)    | Real     | Input   |
-------------------------------------------------------------
```

XX
: Contains the X coordinate of a point to be mapped onto the interpolated surface.

YY
: Contains the Y coordinate of a point to be mapped onto the interpolated surface.

M
: The number of grid lines in the X direction. (M > 1)

N
: The number of grid lines in the Y direction. (N > 1)

X
: An array containing M X coordinates for grid lines in the X direction. These values must be strictly increasing.

Y
: An array containing N Y coordinates for grid lines in the Y direction. These values must be strictly increasing.

Z
: An array containing M x N functional values at the grid points; Z(I,J) contains the functional value at (X(I),Y(J)) for I=1,M and J=1,N.

IZ
: The row dimension of the matrix Z (IZ is greater than or equal to M).

ZP
: An array of size M x N x 3.

SIGMA
: Tension factor. Values near zero result in a cubic spline; large values (e.g. 50) result in nearly a polygonal line. A typical value is 1.

c_ftcurv - 1D interpolation for non-periodic functions

This function calculates an interpolatory spline under tension through a sequence of functional values.

c_ftcurv is called after all of the desired values for control parameters have been set using the procedures c_ftseti, c_ftsetr, c_ftsetc. Control parameters that apply to c_ftcurv are: sig, sl1, sln, sf1.

The value for the parameter sig specifies the tension factor. Values near zero result in a cubic spline;

large values (e.g. 50) result in nearly a polygonal line. A typical value is 1. (the default).

The values for sl1 and sln specify the slope of the curve at the first point and last point, respectively.

The value of sf1 controls whether to use the values for sl1 and sln, or compute those values internally. Specifically, sf1

= 0 if sl1 and sln are user-specified.
= 1 if sl1 is user-specified, but sln is internally calculated.
= 2 if sln is user-specified, but sl1 is internally calculated.
= 3 if sl1 and sln are internally calculated.

By default the slopes at the end points are computed internally.

Function prototype:

```
int c_ftcurv (int, float [], float [], int, float [], float []);
```

Return value:

c_ftcurv returns an error value as per:

= 0 -- no error.
= 1 -- if n is less than 2.
= 2 -- if X values are not strictly increasing.

Argument description:

```
-------------------------------------------------
              Argument | Type       |  Size
-------------------------------------------------
int c_ftcurv (n,       | int        |
              xi,      | float []   | n
              yi,      | float []   | n
              m,       | int        |
              xo,      | float []   | m
              yo       | float []   | m
             );
-------------------------------------------------
```

n
: The number of input data points. (n > 1)

xi
: An array containing the abscissae for the input function.

yi
: An array containing the functional values of the input function -- yi[k] is the functional value at xi[k] for k=0,n-1.

m
: The number of desired interpolated values.

xo
: An array containing the abscissae for the interpolated values.

yo
: An array containing the interpolated functional values -- yo[k] is the functional value at xo[k] for k=0,n.

c_ftcurvd - calculate derivatives

This function calculates the derivatives of an interpolatory spline under tension.

c_ftcurvd is called after all of the desired values for control parameters have been set using the procedures c_ftseti, c_ftsetr, c_ftsetc. Control parameters that apply to c_ftcurvd are: sig, sl1, sln, sf1.

The value for the parameter sig specifies the tension factor. Values near zero result in a cubic spline; large values (e.g. 50) result in nearly a polygonal line. A typical value is 1. (the default).

The values for sl1 and sln specify the slope of the curve at the first point and last point, respectively.

The value of sf1 controls whether to use the values for sl1 and sln, or compute those values internally. Specifically, sf1

> = 0 if sl1 and sln are user-specified.
> = 1 if sl1 is user-specified, but sln is internally calculated.
> = 2 if sln is user-specified, but sl1 is internally calculated.
> = 3 if sl1 and sln are internally calculated.

By default the slopes at the end points are computed internally.

Function prototype:

```
int c_ftcurvd (int, float [], float [], int, float [], float []);
```

Return value:

c_ftcurvd returns an error value as per:

> = 0 -- no error.
> = 1 -- if N is less than 2.
> = 2 -- if X values are not strictly increasing.

Argument description:

```
-------------------------------------------------
               Argument | Type       |  Size
-------------------------------------------------
int c_ftcurvd (n,       | int        |
               xi,      | float []   | n
               yi,      | float []   | n
               m,       | int        |
               xo,      | float []   | m
               yo       | float []   | m
```

```
              );
-------------------------------------------------
```

n
: The number of input data points. (n > 1)

xi
: An array containing the abscissae for the input function.

yi
: An array containing the functional values (y[k] is the functional value at x[k] for k=0,n).

m
: The number of desired derivatives.

xo
: An array containing the abscissae for the output points.

yo
: An array containing the interpolated derivative values -- yo[k] is the functional derivative at xo[k] for k=0,n.

c_ftcurvi - calculate integrals

This function calculates integrals of an interpolatory spline under tension between two user-specified limits.

c_ftcurvi is called after all of the desired values for control parameters have been set using the procedures c_ftseti, c_ftsetr, c_ftsetc. Control parameters that apply to c_ftcurvd are: sig, sl1, sln, sf1.

The value for the parameter sig specifies the tension factor. Values near zero result in a cubic spline; large values (e.g. 50) result in nearly a polygonal line. A typical value is 1. (the default).

The values for sl1 and sln specify the slope of the curve at the first point and last point, respectively.

The value of sf1 controls whether to use the values for sl1 and sln, or compute those values internally. Specifically, sf1

> = 0 if sl1 and sln are user-specified.
> = 1 if sl1 is user-specified, but sln is internally calculated.
> = 2 if sln is user-specified, but sl1 is internally calculated.
> = 3 if sl1 and sln are internally calculated.

By default the slopes at the end points are computed internally.

Function prototype:

```
int c_ftcurvi (float, float, int, float [], float [], float *);
```

Return value:

c_ftcurvi returns an error value as per:

= 0 -- no error.
= 1 -- if n is less than 2.
= 2 -- if X values are not strictly increasing.

Argument description:

```
-------------------------------------------------
               Argument | Type       |  Size
-------------------------------------------------
int c_ftcurvi  (xl,      | float      |
               xr,       | float      |
               n,        | int        |
               xi,       | float []   | n
               yi,       | float []   | n
               integral  | float *    |
              );
-------------------------------------------------
```

xl
: The lower limit of the integration.

xr
: The upper limit of the integration.

n
: The number of input data points. (N > 1)

xi
: An array containing the abscissae for the input function.

yi
: An array containing the functional values of the input function (y[k] is the functional value at x[k] for k=0,n).

integral
: The integral of the function from xl to xr is given by *integral.

c_ftcurvp - 1D interpolation for periodic functions

This function calculates an interpolatory spline under tension through a sequence of functional values for a periodic function.

c_ftcurvp is called after all of the desired values for control parameters have been set using the procedures c_ftseti, c_ftsetr, c_ftsetc. The only control parameter that applies to c_ftcurvp is: sig.

The value for the parameter sig specifies the tension factor. Values near zero result in a cubic spline; large values (e.g. 50) result in nearly a polygonal line. A typical value is 1. (the default).

Function prototype:

```
int c_ftcurvp (int, float [], float [], float, int, float [], float []);
```

Return value:

c_ftcurvp returns an error value as per:

= 0 -- no error.
= 1 -- if n is less than 2.
= 2 -- if the period is strictly less than the span of the abscissae.

Argument description:

```
-------------------------------------------------
                Argument | Type       |  Size
-------------------------------------------------
int c_ftcurvp  (n,       | int        |
                xi,      | float []   | n
                yi,      | float []   | n
                p,       | float      |
                m,       | int        |
                xo,      | float []   | m
                yo       | float []   | m
               );
-------------------------------------------------
```

n
: The number of input data points. (n > 1)

xi
: An array containing the abscissae for the input function.

yi
: An array containing the input functional values (y[k] is the functional value at x[k] for k=0,n).

p
: The period of the function; p must not be less than xi[n-1] - xi[0].

m
: The number of desired interpolated points.

xo
: An array containing the abscissae for the interpolated values.

yo
: An array containing the interpolated functional values (yo[k] is the functional value at xo[k] for k=0,n).

c_ftcurvpi - calculate integrals for periodic functions

This function calculates an integral between two specified points.

c_ftcurvpi is called after all of the desired values for control parameters have been set using the procedures c_ftseti, c_ftsetr, c_ftsetc. The only control parameter that applies to c_ftcurvpi is: sig.

The value for the parameter sig specifies the tension factor. Values near zero result in a cubic spline; large values (e.g. 50) result in nearly a polygonal line. A typical value is 1. (the default).

Function prototype:

```
int c_ftcurvpi (float, float, float, int, float [], float [], float *);
```

Return value:

c_ftcurpi returns an error value as per:

= 0 -- no error.
= 1 -- if n is less than 2.
= 2 -- if the period is strictly less than the span of the abscissae.

Argument description:

```
-------------------------------------------------
                Argument | Type       |  Size
-------------------------------------------------
int c_ftcurvpi (xl,      | float      |
                xr,      | float      |
                p,       | float      |
                m,       | int        |
                xi,      | float []   | m
                yi,      | float []   | m
                integral | float *    |
               );
-------------------------------------------------
```

xl
: The lower limit of the integration.

xr
: The upper limit of the integration.

p
: The period of the function; p must not be less than xi[n-1] - xi[0].

m
: The number of input points.

xi
: An array containing the abscissae for the input points.

yi
: An array containing the values of the input points (yi[k] is the functional value at xi[k] for k=0,n).

integral
: The desired integral value is returned in *integral.

c_ftcurvs - smoothing spline

This function computes an interpolatory smoothing spline under tension through a sequence of functional values.

Two parameters and one function argument used to control the degree of smoothness -- the parameters are smt, and eps and the function argument is d.

The argument *d* is a value indicating the degree of confidence in the accuracy of the input function values -- it should be an approximation of the standard deviation of error. Effectively the value of *d* controls how close the smoothed curve comes to the input data points. If *d* is small, then the interpolated curve will pass close to the input data. The larger the value of *d*, the more freedom the smooth curve has in how close it comes to the input data values.

The parameter *smt* is a more subtle global smoothing parameter; *smt* must be non-negative. For small values of *smt*, the curve approximates the tension spline and for larger values of *smt*, the curve is smoother. A reasonable value for *smt* is (float) n.

The parameter *eps* controls the precision to which *smt* is interpreted; *eps* must be between 0. and 1. inclusive. A reasonable value for *eps* is sqrt(2./(float) n).

c_ftcurvs is called after all of the desired values for control parameters have been set using the procedures c_ftseti, c_ftsetr, c_ftsetc. Control parameters that apply to c_ftcurvs are: sig, smt, eps, sf2.

The value for the parameter sig specifies the tension factor. Values near zero result in a cubic spline; large values (e.g. 50) result in nearly a polygonal line. A typical value is 1. (the default).

Function prototype:

```
   int c_ftcurvs (int, float [], float [], int, float [], int,
                  float [], float []);
```

Return value:

c_ftcurvs returns an error value as per:

- = 1 if N is less than 2.
- = 2 if smt is negative.
- = 3 if eps is negative or greater than 1.
- = 4 if X values are not strictly increasing.
- = 5 if d is negative.

Argument description:

```
--------------------------------------------------------------
               Argument | Type       |  size
--------------------------------------------------------------
int c_ftcurvs  (n,      | int        |
                xi,     | float []   | n
                yi,     | float []   | n
                dflg,   | int        |
                d,      | float []   | n
                m,      | int        |
                xo,     | float []   | m
                yo      | float []   | m
               );
--------------------------------------------------------------
```

n

The number of input data points. (n > 1)

xi

An array containing the abscissae for the input function.

yi

An array containing the functional values of the input function (yi[k] is the functional value at x[k] for k=0,n-1).

dflg

A switch for interpreting the value of d (as described below). If dflg=0, then d is an array of length n (an error estimate for each input data value); if dflg=1, then d is a scalar that serves as an error estimate for every single data item.

d

A user-specified value containing the observed weights. d may either be an array or a scalar, depending on the value of dflg.

m

The number of output values.

xo

Contains the abscissae for the output values.

yo

Contains the functional values for the smoothing spline ((yo[k] is the functional value at xo[k] for k=0,n-1).

c_ftcurvps - smoothing spline for periodic functions

This function computes an interpolatory smoothing spline under tension through a sequence of functional values for a periodic function.

Two parameters and one function argument used to control the degree of smoothness -- the parameters are smt, and eps and the function argument is d.

The argument *d* is a value indicating the degree of confidence in the accuracy of the input function values -- it should be an approximation of the standard deviation of error. Effectively the value of *d* controls how close the smoothed curve comes to the input data points. If *d* is small, then the interpolated curve will pass close to the input data. The larger the value of *d*, the more freedom the smooth curve has in how close it must come to the input data values.

The parameter *smt* is a more subtle global smoothing parameter; *smt* must be positive. For small values of *smt*, the curve approximates the tension spline and for larger values of *smt*, the curve is smoother. A reasonable value for *smt* is (float) n.

The parameter *eps* controls the precision to which *smt* is interpreted; *eps* must be between 0. and 1. inclusive. A reasonable value for *eps* sqrt(2./(float) n).

c_ftcurvps is called after all of the desired values for control parameters have been set using the procedures c_ftseti, c_ftsetr, c_ftsetc. Control parameters that apply to c_ftcurvps are: sig, smt, eps, sf2.

The value for the parameter sig specifies the tension factor. Values near zero result in a cubic spline; large values (e.g. 50) result in nearly a polygonal line. A typical value is 1. (the default).

Function prototype:

```
int c_ftcurvps (int, float [], float [], float, int, float [], int,
```

```
                  float [], float []);
```

Return value:

c_ftcurvps returns an error value as per:

= 1 if N is less than 2.
= 2 if smt is negative.
= 3 if eps is negative or greater than 1.
= 4 if X values are not strictly increasing.
= 5 if d is negative.

Argument description:

```
---------------------------------------------------------------
                Argument | Type      |  size
---------------------------------------------------------------
int c_ftcurvps (n,       | int       |
                xi,      | float []  | n
                yi,      | float []  | n
                p,       | float     |
                dflg,    | int       |
                d,       | float []  | n
                m,       | m         |
                xo,      | float []  | m
                yo       | float []  | m
               );
---------------------------------------------------------------
```

n
: The number of input data values. (n > 1)

xi
: An array containing the abscissae for the input function.

yi
: An array containing the functional values of the input function (yi[k] is the functional value at x[k] for k=0,n-1).

p
: The period of the function.

dflg
: A switch for interpreting the value of the argument d (described below). If dflg=0, then d is an array of length n (an error estimate for each input data value); if dflg=1, then d is a scalar that serves as an error estimate for every single data item.

d
: A user-specified value containing the observed weights. d may either be an array or a scalar, depending on the value of dflg.

m
: The number of output values.

xo
: Contains the abscissae for the output values.

yo
: Contains the functional values for the smoothing spline ((yo[k] is the functional value at xo[k] for k=0,m-1).

c_ftkurv - interpolation for parametric curves

This function calculates an interpolatory spline under tension through a sequence of points in the plane.

Given a sequence of input points ((x[0],y[0]), ... , (x[n-1],y[n-1]), the interpolated curve is parameterized by mapping points in the interval [0.,1.] onto the interpolated curve. The resulting curve has a parametric representation both of whose components are splines under tension and functions of the polygonal arc length. The value 0. is mapped onto (x[0],y[0]) and the value 1. is mapped onto (x[n-1],y[n-1]).

c_ftkurv is called after all of the desired values for control parameters have been set using the procedures c_ftseti, c_ftsetr, c_ftsetc. Control parameters that apply to c_ftkurv are: sig, sl1, sln, sf1.

The value for the parameter sig specifies the tension factor. Values near zero result in a cubic spline; large values (e.g. 50) result in nearly a polygonal line. A typical value is 1. (the default).

The value for parameter sl1 is in radians and contains the slope at (x[0],y[0]). The angle is measured counter-clockwise from the X axis and the positive sense of the curve is assumed to be that moving from point 0 to point n-1. A value for sl1 may be omitted as indicated by the switch sf1.

The value for parameter sln is in radians and contains the slope at (x[n-1],y[n-1]). The angle is measured counter-clockwise from the X axis and the positive sense of the curve is assumed to be that moving from point 0 to point n-1. A value for sln may be omitted as indicated by the switch sf1.

The value of sf1 controls whether to use the values for sl1 and sln, or compute those values internally. Specifically, sf1

= 0 if sl1 and sln are user-specified.
= 1 if sl1 is user-specified, but sln is internally calculated.
= 2 if sln is user-specified, but sl1 is internally calculated.
= 3 if sl1 and sln are internally calculated.

By default the slopes at the end points are computed internally.

Function prototype:

```
int c_ftkurv (int, float [], float [], int, float [], float [], float []);
```

Return value:

c_ftkurv returns an error value as per:

= 0 -- no error.
= 1 -- if n is less than 2.
= 2 -- if adjacent coordinate pairs coincide.

Argument description:

```
--------------------------------------------------
              Argument | Type      |  Size
--------------------------------------------------
int c_ftkurv  (n,      | int       |
              xi,      | float []  | n
              yi,      | float []  | n
              m,       | int       |
              t,       | float []  | m
              xo,      | float []  | m
              yo       | float []  | m
             );
--------------------------------------------------
```

n
: The number of input data values. (n > 1)

xi
: An array containing the abscissae for the input function.

yi
: An array containing the functional values (y[k] is the functional value at x[k] for k=0,n-1).

m
: The number of desired interpolated values.

t
: Contains an array of values for the parameter mapping onto the interpolated curve.

xo
: An array containing the X values for the interpolated points. t[k] maps to (xo[k],yo[k]) for k=0,m-1.

yo
: An array containing the Y values for the interpolated points. t[k] maps to (xo[k],yo[k]) for k=0,m-1.

c_ftkurvp - interpolation for closed parametric curves

This procedure calculates an interpolatory spline under tension through a sequence of points in the plane forming a closed curve.

Given a sequence of distinct input points ((x[0],y[0]), ... , (x[n-1],y[n-1]), the interpolated curve is parameterized by mapping points in the interval [0.,1.] onto the interpolated curve. The resulting curve has a parametric representation both of whose components are splines under tension and functions of the polygonal arc length. The value 0. is mapped onto (x[0],y[0]) and the value 1. is mapped onto (x[0],y[0]) as well (completing the closed curve).

c_ftkurvp is called after all of the desired values for control parameters have been set using the procedures c_ftseti, c_ftsetr, c_ftsetc. The only control parameter that applies to c_ftkurvp is: sig.

The value for the parameter sig specifies the tension factor. Values near zero result in a cubic spline; large values (e.g. 50) result in nearly a polygonal line. A typical value is 1. (the default).

Function prototype:

```
int c_ftkurvp (int, float [], float [], int, float [], float [], float []);
```

Return value:

c_ftkurvp returns an error value as per:

> = 0 -- no error.
> = 1 -- if n is less than 2.
> = 2 -- if adjacent coordinate pairs coincide.

Argument description:

```
-------------------------------------------------
               Argument | Type      |  Size
-------------------------------------------------
int c_ftkurv  (n,       | int       |
               xi,      | float []  | n
               yi,      | float []  | n
               m,       | int       |
               t,       | float []  | m
               xo,      | float []  | m
               yo       | float []  | m
              );
-------------------------------------------------
```

n
: The number of input data points. (n > 1)

xi
: An array containing the abscissae for the input function.

yi
: An array containing the functional values (y[k] is the functional value at x[k] for k=0,n-1).

m
: The number of desired interpolated points.

t
: Contains an array of values for the parameter mapping onto the interpolated curve. Any interval [tt,tt+1.] maps onto the entire curve.

xo
: An array containing the X values for the interpolated points. t[k] maps to (xo[k],yo[k]) for k=0,n-1.

yo
: An array containing the Y values for the interpolated points.

c_ftkurvd - calculate interpolated values and derivatives for parametric curves

---

This procedure behaves like ftkurv except that in addition it returns the first and second derivatives of the component functions in the parameterization.

Given a sequence of input points ((x[0],y[0]), ... , (x[n-1],y[n-1]), the interpolated curve is parameterized by mapping points in the interval [0.,1.] onto the interpolated curve. The resulting curve has a parametric representation both of whose components are splines under tension and functions of the polygonal arc length. The value 0. is mapped onto (x[0],y[0]) and the value 1. is mapped onto (x[n-1],y[n-1]).

c_ftkurvd is called after all of the desired values for control parameters have been set using the procedures c_ftseti, c_ftsetr, c_ftsetc. Control parameters that apply to c_ftkurvd are: sig, sl1, sln, sf1.

The value for the parameter sig specifies the tension factor. Values near zero result in a cubic spline; large values (e.g. 50) result in nearly a polygonal line. A typical value is 1. (the default).

The value for parameter sl1 is in radians and contains the slope at (x[0],y[0]). The angle is measured counter-clockwise from the X axis and the positive sense of the curve is assumed to be that moving from point 0 to point n-1. A value for sl1 may be omitted as indicated by the switch sf1.

The value for parameter sln is in radians and contains the slope at (x[n-1],y[n-1]). The angle is measured counter-clockwise from the X axis and the positive sense of the curve is assumed to be that moving from point 0 to point n-1. A value for sln may be omitted as indicated by the switch sf1.

The value of sf1 controls whether to use the values for sl1 and sln, or compute those values internally. Specifically, sf1

> = 0 if sl1 and sln are user-specified.
> = 1 if sl1 is user-specified, but sln is internally calculated.
> = 2 if sln is user-specified, but sln is internally calculated.
> = 3 if sl1 and sln are internally calculated.

By default the slopes at the end points are computed internally.

Function prototype:

```
  int c_ftkurvd (int, float [], float [], int, float [], float [], float [],
                float [], float [], float [], float []);
```

Return value:

c_ftkurvd returns an error value as per:

> = 0 -- no error.
> = 1 -- if n is less than 2.
> = 2 -- if adjacent coordinate pairs coincide.

Argument description:

```
-------------------------------------------------
               Argument  | Type       |  Size
-------------------------------------------------
int c_ftkurvd (n,        | int        |
               xi,       | float []   | n
```

```
               yi,        | float [] | n
               m,         | int      |
               t,         | float [] | m
               xo,        | float [] | m
               yo,        | float [] | m
               xd,        | float [] | m
               yd,        | float [] | m
               xdd,       | float [] | m
               ydd        | float [] | m
              );
------------------------------------------------
```

n
: The number of input data points. (n > 1)

xi
: An array containing the abscissae for the input function.

yi
: An array containing the functional values (y[k] is the functional value at x[k] for k=0,n).

m
: The number of desired interpolated points.

t
: Contains an array of values for the parameter mapping onto the interpolated curve.

xo
: An array containing the X values for the interpolated points. t[k] maps to (xo[k],yo[k]) for k=0,n-1.

yo
: An array containing the Y values for the interpolated points.

xd
: Contains the first derivatives of the X component with respect to t.

yd
: Contains the first derivatives of the Y component with respect to t.

xdd
: Contains the second derivatives of the X component with respect to t.

ydd
: Contains the second derivatives of the Y component with respect to t.

c_ftkurvpd - interpolation and derivative calculation for closed parametric curves

---

This procedure behaves like ftkurvp except that in addition it returns the first and second derivatives of the component functions in the parameterization.

Given a sequence of distinct input points ((x[0],y[0]), ... , (x[n-1],y[n-1]), the interpolated curve is parameterized by mapping points in the interval [0.,1.] onto the interpolated curve. The resulting curve has a parametric representation both of whose components are splines under tension and functions of the polygonal arc length. The value 0. is mapped onto (x[0],y[0]) and the value 1. is mapped onto (x[0],y[0]) as well (completing the closed curve).

c_ftkurvpd is called after all of the desired values for control parameters have been set using the procedures c_ftseti, c_ftsetr, c_ftsetc. The only control parameter that applies to c_ftkurvpd is: sig.

The value for the parameter sig specifies the tension factor. Values near zero result in a cubic spline; large values (e.g. 50) result in nearly a polygonal line. A typical value is 1. (the default).

Function prototype:

```
int c_ftkurvpd (int, float [], float [], int, float [], float [], float [],
                float [], float [], float [], float []);
```

Return value:

c_ftkurvpd returns an error value as per:

= 0 -- no error.
= 1 -- if n is less than 2.
= 2 -- if adjacent coordinate pairs coincide.

Argument description:

```
-------------------------------------------------
                Argument | Type      |  Size
-------------------------------------------------
int c_ftkurvpd (n,       | int       |
                xi,      | float []  | n
                yi,      | float []  | n
                m,       | int       |
                t,       | float []  | m
                xo,      | float []  | m
                yo       | float []  | m
                xd,      | float []  | m
                yd       | float []  | m
                xdd,     | float []  | m
                ydd      | float []  | m
               );
-------------------------------------------------
```

n
: The number of input data values. (n > 1)

xi
: An array containing the abscissae for the input function.

yi
: An array containing the functional values (y[k] is the functional value at x[k] for k=0,n-1).

m
: The number of desired interpolated points.

t
: Contains an array of values for the parameter mapping onto the interpolated curve. Any interval [tt,tt+1.] maps onto the entire curve.

xo
: An array containing the X values for the interpolated points. t[k] maps to (xo[k],yo[k]) for k=0,n-1.

yo

An array containing the Y values for the interpolated points.

xd
: Contains the first derivatives of the X component with respect to t.

yd
: Contains the first derivatives of the Y component with respect to t.

xdd
: Contains the second derivatives of the X component with respect to t.

ydd
: Contains the second derivatives of the Y component with respect to t.

c_ftsurf - 2D tension spline interpolation of rectangular data

---

This procedure calculates an interpolatory surface passing through a rectangular grid of function values. The surface computed is a tensor product of splines under tension.

c_ftsurf is called after all of the desired values for control parameters have been set using the procedures c_ftseti, c_ftsetr, c_ftsetc, c_ftsetfa. The control parameters that apply to c_ftsurf are: sig, zx1, zxm, zy1, zyn, z11, zm1, z1n, zmn, df1, df2, df3, df4, df5, df6, df7, df8

The value for the parameter sig specifies the tension factor. Values near zero result in a cubic spline; large values (e.g. 50) result in nearly a polygonal line. A typical value is 1. (the default).

zx1 is an array containing ni X-partial derivatives of the function along the line xi[0], that is zx1[j] is the X-partial derivative at point (x[0],y[j]) for j=0,ni-1. This parameter may be defaulted by setting the value for df1 appropriately. The default is to compute zx1 internally. Values for zx1 can be set using the procedure c_ftsetfa.

zxm is an array containing ni X-partial derivatives of the function along the line xi[mi-1], that is zxm[j] is the X-partial derivative at point (xi[mi-1],yi[j]) for j=0,ni-1. This parameter may be defaulted by setting the value for df2 appropriately. The default is to compute zx2 internally. Values for zxm can be set using the procedure c_ftsetfa.

zy1 is an array containing mi Y-partial derivatives of the function along the line yi[0], that is zy1[j] is the Y-partial derivative at point (x[i],y[0]) for i=0,mi-1. This parameter may be defaulted by setting the value for df3 appropriately. The default is to compute zy1 internally. Values for zy1 can be set using the procedure c_ftsetfa.

zyn is an array containing mi Y-partial derivatives of the function along the line yi[ni-1], that is zyn[j] is the Y-partial derivative at point (x[i],y[ni-1]) for i=0,mi-1. This parameter may be defaulted by setting the value for df4 appropriately. The default is to compute zyn internally. Values for zyn can be set using the procedure c_ftsetfa.

z11, zm1, z1n, zmn specify X-Y-partial derivatives of the function at the four corners (xi[0],yi[0]), (xi[mi-1],yi[0]), (xi[0],yi[ni-1]), (xi[mi-1],yi[ni-1]), These parameters may be defaulted by setting the values for df5, df6, df7, df8, appropriately. The default is to compute z11, zm1, z1n, zmn internally.

Function prototype:

```
float *c_ftsurf (int, int, float *, float *, float *,
                 int, int, float *, float *, int *);
```

Return value:

c_ftsurf returns a pointer to an array containing mo x no floats which are the interpolated values on the grid specified by the arrays xo and yo.

Argument description:

```
---------------------------------------------------------------
                 Argument  | Type       |  Size
---------------------------------------------------------------
float *c_ftsurf (mi,       | int        |
                 ni,       | int        |
                 xi,       | float *    | pointer to m floats
                 yi,       | float *    | pointer to n floats
                 zi,       | float *    | pointer to m * n floats
                 mo,       | int        |
                 no,       | int        |
                 xo,       | float *    | pointer to mo floats
                 yo,       | float *    | pointer to no floats
                 ier       | int *      |
                );
---------------------------------------------------------------
```

mi

The number of grid lines in the X direction. (mi > 1)

ni

The number of grid lines in the Y direction. (ni > 1)

xi

Pointer to an array containing mi X coordinates for grid lines in the X direction.

yi

Pointer to an array containing ni Y coordinates for grid lines in the Y direction.

zi

Pointer to mi x ni floats which are the functional values at the grid points defined by xi and yi.

mo

The number of output values in the X direction.

no

The number of output values in the Y direction.

xo

Pointer to an array containing mo X coordinates for grid lines in the X direction in the output interpolation grid.

xo

Pointer to an array containing no Y coordinates for grid lines in the Y direction in the output interpolation grid.

ier

The value *ier is an error flag as per:
- = 0 -- no error.
- = 1 -- if n is less than 2 or m is less than 2.
- = 2 -- if X or Y values are not strictly increasing.

c_ftseti - Set int valued parameters

c_ftseti is used to set values for any of the control parameters that take int values. The values set by c_ftseti remain in effect until changed by subsequent calls to c_ftseti.

Function prototype:

```
void c_ftseti(char *, int);
```

Argument description:

```
-------------------------------------------
              Argument | Type      |  Size
-------------------------------------------
void c_ftseti  (pnam,  | char *    |
               ival);  | int       |
-------------------------------------------
```

pnam
: The name of the control parameter to be assigned an int value.

ival
: The value to be assigned to the control parameter whose name is pointed to by pnam.

c_ftgeti - Retrieve an int valued parameter

c_ftgeti is a called to obtain current values for any of the int valued control parameters.

Function prototype:

```
void c_ftgeti(char *, int *);
```

Argument description:

```
-------------------------------------------
              Argument | Type      |  Size
-------------------------------------------
void c_ftgeti  (pnam,  | char *    |
               ival);  | int  *    |
-------------------------------------------
```

pnam
: The name of the control parameter whose value is to be retrieved.

ival
: *ival will be the value currently assigned to the control parameter whose name is pointed to by pnam.

c_ftsetr - Set float valued parameters

c_ftsetr is used to set values for any of the control parameters that take float values. The values set by c_ftsetr remain in effect until changed by subsequent calls to c_ftsetr.

Function prototype:

```
  void c_ftsetr(char *, float);
```

Argument description:

```
-------------------------------------------
              Argument | Type      |  Size
-------------------------------------------
void c_ftsetr (pnam,   | char *    |
               fval);  | float     |
-------------------------------------------
```

pnam
: The name of the control parameter to be assigned a float value.

fval
: The value to be assigned to the control parameter whose name is pointed to by pnam.

c_ftgetr - Retrieve a float valued parameter

c_ftgetr is a called to obtain current values for any of the float valued control parameters.

Function prototype:

```
  void c_ftgetr(char *, float *);
```

Argument description:

```
-------------------------------------------
              Argument | Type      |  Size
-------------------------------------------
void c_ftgetr (pnam,   | char *    |
               fval);  | float *   |
-------------------------------------------
```

pnam
: The name of the control parameter whose value is to be retrieved.

fval
: *fval will be the value currently assigned to the control parameter whose name is pointed to by pnam.

c_ftsetc - Set char valued parameters

c_ftsetc is used to set values for any of the control parameters that take string values. The values set by c_ftsetc remain in effect until changed by subsequent calls to c_ftsetc.

Function prototype:

```
  void c_ftsetc(char *, char *);
```

Argument description:

```
-------------------------------------------
              Argument | Type      |  Size
-------------------------------------------
void c_ftsetc (pnam,   | char *    |
               cval);  | char *    |
-------------------------------------------
```

pnam
: The name of the control parameter to be assigned a char value.

cval
: The value to be assigned to the control parameter whose name is pointed to by pnam.

c_ftgetc - Retrieve a char valued parameter

---

c_ftgetc is a called to obtain current values for any of the string valued control parameters.

Function prototype:

```
  void c_ftgetc(char *, char *);
```

Argument description:

```
-------------------------------------------
              Argument | Type      |  Size
-------------------------------------------
void c_ftgetc (pnam,   | char *    |
               cval);  | char *    |
-------------------------------------------
```

pnam
: The name of the control parameter whose value is to be retrieved.

cval
: cval will point to a string containing the returned value. The user is required to reserve enough space to store the string.

c_ftsetfa - Set float valued array parameters

---

c_ftsetra is used to set values for any of the control parameters that take floating arrays. The values set by c_ftsetfa remain in effect until changed by subsequent calls to c_ftsetfa

Function prototype:

```
  int c_ftsetfa(char *pnam, int n, float *far);
```

Argument description:

```
-------------------------------------------
              Argument | Type     |  Size
-------------------------------------------
int c_ftsetfa (pnam,   | char *   |
               n,      | int      |
               far);   | float *  |
-------------------------------------------
```

pnam
: The name of the control parameter to be assigned a float array value.

n
: The number of float values in the array.

far
: A pointer to the n-element input array.

c_ftgetfa_size - Retrieve the size of an array

---

c_ftgetfa_size is a called to obtain current sizes for any of the float valued control parameters.

Function prototype:

```
int c_ftgetfa_size(char *);
```

Return value:

c_ftgetfa_size returns the size of the array (or returns a zero if an error occurs).

Argument description:

```
----------------------------------------------
                     Argument | Type     |  Size
----------------------------------------------
int c_ftgetfa_size (pnam    | char *   |
                       );     |          |
----------------------------------------------
```

pnam
: The name of the control parameter whose value is to be retrieved.

c_ftgetfa_data - Retrieve array values

---

c_ftgetfa_data is a called to obtain current values for any of the float valued array control parameters.

Function prototype:

```
float *c_ftgetfa_data(char *);
```

Return value:

c_ftgetfa_data returns a pointer to a array of data (or returns a -1 if an error occurs). The size of the array can be obtained by using c_ftgetfa_size.

Argument description:

```
------------------------------------------------
                      Argument | Type     |  Size
------------------------------------------------
float *c_ftgetfa_data (pnam)   | char *   |
------------------------------------------------
```

pnam
    The name of the control parameter whose value is to be retrieved.

Overview of Fitgrid procedures

The table below summarizes all of the procedures in Fitgrid for all interfaces.

Clicking on the procedure name will link to a detailed description of that procedure.

```
Fortran single precision:
------------------------

 CURV1   -  does set-up for CURV2.
 CURV2   -  1D interpolation for non-periodic functions.
 CURVD   -  derivatives for 1D functions.
 CURVI   -  integrals for 1D functions.
 CURVP1  -  does set-up for CURVP2.
 CURVP2  -  1D interpolation for periodic functions.
 CURVPI  -  integrals for periodic functions.
 CURVS   -  does smoothing and set-up for CURV2.
 CURVPS  -  does smoothing for periodic functions.
 KURV1   -  does set-up for KURV2.
 KURV2   -  interpolation for parametric curves.
 KURVD   -  derivatives for parametric curves.
 KURVP1  -  set-up for KURVP2.
 KURVP2  -  interpolation for closed parametric curves.
 KURVPD  -  derivatives for closed parametric curves.
 SURF1   -  set-up for SURF2.
 SURF2   -  2D interpolation for gridded data.


C single precision:
------------------

 c_ftcurv    -  1D interpolation for non-periodic functions.
 c_ftcurvd   -  derivatives for 1D functions.
 c_ftcurvi   -  integrals for 1D functions.
 c_ftcurvp   -  1D interpolation for periodic functions.
 c_ftcurvpi  -  integrals for periodic functions.
```

```
 c_ftcurvs   -  does smoothing for non-periodic functions.
 c_ftcurvps  -  does smoothing for periodic functions.
 c_ftkurv    -  interpolation for parametric curves.
 c_ftkurvp   -  interpolation for closed parametric curves.
 c_ftkurvd   -  derivatives for parametric curves.
 c_ftkurvpd  -  interpolation for closed parametric curves.
 c_ftsurf    -  2D interpolation for rectangular data.
 c_ftseti    -  set values for int parameters.
 c_ftgeti    -  get values for int parameters.
 c_ftsetr    -  set values for float parameters.
 c_ftgetr    -  get values for float parameters.
 c_ftsetc    -  set values for string parameters.
 c_ftgetc    -  get values for string parameters.
 c_ftsetfa   -  set values for float arrays.
 c_ftgetfa_size  -  get array size of float array.
 c_ftgetfa_data  -  get float array data.


NCL
---

 ftsetp    -  set parameter values.
 ftgetp    -  get parameter values.
 ftcurv    -  1D interpolation for non-periodic functions.
 ftcurvd   -  derivatives for 1D functions.
 ftcurvi   -  integrals for 1D functions.
 ftcurvp   -  1D interpolation for periodic functions.
 ftcurvpi  -  integrals for periodic functions.
 ftcurvs   -  does smoothing for non-periodic functions.
 ftcurvps  -  does smoothing for periodic functions.
 ftkurv    -  interpolation for parametric curves.
 ftkurvp   -  interpolation for closed parametric curves.
 ftkurvd   -  derivatives for parametric curves.
 ftkurvpd  -  interpolation for closed parametric curves.
 ftsurf    -  2D interpolation for rectangular data.
```

Fitgrid Examples

Overview

This module contains source codes for Fitgrid examples.

The following list summarizes the examples:

- Example 1 -- interpolated function, derivative and integral
- Example 2 -- interpolated periodic function and integral
- Example 3 -- interpolated smoothing functions
- Example 4 -- interpolated parametric curves and derivatives
- Example 5 -- interpolated closed parametric curves and derivatives

- Example 6 -- interpolated surface

All of the example codes isolate the drawing of graphics into single calls to procedures for drawing plots for functions or surfaces. If you are not using NCAR Graphics and want to draw the plots, you will have to substitute appropriate calls from the graphics package you are using for the plotting procedures in the examples.

Of course you will need to be using NCAR Graphics software if you are using NCL.

Creating your own executable

If you have access to NCAR Graphics software, you can create executables for all of the Fortran or C examples in one of two ways. The easiest is to use the ncargex utility which is documented with a man page. The other way is to use either *ncargf77* (for Fortran) or *ncargcc* (for C) to compile and execute the source codes.

Use ng4ex to generate the NCL examples.

If you are using NCAR Graphics and the C interface to Fitgrid and you are compiling using *ncargcc*, then function prototypes for all Fitgrid C functions are contained in a C header file. This can be accessed by inserting the line:

```
#include <ncarg/ngmath.h>
```

into your code.

Example 1 -- interpolated function, derivative and integral

- Fortran single precision
- C single precision
- NCL
- Sample plot

Example 2 -- interpolated periodic function and derivative

- Fortran single precision
- C single precision
- NCL
- Sample plot

Example 3 -- interpolated smoothing functions

- Fortran single precision
- C single precision
- NCL
- Sample plot

Example 4 -- interpolated parametric curves and derivatives

- Fortran single precision
- C single precision
- NCL
- Sample plot

Example 5 -- interpolated closed parametric curves and derivatives

- Fortran single precision
- C single precision
- NCL
- Sample plot

Example 6 -- interpolated surface

- Fortran single precision
- C single precision
- NCL
- Sample plot

Example 1 - interpolated function, derivative and integral

```
      PROGRAM FTEX01
C
C  Example of CURV1, CURV2, CURVD, CURVI.
C
C  Define dimensions, declare arrays.
C
      PARAMETER (IDIM=11,IOUT=201)
      DIMENSION X(IDIM),Y(IDIM),YP(IDIM),TEMP(IDIM,2)
      DIMENSION XO(IOUT),YO(IOUT),YD(IOUT),YI(IOUT)
C
C Specify the input data.
C
C
      DATA X/  0.00,   2.00,   5.00,   8.00,  10.00,  13.00,
     +        15.00,  18.00,  21.00,  23.00,  30.00         /
      DATA Y/  1.00,   0.81,   0.00,  -0.81,  -1.00,  -0.84,
     +        -0.56,   0.04,   0.73,   1.18,   2.0          /
C
C  Call CURV1 setup, specifying that the derivatives should be
```

```
C  zero at the end points.
C
      SLP1   = 0.
      SLPN   = 0.
      ISLPSW = 0
      SIGMA  = 1.
      CALL CURV1(IDIM, X, Y, SLP1, SLPN, ISLPSW, YP, TEMP, SIGMA, IERR)
C
C  Call CURV2 and calculate the interpolated values, the derivatives,
C  and the integrals.
C
      XINC = 30./(IOUT-1)
      DO 10 I=1,IOUT
        XO(I) = (I-1)*XINC
        YO(I) = CURV2(XO(I),IDIM,X,Y,YP,SIGMA)
        YD(I) = CURVD(XO(I),IDIM,X,Y,YP,SIGMA)
        YI(I) = CURVI(XO(1),XO(I),IDIM,X,Y,YP,SIGMA)
   10 CONTINUE
C
C  Draw a plot of the interpolated functions and mark the original points.
C
      CALL DRWFT1(IDIM,X,Y,IOUT,XO,YO,YD,YI)
C
      STOP
      END
      SUBROUTINE DRWFT1(II,X,Y,IO,XO,YO,YD,YI)
C
C  Define error file, Fortran unit number, workstation type,
C  and workstation ID.
C
      PARAMETER (IERRF=6, LUNIT=2, IWTYPE=1, IWKID=1)
C
      DATA YPOS_TOP/0.88/
C
C  Open GKS, open and activate a workstation.
C
      CALL GOPKS (IERRF, ISZDM)
      CALL GOPWK (IWKID, LUNIT, IWTYPE)
      CALL GACWK (IWKID)
C
C  Define a color table.
C
      CALL GSCR(IWKID, 0, 1.0, 1.0, 1.0)
      CALL GSCR(IWKID, 1, 0.0, 0.0, 0.0)
      CALL GSCR(IWKID, 2, 1.0, 0.0, 0.0)
      CALL GSCR(IWKID, 3, 0.0, 1.0, 0.0)
      CALL GSCR(IWKID, 4, 0.0, 0.0, 1.0)
      CALL GSCLIP(0)
C
C  Plot main title.
C
      CALL PLCHHQ(0.50,0.95,':F25:Demo for CURV, CURVD, CURVI',
     +            0.030,0.,0.)
C
C  Graph the interpolated function values and mark the original
C  input data points.
C
      YB = -1.0
      YT =  2.0
      CALL BKGFT1(YPOS_TOP,'Function',YB,YT)
      CALL GRIDAL(6,5,3,1,1,1,10,0.0,YB)
```

```
C
C  Mark the original data points.
C
      CALL GSMKSC(2.)
      CALL GSPMCI(4)
      CALL GSLWSC(1.)
      CALL GPM(II,X,Y)
C
C  Graph the interpolated function values.
C
      CALL GPL(IO,XO,YO)
C
C  Graph the derivative.
C
      YB = -0.3
      YT =  0.3
      CALL BKGFT1(YPOS_TOP-0.3,'Derivative',YB,YT)
      CALL GRIDAL(6,5,3,1,1,1,10,0.0,YB)
      CALL GPL(IO,XO,YD)
      CALL GSPLCI(1)
C
C  Graph the integral.
C
      YB = -6.0
      YT = 10.0
      CALL BKGFT1(YPOS_TOP-0.6,'Integral',YB,YT)
      CALL GRIDAL(6,5,4,1,1,1,10,0.0,YB)
      CALL GPL(IO,XO,YI)
      CALL GSPLCI(1)
      CALL FRAME
C
      CALL GDAWK(IWKID)
      CALL GCLWK(IWKID)
      CALL GCLKS
C
      RETURN
      END
      SUBROUTINE BKGFT1(YPOS,LABEL,YB,YT)
      DIMENSION XX(2),YY(2)
      CHARACTER*(*) LABEL
C
      CALL SET(0.,1.,0.,1.,0.,1.,0.,1.,1)
      CALL PCSETI('FN',21)
      CALL PLCHHQ(0.20,YPOS - 0.03,LABEL,0.025,0.,-1.)
      CALL SET(0.13,0.93,YPOS-0.2,YPOS,0.0,30.0, YB, YT, 1)
      XX(1) = 0.
      XX(2) = 30.
      YY(1) = 0.
      YY(2) = 0.
      CALL GSPLCI(2)
      CALL GPL(2,XX,YY)
      CALL GSPLCI(1)

      CALL GASETI('LTY',1)
      CALL PCSETI('FN',21)
      CALL GASETR('XLS',0.02)
      CALL GASETC('XLF','(I3)')
      CALL GASETR('YLS',0.02)
      CALL GASETC('YLF','(F5.1)')
      CALL GASETR('XMJ',0.02)
      CALL GASETR('YMJ',0.02)
```

```
C
      RETURN
      END
```

Example 1 - interpolated function, derivative and integral

```
#include <stdio.h>
#include <stdio.h>
#include <ncarg/ncargC.h>
#include <ncarg/gks.h>
#include <ncarg/ngmath.h>

void c_bkgft1(float, char *, float, float);
void c_drwft1(int, float [], float [], int, float [], float [],
              float [], float []);

/*
 *  Example of curv1, curv2, curvd, curvi.
 */

#define IDIM  11
#define IOUT 201

#define IWTYPE 1
#define WKID   1

main()
{
  float x[] = { 0.00,   2.00,   5.00,   8.00,  10.00,  13.00,
               15.00,  18.00,  21.00,  23.00,  30.00};
  float y[] = { 1.00,   0.81,   0.00,  -0.81,  -1.00,  -0.84,
               -0.56,   0.04,   0.73,   1.18,   2.00};
  float xinc, xo[IOUT], yo[IOUT], yd[IOUT], yi[IOUT];
  int   i;

/*
 *  Create the output X coordinate array.
 */
  xinc =  30./(IOUT-1);
  for (i = 0; i < IOUT; i++) {
    xo[i] = xinc * i;
  }

/*
 *  Require that the derivatives of the interpolated curve are
 *  zero at the end points.
 */
  c_ftseti("sf1",   0);
  c_ftsetr("sl1", 0.0);
  c_ftsetr("sl2", 0.0);

/*
```

```
 *  Calculate the interpolated values, the derivative, and the integral.
 */
  c_ftcurv(IDIM, x, y, IOUT, xo, yo);
  c_ftcurvd(IDIM, x, y, IOUT, xo, yd);
  for (i = 0; i < IOUT; i++) {
    c_ftcurvi(0., xo[i], IDIM, x, y, yi+i);
  }

/*
 *  Draw plot.
 */
  c_drwft1(IDIM,x,y,IOUT,xo,yo,yd,yi);

}

void c_drwft1(int n, float x[], float y[], int m, float xo[], float yo[],
                  float yd[], float yi[])
{

  int   i;
  float yb, yt, ypos_top = 0.88;

  Gcolr_rep rgb;
  Gpoint plist[IDIM];
  Gpoint_list pmk;

/*
 *  Open GKS, open and activate a workstation.
 */
  gopen_gks("stdout",0);
  gopen_ws(WKID, NULL, IWTYPE);
  gactivate_ws(WKID);

/*
 * Define a color table.
 */
  rgb.rgb.red = rgb.rgb.green = rgb.rgb.blue = 1.;
  gset_colr_rep(WKID,0,&rgb);
  rgb.rgb.red = rgb.rgb.green = rgb.rgb.blue = 0.;
  gset_colr_rep(WKID,1,&rgb);
  rgb.rgb.red = 1.;
  rgb.rgb.green = rgb.rgb.blue = 0.;
  gset_colr_rep(WKID,2,&rgb);
  rgb.rgb.red = rgb.rgb.green = 0.;
  rgb.rgb.blue = 1.;
  gset_colr_rep(WKID,3,&rgb);

/*
 * Plot the main title.
 */
  gset_clip_ind(0);
  c_plchhq(.5,.95,":F25:Demo for c_ftcurv, c_ftcurvd, c_ftcurvi",0.03,0.,0.);

/*
 * Graph the interpolated function values and mark the original
 * input data points.
 */
  yb = -1.0;
  yt =  2.0;
  c_bkgft1(ypos_top,"Function",yb,yt);
  c_gridal(6,5,3,1,1,1,10,0.0,yb);
```

```
  c_curve(xo,yo,m);

/*
 *  Mark the input data points.
 */
  for (i = 0; i < n; i++) {
    plist[i].x = x[i];
    plist[i].y = y[i];
  }
  gset_marker_size(2.);
  gset_marker_colr_ind(3);
  pmk.num_points = n;
  pmk.points = plist;
  gpolymarker(&pmk);

/*
 *  Graph the derivatives.
 */
  yb = -0.3;
  yt =  0.3;
  c_bkgft1(ypos_top-0.3,"Derivative",yb,yt);
  c_gridal(6,5,3,1,1,1,10,0.0,yb);
  c_curve(xo,yd,m);

/*
 *  Graph the integral.
 */
  yb = -6.0;
  yt = 10.0;
  c_bkgft1(ypos_top-0.6,"Integral",yb,yt);
  c_gridal(6,5,4,1,1,1,10,0.0,yb);
  c_curve(xo,yi,m);

  c_frame();
/*
 *  Deactivate and close workstation, close GKS.
 */
  gdeactivate_ws(WKID);
  gclose_ws(WKID);
  gclose_gks();
}

void c_bkgft1(float ypos, char *label, float yb, float yt) {
  c_set(0.,1.,0.,1.,0.,1.,0.,1.,1);
  c_pcseti("fn",21);
  c_plchhq(.2,ypos-0.03,label,0.025,0.,-1.0);
  c_set(0.13,0.93,ypos-0.2,ypos,0.0,30.0, yb, yt, 1);
  gset_line_colr_ind(2);
  c_line(0.,0.,30.,0.);
  c_sflush();
  gset_line_colr_ind(1);
  c_gaseti("lty",1);
  c_pcseti("fn",21);
  c_gasetr("xls",0.02);
  c_gasetc("xlf","(i3)");
  c_gasetr("yls",0.02);
  c_gasetc("ylf","(f5.1)");
  c_gasetr("xmj",0.02);
  c_gasetr("ymj",0.02);
}
```

Example 1 - interpolated function, derivative and integral

```
;
;  Description:  This program illustrates the use of ftcurv, ftcurvd,
;                and ftcurvi.

procedure drw_text(label:string, xpos:float, ypos:float, height:float, \
                   output:graphic)
begin
;
; Draw a text string.
;
label1 = create "Label" textItemClass output
    "txPosXF": xpos
    "txPosYF": ypos
    "txFont": 21
    "txJust" : "centercenter"
    "txString": label
    "txFontHeightF": height
end create

draw(label1)
destroy(label1)

end

;
; Main program.
;
begin

;
; Specify the input data.
;
xi = (/  0.00,   2.00,   5.00,   8.00,  10.00,  13.00,     \
        15.00,  18.00,  21.00,  23.00,  30.00          /)
yi = (/  1.00,   0.81,   0.00,  -0.81,  -1.00,  -0.84,     \
        -0.56,   0.04,   0.73,   1.18,   2.0           /)

;
; Create the output X coordinate array.
;
npts = 201
xr = 30.
xo = fspan(0.,xr,npts)

;
; Require the derivatives at the endpoints to be zero.
;
ftsetp("sf1",0)
ftsetp("sl1", 0.0)
ftsetp("sl2", 0.0)
```

```
;
; Interpolate.
;
yo = ftcurv(xi, yi, xo)

;
; Find the derivatives.
;
yd = ftcurvd(xi, yi, xo)

;
; Find the integrals starting from x=0.
;
yint = new( (/ npts /), float)
xinc = xr/(npts-1)
do i = 0, npts-1
  yint(i) = ftcurvi(0., xinc*i, xi, yi)
end do

;
; Create a color map.
;
ncolors = 6
cmap = new( (/ncolors,3/), float)
cmap(0,:) = (/1.0,1.0,1.0/)
cmap(1,:) = (/0.0,0.0,0.0/)
cmap(2,:) = (/1.0,0.0,0.0/)
cmap(3,:) = (/0.0,1.0,0.0/)
cmap(4,:) = (/0.0,0.0,1.0/)
cmap(5,:) = (/1.0,1.0,0.0/)

;
; Default is to display output to an X workstation.
;
NCGM=0
X11=1
PS=0

if (NCGM .eq. 1) then
;
; Create an ncgmWorkstation object.
;
  xworkid = create "ft04Work" ncgmWorkstationClass defaultapp
      "wkMetaName" : "ft01n.ncgm"
      "wkColorMap" : cmap
  end create
else
  if (X11 .eq. 1) then
;
; Create an XWorkstation object.
;
  xworkid = create "ft01Work" xWorkstationClass defaultapp
      "wkPause" : "True"
      "wkColorMap" : cmap
  end create
  else
    if (PS .eq. 1) then
;
; Create a PSWorkstation object.
;
      xworkid = create "ft01Work" psWorkstationClass defaultapp
```

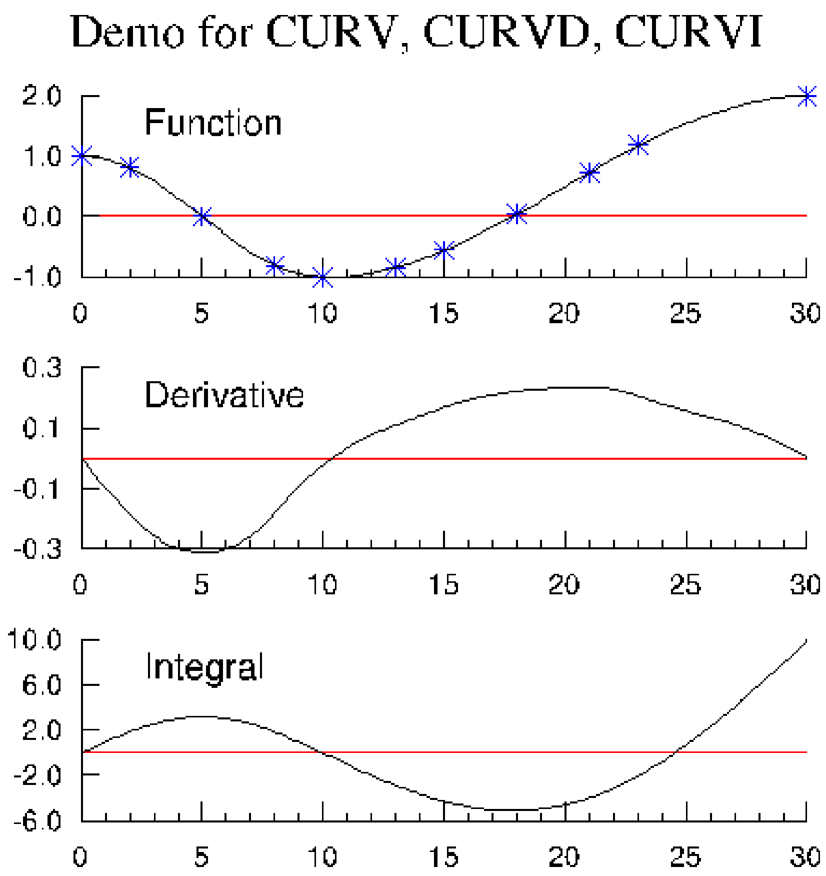
```
        "wkPSFileName" : "ft01n.ps"
        "wkColorMap" : cmap
      end create
    end if
  end if
end if

;
;
; Draw three graphs:
;   1.)  The function values (with markers for the input points),
;   2.)  The derivatives,
;   3.)  The integrals (starting from X=0.).
;

;
; Define a data object with the original points.  The id for this
; object will later be used as the value for an XYPlot data resource,
; 'xyCoordData'.
;
dataid_original_points = create "xyData" coordArraysClass defaultapp
    "caXArray": xi
    "caYArray": yi
end create

;
; Define a data object with the interpolated points.  The id for this
; object will later be used as the value for an XYPlot data resource,
; 'xyCoordData'.
;
dataid_interpolated_points = create "xyData" coordArraysClass defaultapp
    "caXArray": xo
    "caYArray": yo
end create

;
; Define a data object with the derivatives.  The id for this
; object will later be used as the value for an XYPlot data resource,
; 'xyCoordData'.
;
dataid_derivatives = create "xyData" coordArraysClass defaultapp
    "caXArray": xo
    "caYArray": yd
end create

;
; Define a data object with the integrals.  The id for this
; object will later be used as the value for an XYPlot data resource,
; 'xyCoordData'.
;
dataid_integrals = create "xyDatai" coordArraysClass defaultapp
    "caXArray": xo
    "caYArray": yint
end create

;
; Specify a point of origin for three graphs.
;
ypos_top = 0.88

;
```

```
; Main title.
;
drw_text("Demo for ftcurv, ftcurvd, ftcurvi",0.5,.95,0.04,xworkid)

;
; Create an immediate mode polyline for marking the Y=0. line
; in each graph.
;
zero_line_id = create "ZeroLine" graphicStyleClass xworkid
    "gsLineColor": 2
end create
xx = new( (/ 2 /), float)
yy = new( (/ 2 /), float)
xx(0) = 0.
xx(1) = xr
yy(0) = 0.
yy(1) = 0.

;
; Create an XyPlot object for drawing a curve with the interpolated
; points.  This is created as a child of the XWorkstation object.
;
plotid_interpolated_points = create "xyPlot" xyPlotClass xworkid
    "vpXF": .13
    "vpYF": ypos_top
    "vpWidthF" :  .8
    "vpHeightF" : .2
    "trYMaxF" :  2.0
    "trYMinF" : -1.0
    "trXMaxF" : xr
    "trXMinF" :  0.0
    "xyCoordData": dataid_interpolated_points
    "tmXTBorderOn": "False"
    "tmYRBorderOn": "False"
    "tmXBMinorPerMajor": 4
    "tmYLMinorPerMajor": 0
    "tmBorderThicknessF": 1.
    "tmXMajorGridThicknessF": 1.
    "tmXBMajorLengthF": 0.015
    "tmYLMajorLengthF": 0.015
    "tmXBMinorLengthF": 0.0075
    "tmXBMajorThicknessF": 1.0
    "tmYLMajorThicknessF": 1.0
    "tmXTOn": "False"
    "tmYROn": "False"
    "tmXBLabelFont": 21
    "tmXBLabelFontHeightF": 0.025
    "tmYLLabelFont": 21
    "tmYLLabelFontHeightF": 0.025
    "tmYLMode": "Manual"
    "tmYLTickStartF": -1.0
    "tmYLTickSpacingF": 1.0
    "tmYLTickEndF": 2.0
end create
;
; Add original points.
;
orig_points = NhlAddData(plotid_interpolated_points,"xyCoordData",  \
                         dataid_original_points)
getvalues plotid_interpolated_points
    "xyCoordDataSpec": dataspec
```

```
end getvalues

setvalues dataspec(1)
    "xyMarkLineMode": "markers"
    "xyMarkerColor": 4
    "xyMarkerSizeF": 0.025
end setvalues

;
; Draw a graph of the interpolated values and mark the original points.
;
draw(plotid_interpolated_points)
drw_text("Function",0.4,ypos_top - 0.03,0.035,xworkid)
NhlDataPolyline(plotid_interpolated_points, zero_line_id, xx, yy)

;
; Create an XyPlot object for drawing a curve with the derivative points.
;
setvalues plotid_interpolated_points
    "vpYF": ypos_top - 0.3
    "trYMaxF" :  0.35
    "trYMinF" : -0.35
    "xyCoordData": dataid_derivatives
    "tmYLTickStartF": -0.3
    "tmYLTickSpacingF": 0.2
    "tmYLTickEndF": 0.3
end setvalues

;
; Draw the graph of the derivatives.
;
;
draw(plotid_interpolated_points)
drw_text("Derivative",0.3,ypos_top - 0.33,0.035,xworkid)
NhlDataPolyline(plotid_interpolated_points, zero_line_id, xx, yy)

;
; Create an XyPlot object for drawing a curve with the integrals.
;
setvalues plotid_interpolated_points
    "vpYF": ypos_top - 0.6
    "trYMaxF" :  10.
    "trYMinF" :  -6.
    "xyCoordData": dataid_integrals
    "tmYLTickStartF": -6.
    "tmYLTickSpacingF": 4.
    "tmYLTickEndF": 10.
end setvalues

;
; Draw the graph of the integrals.
;
;
draw(plotid_interpolated_points)
drw_text("Integral",0.3,ypos_top - 0.63,0.035,xworkid)
NhlDataPolyline(plotid_interpolated_points, zero_line_id, xx, yy)

frame(xworkid)

;
; End NCL script.
```

```
;
end
```

Plot for example 1 - interpolated function, derivative and integral



Example 2 -- interpolated periodic function and derivative

```
      PROGRAM FTEX02
C
C   Example of CURVP1, CURVPI.
```

```
C
C  Define dimensions, declare arrays.
C
      PARAMETER (IDIM=10,IOUT=201)
      DIMENSION X(IDIM),Y(IDIM),YP(IDIM),TEMP(IDIM,2)
      DIMENSION XO(IOUT),YO(IOUT),YI(IOUT)
C
C Specify the input data.
C
      DATA X/ 0.000, 0.210, 0.360, 0.540, 1.000,
     +        1.500, 1.970, 2.300, 2.500, 2.700/
      DATA Y/ 0.000, 2.600, 3.000, 2.500, 0.000,
     +       -1.000, 0.000, 0.800, 0.920, 0.700/
C
C  Call CURVP1 setup.
C
      PERIOD = 3.
      SIGMA  = 1.
      CALL CURVP1(IDIM, X, Y, PERIOD, YP, TEMP, SIGMA, IERR)
C
C  Call CURVP2 and calculate the interpolated values and the integrals.
C
      XR =  5.
      XL = -1.
      XINC = (XR-XL)/(IOUT-1)
      DO 10 I=1,IOUT
        XO(I) = XL+(I-1)*XINC
        YO(I) = CURVP2(XO(I),IDIM,X,Y,PERIOD,YP,SIGMA)
        YI(I) = CURVPI(0.,XO(I),IDIM,X,Y,PERIOD,YP,SIGMA)
   10 CONTINUE
C
C  Draw a plot of the interpolated functions and mark the original points.
C
      CALL DRWFT2(XL,XR,IDIM,X,Y,IOUT,XO,YO,YI)
C
      STOP
      END
      SUBROUTINE DRWFT2(XL,XR,II,X,Y,IO,XO,YO,YI)
C
C  Define error file, Fortran unit number, workstation type,
C  and workstation ID.
C
      PARAMETER (IERRF=6, LUNIT=2, IWTYPE=1, IWKID=1)
C
      DATA YPOS_TOP/0.85/
C
C  Open GKS, open and activate a workstation.
C
      CALL GOPKS (IERRF, ISZDM)
      CALL GOPWK (IWKID, LUNIT, IWTYPE)
      CALL GACWK (IWKID)
C
C  Define a color table.
C
      CALL GSCR(IWKID, 0, 1.0, 1.0, 1.0)
      CALL GSCR(IWKID, 1, 0.0, 0.0, 0.0)
      CALL GSCR(IWKID, 2, 1.0, 0.0, 0.0)
      CALL GSCR(IWKID, 3, 0.0, 1.0, 0.0)
      CALL GSCR(IWKID, 4, 0.0, 0.0, 1.0)
      CALL GSCLIP(0)
C
```

```
C  Plot main title.
C
      CALL PLCHHQ(0.50,0.95,':F25:Demo for curvp1, curvpi',
     +            0.032,0.,0.)
C
C  Graph the interpolated function values and mark the original
C  input data points.
C
      YB = -2.0
      YT =  3.0
      CALL BKGFT2(XL,XR,YPOS_TOP,'Function',0.42,YB,YT)
      CALL GRIDAL(6,5,5,1,1,1,10,XL,YB)
C
C  Mark the original data points.
C
      CALL GSMKSC(2.)
      CALL GSPMCI(4)
      CALL GPM(II,X,Y)
C
C  Graph the interpolated function values.
C
      CALL GPL(IO,XO,YO)
C
C  Graph the integral.
C
      YB = -1.0
      YT =  4.0
      CALL BKGFT2(XL,XR,YPOS_TOP-0.47,'Integral (from X = 0.)',
     +            0.2,YB,YT)
      CALL GRIDAL(6,5,5,1,1,1,10,XL,YB)
      CALL GPL(IO,XO,YI)
      CALL GSPLCI(1)
C
C  Indicate the period.
C
      CALL DRWPRD(0.,3.,6.5)
C
      CALL FRAME
C
      CALL GDAWK(IWKID)
      CALL GCLWK(IWKID)
      CALL GCLKS
C
      RETURN
      END
      SUBROUTINE BKGFT2(XL,XR,YPOS,LABEL,XLP,YB,YT)
      DIMENSION XX(2),YY(2)
      CHARACTER*(*) LABEL
C
      CALL SET(0.,1.,0.,1.,0.,1.,0.,1.,1)
      CALL PCSETI('FN',21)
      CALL PLCHHQ(XLP,YPOS - 0.03,LABEL,0.025,0.,-1.)
      CALL SET(0.13,0.93,YPOS-0.25,YPOS,XL,XR,YB,YT,1)
      XX(1) = XL
      XX(2) = XR
      YY(1) = 0.
      YY(2) = 0.
      CALL GSPLCI(2)
      CALL GPL(2,XX,YY)
      CALL GSPLCI(1)
```

```
      CALL GASETI('LTY',1)
      CALL PCSETI('FN',21)
      CALL GASETR('XLS',0.02)
      CALL GASETC('XLF','(F4.1)')
      CALL GASETR('YLS',0.02)
      CALL GASETC('YLF','(F5.2)')
      CALL GASETR('XMJ',0.02)
      CALL GASETR('YMJ',0.02)
C
      RETURN
      END
      SUBROUTINE DRWPRD(XL,XR,Y)
C
C  Draws a bounding indicator for the period of the function.
C
      DIMENSION XX(2),YY(2)
C
      YOFF = 0.4
      CALL GSPLCI(2)
      XMID = 0.5*(XR-XL)
      CALL PLCHHQ(XMID,Y,':F25:Period',0.02,0.,0.)
C
C  Vertical lines at the period limits.
C
      XX(1) = XL
      XX(2) = XL
      YY(1) = Y+YOFF
      YY(2) = Y-YOFF
      CALL GPL(2,XX,YY)
      XX(1) = XR
      XX(2) = XR
      CALL GPL(2,XX,YY)
C
C  Horizontal lines between label and vertical lines.
C
      CALL PCSETI('TE',1)
      CALL PCGETR('XB',XB)
      CALL PCGETR('XE',XE)
      XX(1) = XL
      XX(2) = CFUX(XB)-0.09
      YY(1) = Y
      YY(2) = Y
      CALL GPL(2,XX,YY)
C
C  Left arrow.
C
      XX(1) = XL
      YY(1) = Y
      YI = 0.5*YOFF
      YY(2) = Y+YI
      XX(2) = XL+YI
      CALL GPL(2,XX,YY)
      YY(2) = Y-YI
      CALL GPL(2,XX,YY)
C
      XX(1) = XR
      XX(2) = CFUX(XE)+0.09
      YY(1) = Y
      YY(2) = Y
      CALL GPL(2,XX,YY)
C
```

```
C  Right arrow.
C
      XX(1) = XR
      YY(1) = Y
      YY(2) = Y+YI
      XX(2) = XR-YI
      CALL GPL(2,XX,YY)
      YY(2) = Y-YI
      CALL GPL(2,XX,YY)
C
      RETURN
      END
```

Example 2 -- interpolated periodic function and derivative

```
#include <stdio.h>
#include <ncarg/ncargC.h>
#include <ncarg/gks.h>
#include <ncarg/ngmath.h>

void c_bkgft2(float, float, float, char *, float, float, float);
void c_drwft2(float, float, int, float [], float [], int,
              float [], float [], float []);
void c_drwprd(float, float, float);

/*
 *  Example of curvp1, curvp2, curvpi.
 */

#define IDIM  10
#define IOUT 201

#define IWTYPE 1
#define WKID   1

main()
{
  float x[] = { 0.000, 0.210, 0.360, 0.540, 1.000,
                1.500, 1.970, 2.300, 2.500, 2.700};
  float y[] = { 0.000, 2.600, 3.000, 2.500, 0.000,
               -1.000, 0.000, 0.800, 0.920, 0.700};
  float xinc, xo[IOUT], yo[IOUT], yi[IOUT];
  float period = 3., xr = 5., xl = -1.;
  int   i;

/*
 *  Create the output X coordinate array.
 */
  xinc =  (xr-xl)/(IOUT-1);
  for (i = 0; i < IOUT; i++) {
    xo[i] = xl + xinc*i;
  }
```

```
/*
 *  Calculate the interpolated values and the integral.
 */
  c_ftcurvp(IDIM, x, y, period, IOUT, xo, yo);
  for (i = 0; i < IOUT; i++) {
    c_ftcurvpi(0., xo[i], period, IDIM, x, y, yi+i);
  }

/*
 *  Draw plot.
 */
  c_drwft2(xl,xr,IDIM,x,y,IOUT,xo,yo,yi);

}

void c_drwft2(float xl,float xr, int n, float x[], float y[],
              int m, float xo[], float yo[], float yi[])
{

  int   i;
  float yb, yt, ypos_top = 0.85;

  Gcolr_rep rgb;
  Gpoint plist[IDIM];
  Gpoint_list pmk;

/*
 *  Open GKS, open and activate a workstation.
 */
  gopen_gks("stdout",0);
  gopen_ws(WKID, NULL, IWTYPE);
  gactivate_ws(WKID);

/*
 * Define a color table.
 */
  rgb.rgb.red = rgb.rgb.green = rgb.rgb.blue = 1.;
  gset_colr_rep(WKID,0,&rgb);
  rgb.rgb.red = rgb.rgb.green = rgb.rgb.blue = 0.;
  gset_colr_rep(WKID,1,&rgb);
  rgb.rgb.red = 1.;
  rgb.rgb.green = rgb.rgb.blue = 0.;
  gset_colr_rep(WKID,2,&rgb);
  rgb.rgb.red = rgb.rgb.green = 0.;
  rgb.rgb.blue = 1.;
  gset_colr_rep(WKID,3,&rgb);

/*
 * Plot the main title.
 */
  gset_clip_ind(0);
  c_plchhq(.5,.95,":F25:Demo for c_ftcurvp, c_ftcurvpi",0.03,0.,0.);

/*
 * Graph the interpolated function values and mark the original
 * input data points.
 */
  yb = -2.0;
  yt =  3.0;
  c_bkgft2(xl,xr,ypos_top,"Function",0.42,yb,yt);
```

```
  c_gridal(6,5,5,1,1,1,10,xl,yb);
  c_curve(xo,yo,m);

/*
 *  Mark the input data points.
 */
  for (i = 0; i < n; i++) {
    plist[i].x = x[i];
    plist[i].y = y[i];
  }
  gset_marker_size(2.);
  gset_marker_colr_ind(3);
  pmk.num_points = n;
  pmk.points = plist;
  gpolymarker(&pmk);

/*
 *  Graph the integral.
 */
  yb = -1.0;
  yt =  4.0;
  c_bkgft2(xl,xr,ypos_top-0.47,"Integral (from X = 0.)",0.2,yb,yt);
  c_gridal(6,5,5,1,1,1,10,xl,yb);
  c_curve(xo,yi,m);

/*
 *  Indicate the period.
 */
  c_drwprd(0.,3.,6.5);

  c_frame();
/*
 *  Deactivate and close workstation, close GKS.
 */
  gdeactivate_ws(WKID);
  gclose_ws(WKID);
  gclose_gks();
}

void c_bkgft2(float xleft, float xright, float ypos, char *label,
              float xlp, float yb, float yt) {
  c_set(0.,1.,0.,1.,0.,1.,0.,1.,1);
  c_pcseti("fn",21);
  c_plchhq(xlp,ypos-0.03,label,0.025,0.,-1.0);
  c_set(0.13,0.93,ypos-0.25,ypos,xleft,xright, yb, yt, 1);
  gset_line_colr_ind(2);
  c_line(xleft, 0., xright, 0.);
  c_sflush();
  gset_line_colr_ind(1);
  c_gaseti("lty",1);
  c_pcseti("fn",21);
  c_gasetr("xls",0.02);
  c_gasetc("xlf","(f4.1)");
  c_gasetr("yls",0.02);
  c_gasetc("ylf","(f5.2)");
  c_gasetr("xmj",0.02);
  c_gasetr("ymj",0.02);
}

void c_drwprd(float xl, float xr, float y)
{
```

```
/*
 *  Draw a bounding indicator for the period of the function.
 */
  float xx[2], yy[2], xmid, yoff = 0.4, yi, xb, xe;

  gset_line_colr_ind(2);
  xmid = 0.5*(xr-xl);
  c_plchhq(xmid,y,":F25:Period",0.02,0.,0.);

/*
 *  Vertical lines at the period limits.
 */
  c_line(xl, y+yoff, xl, y-yoff);
  c_line(xr, y+yoff, xr, y-yoff);

/*
 *  Horizontal lines between label and vertical lines.
 */
  c_pcseti("te",1);
  c_pcgetr("xb",&xb);
  c_pcgetr("xe",&xe);
  c_line(xl, y, c_cfux(xb)-0.09, y);

/*
 *  Left arrow.
 */
  yi = 0.5*yoff;
  c_line(xl, y, xl+yi, y+yi);
  c_line(xl, y, xl+yi, y-yi);

  c_line(xr, y, c_cfux(xe)+0.09, y);

/*
 *  Right arrow.
 */
  c_line(xr, y, xr-yi, y+yi);
  c_line(xr, y, xr-yi, y-yi);
}
```

Example 2 -- interpolated periodic function and integral

```
;
;  Description:  This program illustrates the use of ftcurvp and ftcurvpi.
;

procedure drw_text(label:string, xpos:float, ypos:float, height:float, \
                   output:graphic)
begin
;
; Draw a text string.
;
label1 = create "Label" textItemClass output
```

```
    "txPosXF": xpos
    "txPosYF": ypos
    "txFont": 21
    "txJust" : "centercenter"
    "txString": label
    "txFontHeightF": height
end create

draw(label1)
destroy(label1)

end

;
; Main program.
;
begin

;
; Specify the input data.
;
xi = (/  0.000, 0.210, 0.360, 0.540, 1.000,    \
         1.500, 1.970, 2.300, 2.500, 2.700  /)
yi = (/  0.000, 2.600, 3.000, 2.500, 0.000,    \
        -1.000, 0.000, 0.800, 0.920, 0.700  /)

;
; Create the output X coordinate array.
;
npts = 201
xr = 5.
xl = -1.
period = 3.
xo = fspan(xl,xr,npts)

;
; Interpolate.
;
yo = ftcurvp(xi, yi, period, xo)

;
; Find the integrals starting from x=0.
;
yint = new( (/ npts /), float)
do i = 0, npts-1
  yint(i) = ftcurvpi(0., xo(i), period, xi, yi)
end do

;
; Create a color map.
;
ncolors = 6
cmap = new( (/ncolors,3/), float)
cmap(0,:) = (/1.0,1.0,1.0/)
cmap(1,:) = (/0.0,0.0,0.0/)
cmap(2,:) = (/1.0,0.0,0.0/)
cmap(3,:) = (/0.0,1.0,0.0/)
cmap(4,:) = (/0.0,0.0,1.0/)
cmap(5,:) = (/1.0,1.0,0.0/)

;
```

```
; Default is to display output to an X workstation.
;
NCGM=0
X11=1
PS=0

if (NCGM .eq. 1) then
;
; Create an ncgmWorkstation object.
;
  xworkid = create "ft04Work" ncgmWorkstationClass defaultapp
      "wkMetaName" : "ft01n.ncgm"
      "wkColorMap" : cmap
  end create
else
  if (X11 .eq. 1) then
;
; Create an XWorkstation object.
;
  xworkid = create "ft01Work" xWorkstationClass defaultapp
      "wkPause" : "True"
      "wkColorMap" : cmap
  end create
  else
    if (PS .eq. 1) then
;
; Create a PSWorkstation object.
;
      xworkid = create "ft01Work" psWorkstationClass defaultapp
        "wkPSFileName" : "ft01n.ps"
        "wkColorMap" : cmap
      end create
    end if
  end if
end if

;
;
; Draw two graphs:
;   1.)  The function values (with markers for the input points),
;   2.)  The integrals (starting from X=0.).
;

;
; Define a data object with the original points.
;
dataid_original_points = create "xyData" coordArraysClass defaultapp
    "caXArray": xi
    "caYArray": yi
end create

;
; Define a data object with the interpolated points.
;
dataid_interpolated_points = create "xyData" coordArraysClass defaultapp
    "caXArray": xo
    "caYArray": yo
end create

;
; Define a data object with the integrals.
```

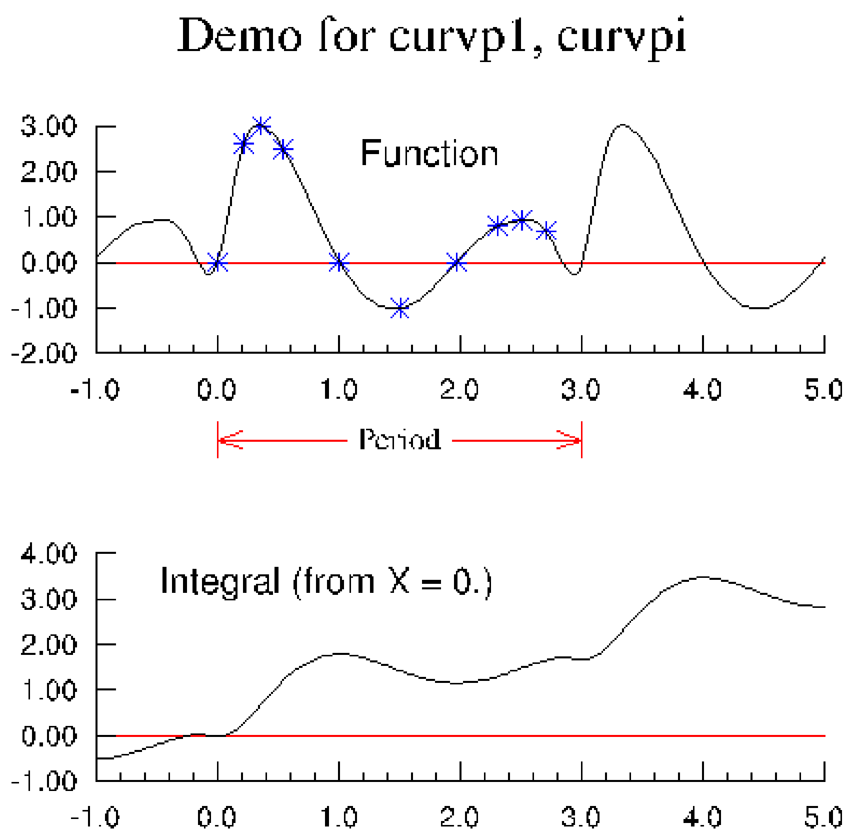
```
;
dataid_integrals = create "xyDatai" coordArraysClass defaultapp
    "caXArray": xo
    "caYArray": yint
end create

;
; Create an immediate mode polyline for marking the Y=0. line
; in each graph.
;
zero_line_id = create "ZeroLine" graphicStyleClass xworkid
    "gsLineColor": 2
end create
xx = new( (/ 2 /), float)
yy = new( (/ 2 /), float)
xx(0) = xl
xx(1) = xr
yy(0) = 0.
yy(1) = 0.

;
; Create a blank screen plot object.
;
period_legend = create "PeriodLegend" logLinPlotClass xworkid
    "vpXF": 0.13
    "vpYF": 1.00
    "vpWidthF" :  .8
    "vpHeightF" : 1.00
    "trYMaxF" :   6.00
    "trYMinF" :   0.00
    "trXMaxF" :  xr
    "trXMinF" :  xl
end create
;
; Set GraphicStyle resources to modify the immediate mode line attributes.
;
getvalues xworkid
  "wkDefGraphicStyleId": gid
end getvalues

setvalues gid
      "gsLineColor" : 2
end setvalues

ybase = 3.
lbar = xl+1.
off1 = 0.075
off2 = 2.*off1
xa = new( (/ 9 /), float)
ya = new( (/ 9 /), float)
xa(0) = 1.1
ya(0) = ybase
xa(1) = lbar
ya(1) = ybase
xa(2) = lbar
ya(2) = ybase+off2
xa(3) = xa(1)
ya(3) = ya(1)
xa(4) = lbar+off2
ya(4) = ybase+off1
xa(5) = xa(1)
```

```
ya(5) = ya(1)
xa(6) = xa(4)
ya(6) = ybase-off1
xa(7) = xa(1)
ya(7) = ya(1)
xa(8) = lbar
ya(8) = ybase-off2

rbar = lbar+3.
xb = new( (/ 8 /), float)
yb = new( (/ 8 /), float)
xb(0) = 1.90
yb(0) = ybase
xb(1) = rbar
yb(1) = yb(0)
xb(2) = ybase-off2
yb(2) = ybase+off1
xb(3) = xb(1)
yb(3) = yb(1)
xb(4) = xb(2)
yb(4) = yb(1)-off1
xb(5) = xb(1)
yb(5) = yb(1)
xb(6) = rbar
yb(6) = ybase+off2
xb(7) = rbar
yb(7) = yb(0)-off2

;
; Main title.
;
drw_text("Demo for ftcurvp, ftcurvpi",0.5,.95,0.04,xworkid)

;
; Create an XyPlot object for drawing a curve with the interpolated points.
;
plotid_interpolated_points = create "xyPlot" xyPlotClass xworkid
    "vpXF": .13
    "vpYF": .85
    "vpWidthF" :  .8
    "vpHeightF" : .25
    "trYMaxF" :   3.
    "trYMinF" :  -2.
    "trXMaxF" :  xr
    "trXMinF" :  xl
    "xyCoordData": dataid_interpolated_points
    "tmXTBorderOn": "False"
    "tmYRBorderOn": "False"
    "tmXBMinorPerMajor": 4
    "tmYLMinorPerMajor": 0
    "tmBorderThicknessF": 1.
    "tmXMajorGridThicknessF": 1.
    "tmXBMajorLengthF": 0.015
    "tmYLMajorLengthF": 0.015
    "tmXBMinorLengthF": 0.0075
    "tmXBMajorThicknessF": 1.0
    "tmYLMajorThicknessF": 1.0
    "tmXTOn": "False"
    "tmYROn": "False"
    "tmXBLabelFont": 21
    "tmXBLabelFontHeightF": 0.025
```

```
    "tmYLLabelFont": 21
    "tmYLLabelFontHeightF": 0.025
    "tmYLMode": "Manual"
    "tmYLTickStartF": -2.
    "tmYLTickSpacingF": 1.0
    "tmYLTickEndF": 3.
    "tmXBMode": "Manual"
    "tmXBTickStartF": -1.
    "tmXBTickSpacingF": 1.0
    "tmXBTickEndF": 5.
end create
;
; Add original points.
;
orig_points = NhlAddData(plotid_interpolated_points,"xyCoordData",  \
                         dataid_original_points)
getvalues plotid_interpolated_points
    "xyCoordDataSpec": dataspec
end getvalues

setvalues dataspec(1)
    "xyMarkLineMode": "markers"
    "xyMarkerColor": 4
    "xyMarkerSizeF": 0.025
end setvalues

;
; Draw a graph of the interpolated values and mark the original points.
;
draw(plotid_interpolated_points)
drw_text("Function",0.46,.82,0.03,xworkid)
NhlDataPolyline(plotid_interpolated_points, zero_line_id, xx, yy)


;
; Create an XyPlot object for drawing a curve with the integrals.
;
setvalues plotid_interpolated_points
    "vpYF": 0.37
    "trYMaxF" :  4.0
    "trYMinF" :  -1.0
    "xyCoordData": dataid_integrals
    "tmYLTickStartF": -2.
    "tmYLTickSpacingF": 1.
    "tmYLTickEndF": 4.
end setvalues

;
; Draw the period legend.
;
NhlDataPolyline(period_legend, gid, xb, yb)
NhlDataPolyline(period_legend, gid, xa, ya)
drw_text("Period",0.465,0.5,0.024,xworkid)

;
;
; Draw a graph of the integrals, label the graph,
; and draw a line indicating Y=0.
;
drw_text("Integral (from X = 0.)",0.4,.35,0.03,xworkid)
NhlDataPolyline(plotid_interpolated_points, zero_line_id, xx, yy)
```

```
draw(plotid_interpolated_points)

frame(xworkid)

;
; End NCL script.
;
end
```

Plot for example 2 -- interpolated periodic function and integral



Example 3 -- interpolated smoothing functions

```
      PROGRAM FTEX03
C
```

```
C  Example of CURVS, CURVPS.
C
C  Define dimensions, declare arrays.
C
      PARAMETER (IDIM=10,IOUT=201)
      DIMENSION X(IDIM),Y(IDIM),YS(IDIM),YSP(IDIM),TEMP(IDIM,11)
      DIMENSION XO(IOUT),YOS(IOUT),YOSP(IOUT)
C
C Specify the input data.
C
C
      DATA X/ 0.000, 0.210, 0.360, 0.540, 1.000,
     +        1.500, 1.970, 2.300, 2.500, 2.700/
      DATA Y/ 0.000, 2.600, 3.000, 2.500, 0.000,
     +       -1.000, 0.000, 0.800, 0.920, 0.700/
C
C  Call CURVS setup.
C
      SIGMA = 1.0
      D     = 0.3
      ISW   = 1
      S     = REAL(IDIM)
      EPS   = SQRT(2./S)
      CALL CURVS(IDIM,X,Y,D,ISW,S,EPS,YS,YSP,SIGMA,TEMP,IERR)
C
C  Call CURVP2 and calculate the interpolated values and the integrals.
C
      XR =  5.
      XL = -1.
      XINC = (XR-XL)/(IOUT-1)
      DO 10 I=1,IOUT
        XO(I) = XL+(I-1)*XINC
        YOS(I) = CURV2(XO(I),IDIM,X,YS,YSP,SIGMA)
   10 CONTINUE
C
C  Call CURVPS setup.
C
      P = 3.
      CALL CURVPS(IDIM,X,Y,P,D,ISW,S,EPS,YS,YSP,SIGMA,TEMP,IERR)
C
C  Call CURVP2 and calculate the interpolated values.
C
      DO 15 I=1,IOUT
        YOSP(I) = CURVP2(XO(I),IDIM,X,YS,P,YSP,SIGMA)
   15 CONTINUE
C
C  Plot the results.
C
      CALL DRWFT3(XL,XR,IDIM,X,Y,IOUT,XO,YO,YOS,YOSP)
C
      STOP
      END
      SUBROUTINE DRWFT3(XL,XR,II,X,Y,IO,XO,YO,YOS,YOSP)
C
C  Define error file, Fortran unit number, workstation type,
C  and workstation ID.
C
      PARAMETER (IERRF=6, LUNIT=2, IWTYPE=1, IWKID=1)
C
      DATA YPOS_TOP/0.95/
C
```

```
C  Open GKS, open and activate a workstation.
C
      CALL GOPKS (IERRF, ISZDM)
      CALL GOPWK (IWKID, LUNIT, IWTYPE)
      CALL GACWK (IWKID)
C
C  Define a color table.
C
      CALL GSCR(IWKID, 0, 1.0, 1.0, 1.0)
      CALL GSCR(IWKID, 1, 0.0, 0.0, 0.0)
      CALL GSCR(IWKID, 2, 1.0, 0.0, 0.0)
      CALL GSCR(IWKID, 3, 0.0, 1.0, 0.0)
      CALL GSCR(IWKID, 4, 0.0, 0.0, 1.0)
      CALL GSCLIP(1)
C
C  Graph the interpolated function values and mark the original
C  input data points.
C
      YB = -2.0
      YT =  4.0
      CALL BKGFT3(XL,XR,YPOS_TOP,'CURVS',0.42,YB,YT)
      CALL GRIDAL(6,5,3,1,1,1,10,XL,YB)
C
C  Mark the original data points.
C
      CALL GSMKSC(2.)
      CALL GSPMCI(4)
      CALL GPM(II,X,Y)
C
C  Graph the interpolated function values.
C
      CALL GPL(IO,XO,YOS)
C
C  Graph the periodic function.
C
      CALL BKGFT3(XL,XR,YPOS_TOP-0.5,'CURVPS',0.42,YB,YT)
      CALL GRIDAL(6,5,3,1,1,1,10,XL,YB)
      CALL GPL(IO,XO,YOSP)
      CALL GSPLCI(1)
C
C  Mark the original data points.
C
      CALL GSMKSC(2.)
      CALL GSPMCI(4)
      CALL GPM(II,X,Y)
C
      CALL FRAME
C
      CALL GDAWK(IWKID)
      CALL GCLWK(IWKID)
      CALL GCLKS
C
      RETURN
      END
      SUBROUTINE BKGFT3(XL,XR,YPOS,LABEL,XLP,YB,YT)
      DIMENSION XX(2),YY(2)
      CHARACTER*(*) LABEL
C
      CALL SET(0.,1.,0.,1.,0.,1.,0.,1.,1)
      CALL PCSETI('FN',21)
      CALL PLCHHQ(XLP,YPOS - 0.03,LABEL,0.025,0.,-1.)
```

```
      CALL SET(0.13,0.93,YPOS-0.35,YPOS,XL,XR,YB,YT,1)
      XX(1) = XL
      XX(2) = XR
      YY(1) = 0.
      YY(2) = 0.
      CALL GSPLCI(2)
      CALL GPL(2,XX,YY)
      CALL GSPLCI(1)

      CALL GASETI('LTY',1)
      CALL PCSETI('FN',21)
      CALL GASETR('XLS',0.02)
      CALL GASETC('XLF','(F4.1)')
      CALL GASETR('YLS',0.02)
      CALL GASETC('YLF','(F5.2)')
      CALL GASETR('XMJ',0.02)
      CALL GASETR('YMJ',0.02)
C
      RETURN
      END
```

Example 3 -- interpolated smoothing functions

```
#include <stdio.h>
#include <ncarg/ncargC.h>
#include <ncarg/gks.h>
#include <ncarg/ngmath.h>

void c_drwft3(float, float, int, float [], float [],
              int, float [], float [], float []);
void c_bkgft3(float, float, float, char *, float, float, float);

/*
 *  Example of curvs, curvps.
 */

#define IDIM  10
#define IOUT 201

#define IWTYPE 1
#define WKID   1

main()
{
  float x[] = { 0.000, 0.210, 0.360, 0.540, 1.000,
                1.500, 1.970, 2.300, 2.500, 2.700};
  float y[] = { 0.000, 2.600, 3.000, 2.500, 0.000,
               -1.000, 0.000, 0.800, 0.920, 0.700};
  float xinc, xo[IOUT], yos[IOUT], yosp[IOUT];
  float p = 3., d, xr = 5., xl = -1.;
  int   i;

/*
 *  Create the output X coordinate array.
```

```
 */
  xinc =  (xr-xl)/(IOUT-1);
  for (i = 0; i < IOUT; i++) {
    xo[i] = xl + xinc*i;
  }

/*
 *  Calculate the interpolated values.
 */
  d = 0.3;
  c_ftcurvs(IDIM, x, y, 1, &d, IOUT, xo, yos);
/*
 *  Calculate the interpolated values for a periodic function.
 */
  c_ftcurvps(IDIM, x, y, p, 1, &d, IOUT, xo, yosp);

/*
 *  Draw plot.
 */
  c_drwft3(xl, xr, IDIM, x, y, IOUT, xo, yos, yosp);

}
void c_drwft3(float xl, float xr, int n, float x[], float y[],
              int m, float xo[], float yos[], float yosp[])
{

  float ypos_top = 0.95, yb, yt;
  int   i;

  Gcolr_rep rgb;
  Gpoint plist[IDIM];
  Gpoint_list pmk;

/*
 *  Open GKS, open and activate a workstation.
 */
  gopen_gks("stdout",0);
  gopen_ws(WKID, NULL, IWTYPE);
  gactivate_ws(WKID);

/*
 * Define a color table.
 */
  rgb.rgb.red = rgb.rgb.green = rgb.rgb.blue = 1.;
  gset_colr_rep(WKID,0,&rgb);
  rgb.rgb.red = rgb.rgb.green = rgb.rgb.blue = 0.;
  gset_colr_rep(WKID,1,&rgb);
  rgb.rgb.red = 1.;
  rgb.rgb.green = rgb.rgb.blue = 0.;
  gset_colr_rep(WKID,2,&rgb);
  rgb.rgb.red = rgb.rgb.green = 0.;
  rgb.rgb.blue = 1.;
  gset_colr_rep(WKID,3,&rgb);

  gset_clip_ind(1);

/*
 * Graph the interpolated function values and mark the original
 * input data points.
 */
  yb = -2.0;
```

```
  yt =  4.0;
  c_bkgft3(xl,xr,ypos_top,"c_curvs",0.4,yb,yt);
  c_gridal(6,5,3,1,1,1,10,xl,yb);
  c_curve(xo,yos,m);

/*
 *  Mark the input data points.
 */
  for (i = 0; i < n; i++) {
    plist[i].x = x[i];
    plist[i].y = y[i];
  }
  gset_marker_size(2.);
  gset_marker_colr_ind(3);
  pmk.num_points = n;
  pmk.points = plist;
  gpolymarker(&pmk);

/*
 *  Graph the periodic function.
 */
  c_bkgft3(xl,xr,ypos_top-0.47,"c_curvps",0.4,yb,yt);
  c_gridal(6,5,3,1,1,1,10,xl,yb);
  c_curve(xo,yosp,m);

/*
 *  Mark the input data points.
 */
  for (i = 0; i < n; i++) {
    plist[i].x = x[i];
    plist[i].y = y[i];
  }
  gset_marker_size(2.);
  gset_marker_colr_ind(3);
  pmk.num_points = n;
  pmk.points = plist;
  gpolymarker(&pmk);
  c_frame();

/*
 *  Deactivate and close workstation, close GKS.
 */
  gdeactivate_ws(WKID);
  gclose_ws(WKID);
  gclose_gks();
}

void c_bkgft3(float xleft, float xright, float ypos, char *label,
              float xlp, float yb, float yt) {
  c_set(0.,1.,0.,1.,0.,1.,0.,1.,1);
  c_pcseti("fn",21);
  c_plchhq(xlp,ypos-0.03,label,0.035,0.,-1.0);
  c_set(0.13,0.93,ypos-0.35,ypos,xleft,xright, yb, yt, 1);
  gset_line_colr_ind(2);
  c_line(xleft, 0., xright, 0.);
  c_sflush();
  gset_line_colr_ind(1);
  c_gaseti("lty",1);
  c_pcseti("fn",21);
  c_gasetr("xls",0.02);
  c_gasetc("xlf","(f4.1)");
```

```
  c_gasetr("yls",0.02);
  c_gasetc("ylf","(f5.2)");
  c_gasetr("xmj",0.02);
  c_gasetr("ymj",0.02);
}
```

Example 3 -- interpolated smoothing functions

```
;
;  Description:  This program illustrates the use of ftcurvs and ftcurvps.
;

;
; Main program.
;
begin
;
; Specify the input data.
;
xi = (/  0.000, 0.210, 0.360, 0.540, 1.000,    \
         1.500, 1.970, 2.300, 2.500, 2.700  /)
yi = (/  0.000, 2.600, 3.000, 2.500, 0.000,    \
        -1.000, 0.000, 0.800, 0.920, 0.700  /)

;
; Create the output X coordinate array.
;
npts = 201
xr = 5.
xl = -1.
period = 3.
xo = fspan(xl,xr,npts)

;
; Interpolate.
;
d = 0.3
yos = ftcurvs(xi, yi, d, xo)

;
; Same as above, except for a periodic function.
;
yosp = ftcurvps(xi, yi, period, d, xo)

;
; Create a color map.
;
ncolors = 6
cmap = new( (/ncolors,3/), float)
cmap(0,:) = (/1.0,1.0,1.0/)
cmap(1,:) = (/0.0,0.0,0.0/)
cmap(2,:) = (/1.0,0.0,0.0/)
cmap(3,:) = (/0.0,1.0,0.0/)
cmap(4,:) = (/0.0,0.0,1.0/)
```

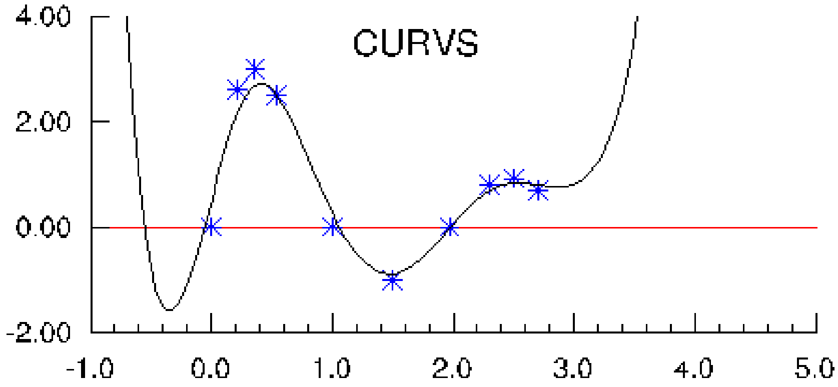
```
cmap(5,:) = (/1.0,1.0,0.0/)

;
; Default is to display output to an X workstation.
;
NCGM=0
X11=1
PS=0

if (NCGM .eq. 1) then
;
; Create an ncgmWorkstation object.
;
  xworkid = create "ft04Work" ncgmWorkstationClass defaultapp
      "wkMetaName" : "ft01n.ncgm"
      "wkColorMap" : cmap
  end create
else
  if (X11 .eq. 1) then
;
; Create an XWorkstation object.
;
  xworkid = create "ft01Work" xWorkstationClass defaultapp
      "wkPause" : "True"
      "wkColorMap" : cmap
  end create
  else
    if (PS .eq. 1) then
;
; Create a PSWorkstation object.
;
      xworkid = create "ft01Work" psWorkstationClass defaultapp
        "wkPSFileName" : "ft01n.ps"
        "wkColorMap" : cmap
      end create
    end if
  end if
end if

;
;
; Draw two graphs:
;   1.)  The function values for the non-periodic curve.
;   2.)  The function values for the periodic curve.
;

;
; Define a data object with the original points.  The id for this
; object will later be used as the value for an XYPlot data resource,
; 'xyCoordData'.
;
dataid_original_points = create "xyData" coordArraysClass defaultapp
    "caXArray": xi
    "caYArray": yi
end create

;
; Define a data object with the interpolated points.
;
dataid_interpolated_points = create "xyData" coordArraysClass defaultapp
    "caXArray": xo
```

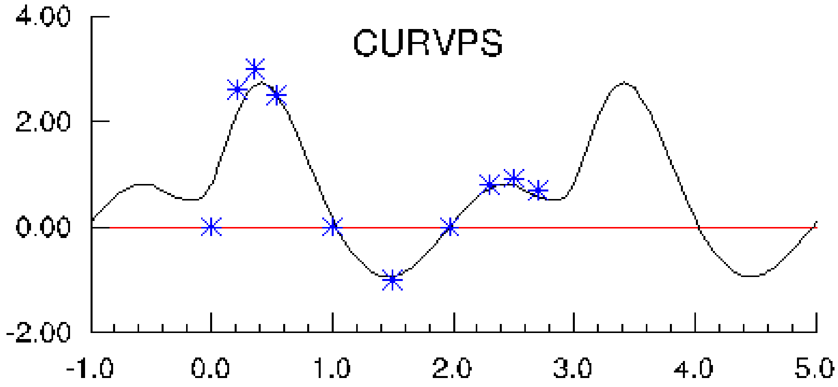
```
    "caYArray": yos
end create

;
; Define a data object with the functional values for the periodic function.
;
dataid_periodic = create "xyDatai" coordArraysClass defaultapp
    "caXArray": xo
    "caYArray": yosp
end create

;
; Specify a point of origin for the graphs.
;
ypos_top = 0.95

;
; Create an immediate mode polyline for marking the Y=0. line
; in each graph.
;
zero_line_id = create "ZeroLine" graphicStyleClass xworkid
    "gsLineColor": 2
end create
xx = new( (/ 2 /), float)
yy = new( (/ 2 /), float)
xx(0) = xl
xx(1) = xr
yy(0) = 0.
yy(1) = 0.

;
; Create a label for the graph of function values of the non-periodic
; function.
;
label1 = create "FunctionLabel" textItemClass xworkid
    "txPosXF": 0.42
    "txPosYF": ypos_top - 0.05
    "txFont": 21
    "txJust" : "centerleft"
    "txString": "ftcurvs"
    "txFontHeightF": 0.035
end create
;
; Create an XyPlot object for drawing a curve with the interpolated
; points for the non-periodic function.
;
plotid_interpolated_points = create "xyPlot" xyPlotClass xworkid
    "vpXF": .13
    "vpYF": ypos_top
    "vpWidthF" :  .8
    "vpHeightF" : .35
    "trYMaxF" :   4.
    "trYMinF" :  -2.
    "trXMaxF" :  xr
    "trXMinF" :  xl
    "xyCoordData": dataid_interpolated_points
    "tmXTBorderOn": "False"
    "tmYRBorderOn": "False"
    "tmXBMinorPerMajor": 4
    "tmYLMinorPerMajor": 0
    "tmBorderThicknessF": 1.
```

```
    "tmXMajorGridThicknessF": 1.
    "tmXBMajorLengthF": 0.015
    "tmYLMajorLengthF": 0.015
    "tmXBMinorLengthF": 0.0075
    "tmXBMajorThicknessF": 1.0
    "tmYLMajorThicknessF": 1.0
    "tmXTOn": "False"
    "tmYROn": "False"
    "tmXBLabelFont": 21
    "tmXBLabelFontHeightF": 0.025
    "tmYLLabelFont": 21
    "tmYLLabelFontHeightF": 0.025
    "tmYLMode": "Manual"
    "tmYLTickStartF": -2.
    "tmYLTickSpacingF": 2.0
    "tmYLTickEndF": 4.
    "tmXBMode": "Manual"
    "tmXBTickStartF": -1.
    "tmXBTickSpacingF": 1.0
    "tmXBTickEndF": 5.
end create

;
; Add original points.
;
orig_points = NhlAddData(plotid_interpolated_points,"xyCoordData",  \
                         dataid_original_points)

getvalues plotid_interpolated_points
    "xyCoordDataSpec": dataspec
end getvalues

setvalues dataspec(1)
    "xyMarkLineMode": "markers"
    "xyMarkerColor": 4
    "xyMarkerSizeF": 0.025
end setvalues

;
; Create a label for the graph of the periodic function.
;
label3 = create "IntegralLabel" textItemClass xworkid
    "txPosXF": 0.42
    "txPosYF": ypos_top - 0.55
    "txFont": 21
    "txJust" : "centerleft"
    "txString": "ftcurvps"
    "txFontHeightF": 0.035
end create

;
;  Draw the curve for the non-periodic function and mark the original points.
;
draw(plotid_interpolated_points)
draw(label1)
NhlDataPolyline(plotid_interpolated_points, zero_line_id, xx, yy)

;
; Create an XyPlot object for drawing a curve with the interpolated
; values of the periodic function.
;
```

```
setvalues plotid_interpolated_points
    "vpYF": ypos_top - 0.5
    "trYMaxF" :  4.0
    "trYMinF" :  -2.0
    "xyCoordData": dataid_periodic
    "tmYLTickStartF": -2.
    "tmYLTickSpacingF": 2.0
    "tmYLTickEndF": 4.
end setvalues

;
; Draw a graph of the periodic function, label the graph,
; and draw a line indicating Y=0.
;
draw(plotid_interpolated_points)
draw(label3)
NhlDataPolyline(plotid_interpolated_points, zero_line_id, xx, yy)

frame(xworkid)

;
; End NCL script.
;
end
```

Plot for example 3 -- interpolated smoothing functions





Example 4 -- interpolated parametric curves and derivatives

```
      PROGRAM FTEX04
C
C  Example of KURV1, KURV2, KURVD.
C
      PARAMETER (IDIM=11,IOUT=201,IDTEMP=9*IDIM)
      DIMENSION X(IDIM),Y(IDIM),TEMP(IDTEMP),U(IOUT),XO(IOUT),YO(IOUT),
     +          XS(IOUT),YS(IOUT),XD(IOUT),YD(IOUT),XDD(IOUT),YDD(IOUT)
      DIMENSION XP(IDIM),YP(IDIM),S(IDIM)
C
      DATA X/ 3.,  4.,  9., 16., 21., 27., 34., 36., 34., 26., 18./
      DATA Y/ 2.4,  9.6, 14.4, 12.0,  9.6,  8.4,
     +       13.2, 21.6, 30.0, 37.2, 38.4/
C
C  Do KURV1 setup.
C
      SIGMA = 1.
      ISF   = 3
      CALL KURV1(IDIM,X,Y,SLOP1,SLOP2,ISF,XP,YP,TEMP,S,SIGMA,IERR)
```

```
      IF (IERR .NE. 0) THEN
        PRINT *, 'Error return from KURV1 =',IERR
        STOP
      ENDIF
C
C  Get interpolated points using KURV2.
C
      TINC = 1.0/(IOUT-1)
      DO 10 I=1,IOUT
        U(I) = (I-1)*TINC
        CALL KURV2(U(I),XO(I),YO(I),IDIM,X,Y,XP,YP,S,SIGMA)
   10 CONTINUE
C
C  Get the derivatives.
C
      DO 20 I=1,IOUT
        CALL KURVD(U(I),XS(I),YS(I),XD(I),YD(I),XDD(I),YDD(I),
     +             IDIM,X,Y,XP,YP,S,SIGMA)
   20 CONTINUE
C
C  Draw plot.
C
      CALL DRWFT4(IDIM,X,Y,IOUT,XO,YO,U,XD,YD)
C
      STOP
      END
      SUBROUTINE DRWFT4(II,X,Y,IOUT,XO,YO,U,XD,YD)
C
C  Define error file, Fortran unit number, workstation type,
C  and workstation ID.
C
      PARAMETER (IERRF=6, LUNIT=2, IWTYPE=8, IWKID=1)
C
C  Open GKS, open and activate a workstation.
C
      CALL GOPKS (IERRF, ISZDM)
      CALL GOPWK (IWKID, LUNIT, IWTYPE)
      CALL GACWK (IWKID)
C
C  Define a color table.
C
      CALL GSCR(IWKID, 0, 1.0, 1.0, 1.0)
      CALL GSCR(IWKID, 1, 0.0, 0.0, 0.0)
      CALL GSCR(IWKID, 2, 1.0, 0.0, 0.0)
      CALL GSCR(IWKID, 3, 0.0, 1.0, 0.0)
      CALL GSCR(IWKID, 4, 0.0, 0.0, 1.0)
C
C  Draw markers at original points.
C
      CALL BKGFT4(0.,40.,0.,40.,0.15,0.85,'Demo for KURV1/KURV2',0.035,
     +            0.5,0.93,0)
      CALL GRIDAL(4,5,4,5,1,1,10,0.,0.)
      CALL GSMKSC(2.)
      CALL GSPMCI(4)
      CALL GPM(II,X,Y)
C
C  Draw the interpolated curve.
C
      CALL CURVE(XO,YO,IOUT)
      CALL FRAME
C
```

```
C  Plot the first derivatives of X and Y with respect to the parametric
C  variable U.
C
      CALL SET(0.,1.,0.,1.,0.,1.,0.,1.,1)
      CALL PCSETI('FN',21)
      CALL PLCHHQ(0.5,0.95,'Derivatives from KURVD',0.035,0.,0.)
      CALL BKGFT4(0.,1.,-80.,80.,0.55,0.87,'dx/du',0.030,0.65,0.82,1)
      CALL GRIDAL(5,5,4,5,1,1,10,0.,-80.)
      CALL CURVE(U,XD,IOUT)
      CALL BKGFT4(0.,1.,-40.,80.,0.10,0.42,'dy/du',0.030,0.39,0.37,1)
      CALL GRIDAL(5,5,3,5,1,1,10,0.,-40.)
      CALL CURVE(U,YD,IOUT)
      CALL FRAME
C
      CALL GDAWK (IWKID)
      CALL GCLWK (IWKID)
      CALL GCLKS
C
      RETURN
      END
      SUBROUTINE BKGFT4(XL,XR,YB,YT,YPB,YPT,LABEL,SIZL,POSXL,POSYL,IZL)
      DIMENSION XX(2),YY(2)
      CHARACTER*(*) LABEL
C
      CALL SET(0.,1.,0.,1.,0.,1.,0.,1.,1)
      CALL PCSETI('FN',21)
      CALL PLCHHQ(POSXL,POSYL,LABEL,SIZL,0.,0.)
      CALL SET(0.17,0.87,YPB,YPT,XL,XR,YB,YT,1)
      IF (IZL .NE. 0) THEN
        XX(1) = XL
        XX(2) = XR
        YY(1) = 0.
        YY(2) = 0.
        CALL GSPLCI(2)
        CALL GPL(2,XX,YY)
        CALL GSPLCI(1)
      ENDIF
C
      CALL GASETI('LTY',1)
      CALL PCSETI('FN',21)
      CALL GASETR('XLS',0.02)
      CALL GASETC('XLF','(F4.1)')
      CALL GASETR('YLS',0.02)
      CALL GASETC('YLF','(F5.1)')
      CALL GASETR('XMJ',0.02)
      CALL GASETR('YMJ',0.02)
C
      RETURN
      END
```

Example 4 -- interpolated parametric curves and derivatives

```
#include <stdio.h>
#include <ncarg/ncargC.h>
#include <ncarg/gks.h>
#include <ncarg/ngmath.h>

void c_drwft4(int, float [], float [], int,
              float [], float [], float [], float [], float []);
void c_bkgft4(float, float, float, float, float, float,
              char *, float, float, float, int);

/*
 *  Example of kurv1, kurv2, kurvd.
 */

#define IDIM  11
#define IOUT 201

#define IWTYPE 1
#define WKID   1

main()
{
  float x[] = {  3.0,  4.0,  9.0, 16.0, 21.0, 27.0,
                34.0, 36.0, 34.0, 26.0, 18.0       };
  float y[] = {  2.4,  9.6, 14.4, 12.0,  9.6,  8.4,
                13.2, 21.6, 30.0, 37.2, 38.4       };
  float xinc, xo[IOUT], yo[IOUT], xs[IOUT], ys[IOUT],
              xd[IOUT], yd[IOUT], xdd[IOUT], ydd[IOUT],
              xp[IDIM], yp[IDIM], u[IOUT];
  float tinc;
  int   i;

/*
 *  Set up the array of parameter values where we want to obtain
 *  interpolated values.
 */
  tinc = 1./(IOUT-1.);
  for (i = 0; i < IOUT; i++) {
    u[i] = tinc * (float) i;
  }

/*
 *  Get the interpolated points.
 */
  c_ftkurv(IDIM, x, y, IOUT, u, xo, yo);

/*
 *  Get the derivatives (this returns the interpolated values as well).
 */
  c_ftkurvd(IDIM, x, y, IOUT, u, xs, ys, xd, yd, xdd, ydd);

/*
 *  Draw plot.
 */
  c_drwft4(IDIM, x, y, IOUT, xo, yo, u, xd, yd);

}

void c_drwft4(int n, float x[], float y[], int m,
              float xo[], float yo[], float u[], float xd[], float yd[])
{
```

```
  int i;

  Gcolr_rep rgb;
  Gpoint plist[IDIM];
  Gpoint_list pmk;

/*
 *  Open GKS, open and activate a workstation.
 */
  gopen_gks("stdout",0);
  gopen_ws(WKID, NULL, IWTYPE);
  gactivate_ws(WKID);

/*
 * Define a color table.
 */
  rgb.rgb.red = rgb.rgb.green = rgb.rgb.blue = 1.;
  gset_colr_rep(WKID,0,&rgb);
  rgb.rgb.red = rgb.rgb.green = rgb.rgb.blue = 0.;
  gset_colr_rep(WKID,1,&rgb);
  rgb.rgb.red = 1.;
  rgb.rgb.green = rgb.rgb.blue = 0.;
  gset_colr_rep(WKID,2,&rgb);
  rgb.rgb.red = rgb.rgb.green = 0.;
  rgb.rgb.blue = 1.;
  gset_colr_rep(WKID,3,&rgb);

/*
 * Graph the interpolated function values and mark the original
 * input data points.
 */
  c_bkgft4(0.,40.,0.,40.,0.15,0.85,"Demo for c_ftkurv",0.03,0.5,0.93,0);
  c_gridal(4,5,4,5,1,1,10,0.,0.);
  c_curve(xo,yo,m);

/*
 *  Mark the input data points.
 */
  for (i = 0; i < n; i++) {
    plist[i].x = x[i];
    plist[i].y = y[i];
  }
  gset_marker_size(2.);
  gset_marker_colr_ind(3);
  pmk.num_points = n;
  pmk.points = plist;
  gpolymarker(&pmk);
  c_frame();

/*
 *  Plot the first derivatives.
 */
  c_set(0.,1.,0.,1.,0.,1.,0.,1.,1);
  c_pcseti("fn",21);
  c_plchhq(0.5,0.95,"Derivatives from c_ftkurvd",0.035,0.,0.);
  c_bkgft4(0.,1.,-80.,80.,0.55,0.87,"dx/du",0.030,0.65,0.82,1);
  c_gridal(5,5,4,5,1,1,10,0.,-80.);
  c_curve(u,xd,m);
  c_bkgft4(0.,1.,-40.,80.,0.1,0.42,"dy/du",0.030,0.39,0.37,1);
  c_gridal(5,5,3,5,1,1,10,0.,-40.);
  c_curve(u,yd,m);
```

```
  c_frame();

/*
 *  Deactivate and close workstation, close GKS.
 */
  gdeactivate_ws(WKID);
  gclose_ws(WKID);
  gclose_gks();
}

void c_bkgft4(float xleft, float xright, float ybot, float ytop,
              float ypos_bot, float ypos_top, char *label, float lab_size,
              float lab_posx, float lab_posy, int zero_line) {
  c_set(0.,1.,0.,1.,0.,1.,0.,1.,1);
  c_pcseti("fn",21);
  c_plchhq(lab_posx,lab_posy,label,lab_size,0.,0.0);
  c_set(0.17,0.87,ypos_bot,ypos_top,xleft,xright, ybot, ytop, 1);
  if (zero_line != 0) {
     gset_line_colr_ind(2);
     c_line(xleft, 0., xright, 0.);
     c_sflush();
     gset_line_colr_ind(1);
  }
  c_gaseti("lty",1);
  c_pcseti("fn",21);
  c_gasetr("xls",0.02);
  c_gasetc("xlf","(f4.1)");
  c_gasetr("yls",0.02);
  c_gasetc("ylf","(f5.1)");
  c_gasetr("xmj",0.02);
  c_gasetr("ymj",0.02);
}
```

Example 4 -- interpolated parametric curves and derivatives

```
;
;  Description:  This program illustrates the use of ftkurv and ftkurvd.
;
procedure drw_text(label:string, xpos:float, ypos:float, height:float, \
                   output:graphic)
begin
;
; Create and draw a text string.
;
label1 = create "Label" textItemClass output
    "txPosXF": xpos
    "txPosYF": ypos
    "txFont": 21
    "txJust" : "centercenter"
    "txString": label
    "txFontHeightF": height
end create
```

```
draw(label1)
destroy(label1)

end

;
; Main program.
;
begin
;
; Specify the input data.
;
x = (/  3.0,  4.0,  9.0, 16.0, 21.0, 27.0,    \
       34.0, 36.0, 34.0, 26.0, 18.0          /)
y = (/  2.4,  9.6, 14.4, 12.0,  9.6,  8.4,    \
       13.2, 21.6, 30.0, 37.2, 38.4          /)

;
; Create an array of parameterized values where interpolated values
; are desired.
;
npts = 201
u = fspan(0.,1.,npts)

;
; Interpolate.
;
xo = new( (/ npts /), float)
yo = new( (/ npts /), float)
ftkurv(x, y, u, xo, yo)

;
;  Get the derivatives (this returns the interpolated values as well).
;
xs = new( (/ npts /), float)
ys = new( (/ npts /), float)
xd = new( (/ npts /), float)
yd = new( (/ npts /), float)
xdd = new( (/ npts /), float)
ydd = new( (/ npts /), float)
ftkurvd(x, y, u, xs, ys, xd, yd, xdd, ydd)

;
; Create a color map.
;
ncolors = 6
cmap = new( (/ncolors,3/), float)
cmap(0,:) = (/1.0,1.0,1.0/)
cmap(1,:) = (/0.0,0.0,0.0/)
cmap(2,:) = (/1.0,0.0,0.0/)
cmap(3,:) = (/0.0,1.0,0.0/)
cmap(4,:) = (/0.0,0.0,1.0/)
cmap(5,:) = (/1.0,1.0,0.0/)

;
; Default is to display output to an X workstation.
;
NCGM=0
X11=1
PS=0
```

```
if (NCGM .eq. 1) then
;
; Create an ncgmWorkstation object.
;
  xworkid = create "ft04Work" ncgmWorkstationClass defaultapp
      "wkMetaName" : "ft01n.ncgm"
      "wkColorMap" : cmap
  end create
else
  if (X11 .eq. 1) then
;
; Create an XWorkstation object.
;
  xworkid = create "ft01Work" xWorkstationClass defaultapp
      "wkPause" : "True"
      "wkColorMap" : cmap
  end create
  else
    if (PS .eq. 1) then
;
; Create a PSWorkstation object.
;
      xworkid = create "ft01Work" psWorkstationClass defaultapp
        "wkPSFileName" : "ft01n.ps"
        "wkColorMap" : cmap
      end create
    end if
  end if
end if

;
;
; Draw two graphs:
;   1.)  The interpolated function values.
;   2.)  The first derivatives with respect to the parameter variable.
;

;
; Define data objects:
;   dataid1(0) - coordinates of the original points.
;   dataid1(1) - coordinates of the interpolated points.
;   dataid2(0) - zero line for derivative plots.
;   dataid2(1) - coordinates of the first derivative of x wrt u.
;   dataid3(0) - zero line for derivative plots.
;   dataid3(1) - coordinates of the first derivative of y wrt u.
;
dataid1 = new(2,graphic)
dataid2 = new(2,graphic)
dataid3 = new(2,graphic)

; Define a data object with the original points.  The id for this
; object will later be used as the value for an XYPlot data resource,
; 'xyCoordData'.
;
dataid1(0) = create "OriginalPoints" coordArraysClass defaultapp
    "caXArray": x
    "caYArray": y
end create

;
; Define a data object with the interpolated points.  The id for this
```

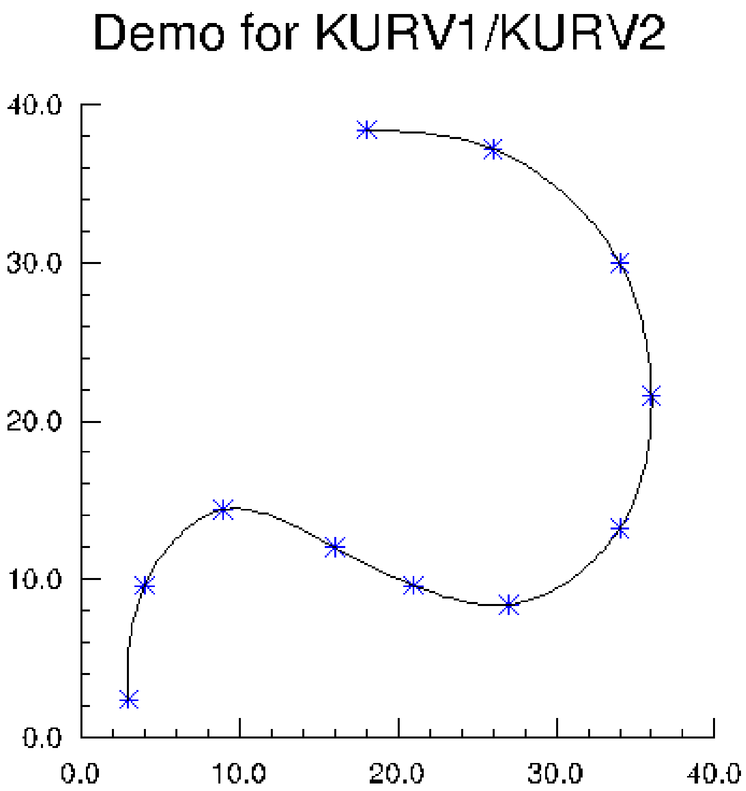
```
; object will later be used as the value for an XYPlot data resource,
; 'xyCoordData'.
;
dataid1(1) = create "InterpolatedPoints" coordArraysClass defaultapp
    "caXArray": xo
    "caYArray": yo
end create

;
; Define a data object to mark the zero line in the derivative plots.
;
xz = new(2,float)
yz = new(2,float)
xz(0) = 0.
xz(1) = 1.
yz(0) = 0.
yz(1) = 0.
dataid2(0) = create "ZeroLine" coordArraysClass defaultapp
    "caXArray": xz
    "caYArray": yz
end create
dataid3(0) = dataid2(0)

;
; Define a data object with the derivatives of X wrt the parameter
; variable.
;
dataid2(1) = create "XDerivatives" coordArraysClass defaultapp
    "caXArray": u
    "caYArray": xd
end create

;
; Define a data object with the derivatives of Y wrt the parameter
; variable.
;
dataid3(1) = create "YDerivatives" coordArraysClass defaultapp
    "caXArray": u
    "caYArray": yd
end create

;
; Create an XyPlot object for drawing a curve with the interpolated
; points.
;
plotid_function = create "xyPlot" xyPlotClass xworkid
    "vpXF": .15
    "vpYF": .85
    "vpWidthF" :  .7
    "vpHeightF" : .7
    "trYMaxF" :   40.
    "trYMinF" :   0.
    "trXMaxF" :   40.
    "trXMinF" :   0.
    "xyCoordData": dataid1
    "tmXTBorderOn": "False"
    "tmYRBorderOn": "False"
    "tmXBMinorPerMajor": 5
    "tmYLMinorPerMajor": 5
    "tmBorderThicknessF": 1.
    "tmXMajorGridThicknessF": 1.
```

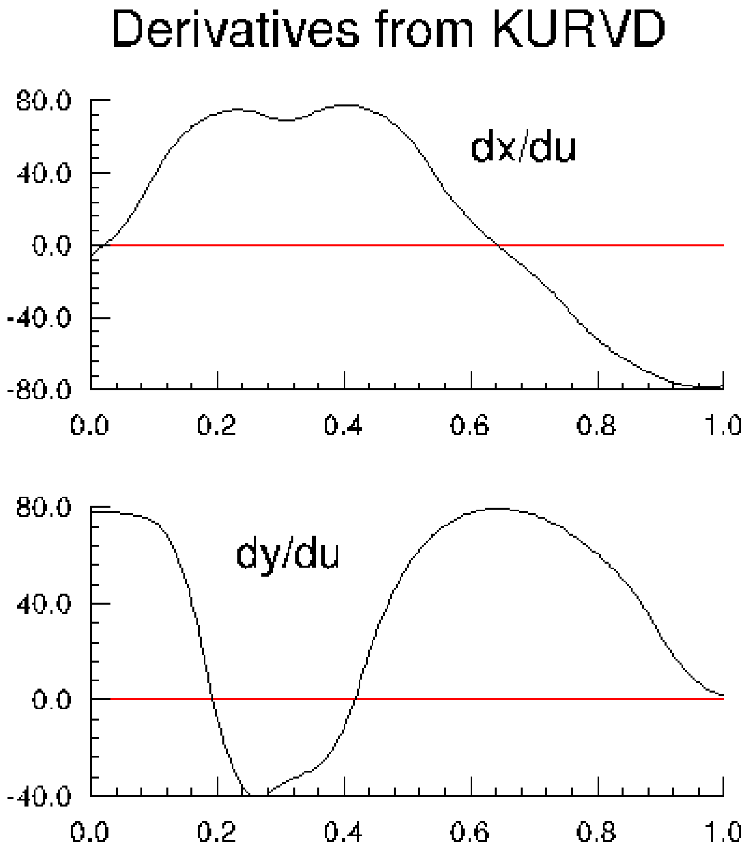
```
    "tmXBMajorLengthF": 0.015
    "tmYLMajorLengthF": 0.015
    "tmXBMinorLengthF": 0.0075
    "tmXBMajorThicknessF": 1.0
    "tmYLMajorThicknessF": 1.0
    "tmXTOn": "False"
    "tmYROn": "False"
    "tmXBLabelFont": 21
    "tmXBLabelFontHeightF": 0.025
    "tmYLLabelFont": 21
    "tmYLLabelFontHeightF": 0.025
    "tmYLMode": "Manual"
    "tmYLTickStartF": 0.
    "tmYLTickSpacingF": 10.0
    "tmYLTickEndF": 40.
    "tmXBMode": "Manual"
    "tmXBTickStartF": 0.
    "tmXBTickSpacingF": 10.0
    "tmXBTickEndF": 40.
end create

getvalues plotid_function
    "xyCoordDataSpec": dataspec
end getvalues

setvalues dataspec(0)
    "xyMarkLineMode": "markers"
    "xyMarkerColor": 4
    "xyMarkerSizeF": 0.025
end setvalues

;
; Draw a graph of the interpolated points.
;
draw(plotid_function)
drw_text("Demo for ftkurv",0.5,0.93,0.04,xworkid)
frame(xworkid)

;
; Draw a graph of the X derivative.
;
setvalues plotid_function
    "trYMaxF" :   80.
    "trYMinF" :  -80.
    "trXMaxF" :   1.
    "trXMinF" :   0.
    "xyCoordData": dataid2
    "tmYLTickStartF": -80.
    "tmYLTickSpacingF": 40.0
    "tmYLTickEndF": 80.
    "tmXBTickStartF": 0.
    "tmXBTickSpacingF": 0.2
    "tmXBTickEndF": 1.
    "tmXBMajorLengthF": 0.020
    "tmXBMinorLengthF": 0.010
    "tmYLMajorLengthF": 0.020
    "tmYLMinorLengthF": 0.010
    "tmXBMinorPerMajor": 4
    "tmYLMinorPerMajor": 4
    "vpXF": .15
    "vpYF": .87
```

```
    "vpWidthF" :  .7
    "vpHeightF" : .32
    "tmYLLabelFontHeightF": 0.023
    "tmXBLabelFontHeightF": 0.023
end setvalues
getvalues plotid_function
    "xyCoordDataSpec": dataspec
end getvalues
setvalues dataspec(0)
    "xyLineColor": 2
end setvalues
draw(plotid_function)

;
; Draw a graph of the Y derivative.
;
setvalues plotid_function
    "trYMinF" :  -40.
    "xyCoordData": dataid3
    "tmYLTickStartF": -40.
    "vpXF": .15
    "vpYF": .42
    "vpHeightF" : .32
end setvalues
getvalues plotid_function
    "xyCoordDataSpec": dataspec
end getvalues
setvalues dataspec(0)
    "xyLineColor": 2
end setvalues

drw_text("Derivatives from ftcurvd",0.5,0.95,0.04,xworkid)
drw_text("dx/du",0.65,0.82,0.033,xworkid)
drw_text("dy/du",0.39,0.37,0.033,xworkid)
draw(plotid_function)
frame(xworkid)

;
; End NCL script.
;
end
```

Plots for example 4 -- interpolated parametric curves and derivatives

---


Demo for KURV1/KURV2
40.0
30.0
20.0
10.0
0.0
0.0
10.0
20.0
30.0
40.0



Example 5 -- interpolated closed parametric curves and derivatives

```
      PROGRAM FTEX05
C
C  Example of kurvp1, kurvp2, kurvpd.
C
      PARAMETER (IDIM=11,IOUT=201,IDTEMP=2*IDIM)
      DIMENSION X(IDIM),Y(IDIM),TEMP(IDTEMP),U(IOUT),XO(IOUT),YO(IOUT),
     +          XS(IOUT),YS(IOUT),XD(IOUT),YD(IOUT),XDD(IOUT),YDD(IOUT)
      DIMENSION XP(IDIM),YP(IDIM),S(IDIM)
C
      DATA X/13.0,  9.0,  9.0, 16.0, 21.0, 27.0,
     +       34.0, 36.0, 34.0, 26.0, 19.5/
      DATA Y/35.0, 31.0, 18.0, 12.0,  9.6,  8.4,
     +       13.2, 21.6, 30.0, 37.2, 37.4/
C
C  Do KURVP1 setup.
C
      SIGMA = 1.
      ISF   = 3
```

```
      CALL KURVP1(IDIM,X,Y,XP,YP,TEMP,S,SIGMA,IERR)
      IF (IERR .NE. 0) THEN
        PRINT *, 'Error return from KURVP1 =',IERR
        STOP
      ENDIF
C
C  Get interpolated points using KURV2.
C
      TINC = 1.0/(IOUT-1)
      DO 10 I=1,IOUT
        U(I) = (I-1)*TINC
        CALL KURVP2(U(I),XO(I),YO(I),IDIM,X,Y,XP,YP,S,SIGMA)
   10 CONTINUE
C
C  Get the derivatives.
C
      DO 20 I=1,IOUT
        CALL KURVPD(U(I),XS(I),YS(I),XD(I),YD(I),XDD(I),YDD(I),
     +              IDIM,X,Y,XP,YP,S,SIGMA)
   20 CONTINUE
C
C  Draw plot.
C
      CALL DRWFT5(IDIM,X,Y,IOUT,XO,YO,U,XD,YD)
C
      STOP
      END
      SUBROUTINE DRWFT5(II,X,Y,IO,XO,YO,U,XD,YD)
C
C  Define error file, Fortran unit number, workstation type,
C  and workstation ID.
C
      PARAMETER (IERRF=6, LUNIT=2, IWTYPE=1, IWKID=1)
C
C  Open GKS, open and activate a workstation.
C
      CALL GOPKS (IERRF, ISZDM)
      CALL GOPWK (IWKID, LUNIT, IWTYPE)
      CALL GACWK (IWKID)
C
C  Define a color table.
C
      CALL GSCR(IWKID, 0, 1.0, 1.0, 1.0)
      CALL GSCR(IWKID, 1, 0.0, 0.0, 0.0)
      CALL GSCR(IWKID, 2, 1.0, 0.0, 0.0)
      CALL GSCR(IWKID, 3, 0.0, 1.0, 0.0)
      CALL GSCR(IWKID, 4, 0.0, 0.0, 1.0)
C
C  Draw markers at original points.
C
      CALL BKGFT5(0.,40.,0.,40.,0.15,0.85,'Demo for KURVP1/KURVP2',
     +            0.035,0.5,0.93,0)
      CALL GRIDAL(4,5,4,5,1,1,10,0.,0.)
      CALL GSMKSC(2.)
      CALL GSPMCI(4)
      CALL GPM(II,X,Y)
C
C  Draw the interpolated curve.
C
      CALL CURVE(XO,YO,IO)
      CALL FRAME
```

```
C
C  Plot the first derivatives of X and Y with respect to the parametric
C  variable U.
C
      CALL SET(0.,1.,0.,1.,0.,1.,0.,1.,1)
      CALL PCSETI('FN',21)
      CALL PLCHHQ(0.5,0.95,'Derivatives from KURVPD',0.035,0.,0.)
      CALL BKGFT5(0.,1.,-100.,100.,0.55,0.87,'dx/du',0.030,0.65,0.82,1)
      CALL GRIDAL(5,5,4,5,1,1,10,0.,-100.)
      CALL CURVE(U,XD,IO)
      CALL BKGFT5(0.,1.,-100.,100.,0.10,0.42,'dy/du',0.030,0.39,0.37,1)
      CALL GRIDAL(5,5,4,5,1,1,10,0.,-100.)
      CALL CURVE(U,YD,IO)
      CALL FRAME
C
      CALL GDAWK (IWKID)
      CALL GCLWK (IWKID)
      CALL GCLKS
C
      STOP
      END
      SUBROUTINE BKGFT5(XL,XR,YB,YT,YPB,YPT,LABEL,SIZL,POSXL,POSYL,IZL)
      DIMENSION XX(2),YY(2)
      CHARACTER*(*) LABEL
C
      CALL SET(0.,1.,0.,1.,0.,1.,0.,1.,1)
      CALL PCSETI('FN',21)
      CALL PLCHHQ(POSXL,POSYL,LABEL,SIZL,0.,0.)
      CALL SET(0.17,0.87,YPB,YPT,XL,XR,YB,YT,1)
      IF (IZL .NE. 0) THEN
        XX(1) = XL
        XX(2) = XR
        YY(1) = 0.
        YY(2) = 0.
        CALL GSPLCI(2)
        CALL GPL(2,XX,YY)
        CALL GSPLCI(1)
      ENDIF
C
      CALL GASETI('LTY',1)
      CALL PCSETI('FN',21)
      CALL GASETR('XLS',0.02)
      CALL GASETC('XLF','(F4.1)')
      CALL GASETR('YLS',0.02)
      CALL GASETC('YLF','(F6.1)')
      CALL GASETR('XMJ',0.02)
      CALL GASETR('YMJ',0.02)
C
      RETURN
      END
```

Example 5 -- interpolated closed parametric curves and derivatives

```
#include <stdio.h>
#include <ncarg/ncargC.h>
#include <ncarg/gks.h>
#include <ncarg/ngmath.h>

void c_drwft5(int, float [], float [], int,
              float [], float [], float [], float [], float []);
void c_bkgft5(float, float, float, float, float, float,
              char *, float, float, float, int);

/*
 *  Example of kurvp1, kurvp2, kurvpd.
 */

#define IDIM  11
#define IOUT 201

#define IWTYPE 1
#define WKID   1

main()
{
  float x[] = { 13.0,  9.0,  9.0, 16.0, 21.0, 27.0,
                34.0, 36.0, 34.0, 26.0, 19.5       };
  float y[] = { 35.0, 31.0, 18.0, 12.0,  9.6,  8.4,
                13.2, 21.6, 30.0, 37.2, 37.4       };
  float xinc, xo[IOUT], yo[IOUT], xs[IOUT], ys[IOUT],
              xd[IOUT], yd[IOUT], xdd[IOUT], ydd[IOUT],
              xp[IDIM], yp[IDIM], u[IOUT];
  float tinc;
  int   i;

/*
 *  Set up the array of parameter values where we want to obtain
 *  interpolated values.
 */
  tinc = 1./(IOUT-1.);
  for (i = 0; i < IOUT; i++) {
    u[i] = tinc * (float) i;
  }

/*
 *  Get the interpolated points.
 */
  c_ftkurvp(IDIM, x, y, IOUT, u, xo, yo);

/*
 *  Get the derivatives (this returns the interpolated values as well).
 */
  c_ftkurvpd(IDIM, x, y, IOUT, u, xs, ys, xd, yd, xdd, ydd);

/*
 *  Draw plots.
 */
  c_drwft5(IDIM, x, y, IOUT, xo, yo, u, xd, yd);
}

void c_drwft5(int n, float x[], float y[], int m,
              float xo[], float yo[], float u[], float xd[], float yd[])
{
```

```
  int i;

  Gcolr_rep rgb;
  Gpoint plist[IDIM];
  Gpoint_list pmk;

/*
 *  Open GKS, open and activate a workstation.
 */
  gopen_gks("stdout",0);
  gopen_ws(WKID, NULL, IWTYPE);
  gactivate_ws(WKID);

/*
 * Define a color table.
 */
  rgb.rgb.red = rgb.rgb.green = rgb.rgb.blue = 1.;
  gset_colr_rep(WKID,0,&rgb);
  rgb.rgb.red = rgb.rgb.green = rgb.rgb.blue = 0.;
  gset_colr_rep(WKID,1,&rgb);
  rgb.rgb.red = 1.;
  rgb.rgb.green = rgb.rgb.blue = 0.;
  gset_colr_rep(WKID,2,&rgb);
  rgb.rgb.red = rgb.rgb.green = 0.;
  rgb.rgb.blue = 1.;
  gset_colr_rep(WKID,3,&rgb);

/*
 * Graph the interpolated function values and mark the original
 * input data points.
 */
  c_bkgft5(0.,40.,0.,40.,0.15,0.85,"Demo for c_ftkurvp",0.035,0.5,0.93,0);
  c_gridal(4,5,4,5,1,1,10,0.,0.);
  c_curve(xo,yo,m);

/*
 *  Mark the input data points.
 */
  for (i = 0; i < n; i++) {
    plist[i].x = x[i];
    plist[i].y = y[i];
  }
  gset_marker_size(2.);
  gset_marker_colr_ind(3);
  pmk.num_points = n;
  pmk.points = plist;
  gpolymarker(&pmk);
  c_frame();

/*
 *  Plot the first derivatives.
 */
  c_set(0.,1.,0.,1.,0.,1.,0.,1.,1);
  c_pcseti("fn",21);
  c_plchhq(0.5,0.95,"Derivatives from c_ftkurvpd",0.035,0.,0.);
  c_bkgft5(0.,1.,-100.,100.,0.55,0.87,"dx/du",0.030,0.70,0.82,1);
  c_gridal(5,5,4,5,1,1,10,0.,-100.);
  c_curve(u,xd,m);
  c_bkgft5(0.,1.,-100.,100.,0.1,0.42,"dy/du",0.030,0.34,0.37,1);
  c_gridal(5,5,4,5,1,1,10,0.,-100.);
  c_curve(u,yd,m);
```

```
  c_frame();

/*
 *  Deactivate and close workstation, close GKS.
 */
  gdeactivate_ws(WKID);
  gclose_ws(WKID);
  gclose_gks();
}

void c_bkgft5(float xleft, float xright, float ybot, float ytop,
              float ypos_bot, float ypos_top, char *label, float lab_size,
              float lab_posx, float lab_posy, int zero_line) {
  c_set(0.,1.,0.,1.,0.,1.,0.,1.,1);
  c_pcseti("fn",21);
  c_plchhq(lab_posx,lab_posy,label,lab_size,0.,0.0);
  c_set(0.17,0.87,ypos_bot,ypos_top,xleft,xright, ybot, ytop, 1);
  if (zero_line != 0) {
     gset_line_colr_ind(2);
     c_line(xleft, 0., xright, 0.);
     c_sflush();
     gset_line_colr_ind(1);
  }
  c_gaseti("lty",1);
  c_pcseti("fn",21);
  c_gasetr("xls",0.02);
  c_gasetc("xlf","(f4.1)");
  c_gasetr("yls",0.02);
  c_gasetc("ylf","(f6.1)");
  c_gasetr("xmj",0.02);
  c_gasetr("ymj",0.02);
}
```

Example 5 -- interpolated closed parametric curves and derivatives

---

```
;
;  Description:  This program illustrates the use of ftkurvp and ftkurvpd.
;
procedure drw_text(label:string, xpos:float, ypos:float, height:float, \
                   output:graphic)
begin
;
; Create a label for the graph of function values.
;
label1 = create "Label" textItemClass output
    "txPosXF": xpos
    "txPosYF": ypos
    "txFont": 21
    "txJust" : "centercenter"
    "txString": label
    "txFontHeightF": height
end create
```

```
draw(label1)
destroy(label1)

end

;
; Main program.
;
begin
;
; Specify the input data.
;
x = (/ 13.0,  9.0,  9.0, 16.0, 21.0, 27.0,    \
       34.0, 36.0, 34.0, 26.0, 19.5         /)
y = (/ 35.0, 31.0, 18.0, 12.0,  9.6,  8.4,    \
       13.2, 21.6, 30.0, 37.2, 37.4         /)

;
; Create an array of parameterized values where interpolated values
; are desired.
;
npts = 201
u = fspan(0.,1.,npts)

;
; Interpolate.
;
xo = new( (/ npts /), float)
yo = new( (/ npts /), float)
ftkurvp(x, y, u, xo, yo)

;
;  Get the derivatives (this returns the interpolated values as well).
;
xs = new( (/ npts /), float)
ys = new( (/ npts /), float)
xd = new( (/ npts /), float)
yd = new( (/ npts /), float)
xdd = new( (/ npts /), float)
ydd = new( (/ npts /), float)
ftkurvpd(x, y, u, xs, ys, xd, yd, xdd, ydd)

;
; Create a color map.
;
ncolors = 6
cmap = new( (/ncolors,3/), float)
cmap(0,:) = (/1.0,1.0,1.0/)
cmap(1,:) = (/0.0,0.0,0.0/)
cmap(2,:) = (/1.0,0.0,0.0/)
cmap(3,:) = (/0.0,1.0,0.0/)
cmap(4,:) = (/0.0,0.0,1.0/)
cmap(5,:) = (/1.0,1.0,0.0/)

;
; Default is to display output to an X workstation.
;
NCGM=0
X11=1
PS=0
```

```
if (NCGM .eq. 1) then
;
; Create an ncgmWorkstation object.
;
  xworkid = create "ft04Work" ncgmWorkstationClass defaultapp
      "wkMetaName" : "ft01n.ncgm"
      "wkColorMap" : cmap
  end create
else
  if (X11 .eq. 1) then
;
; Create an XWorkstation object.
;
  xworkid = create "ft01Work" xWorkstationClass defaultapp
      "wkPause" : "True"
      "wkColorMap" : cmap
  end create
  else
    if (PS .eq. 1) then
;
; Create a PSWorkstation object.
;
      xworkid = create "ft01Work" psWorkstationClass defaultapp
        "wkPSFileName" : "ft01n.ps"
        "wkColorMap" : cmap
      end create
    end if
  end if
end if

;
;
; Draw two graphs:
;   1.)  The interpolated function values.
;   2.)  The first derivatives with respect to the parameter variable.
;

;
; Define data objects:
;   dataid1(0) - coordinates of the original points.
;   dataid1(1) - coordinates of the interpolated points.
;   dataid2(0) - zero line for derivative plots.
;   dataid2(1) - coordinates of the first derivative of x wrt u.
;   dataid3(0) - zero line for derivative plots.
;   dataid3(1) - coordinates of the first derivative of y wrt u.
;
dataid1 = new(2,graphic)
dataid2 = new(2,graphic)
dataid3 = new(2,graphic)

; Define a data object with the original points.  The id for this
; object will later be used as the value for an XYPlot data resource,
; 'xyCoordData'.
;
dataid1(0) = create "OriginalPoints" coordArraysClass defaultapp
    "caXArray": x
    "caYArray": y
end create

;
; Define a data object with the interpolated points.  The id for this
```

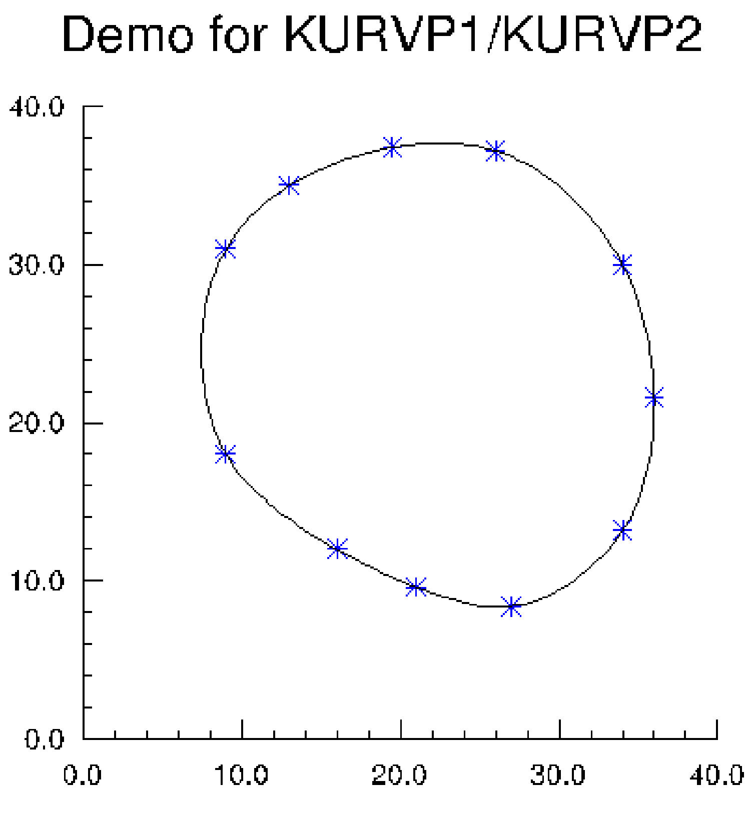
```
; object will later be used as the value for an XYPlot data resource,
; 'xyCoordData'.
;
dataid1(1) = create "InterpolatedPoints" coordArraysClass defaultapp
    "caXArray": xo
    "caYArray": yo
end create

;
; Define a data object to mark the zero line in the derivative plots.
;
xz = new(2,float)
yz = new(2,float)
xz(0) = 0.
xz(1) = 1.
yz(0) = 0.
yz(1) = 0.
dataid2(0) = create "ZeroLine" coordArraysClass defaultapp
    "caXArray": xz
    "caYArray": yz
end create
dataid3(0) = dataid2(0)

;
; Define a data object with the derivatives of X wrt the parameter
; variable.
;
dataid2(1) = create "XDerivatives" coordArraysClass defaultapp
    "caXArray": u
    "caYArray": xd
end create

;
; Define a data object with the derivatives of Y wrt the parameter
; variable.
;
dataid3(1) = create "YDerivatives" coordArraysClass defaultapp
    "caXArray": u
    "caYArray": yd
end create

;
; Create an XyPlot object for drawing a curve with the interpolated
; points.
;
plotid_function = create "xyPlot" xyPlotClass xworkid
    "vpXF": .15
    "vpYF": .85
    "vpWidthF" :  .7
    "vpHeightF" : .7
    "trYMaxF" :   40.
    "trYMinF" :   0.
    "trXMaxF" :   40.
    "trXMinF" :   0.
    "xyCoordData": dataid1
    "tmXTBorderOn": "False"
    "tmYRBorderOn": "False"
    "tmXBMinorPerMajor": 5
    "tmYLMinorPerMajor": 5
    "tmBorderThicknessF": 1.
    "tmXMajorGridThicknessF": 1.
```

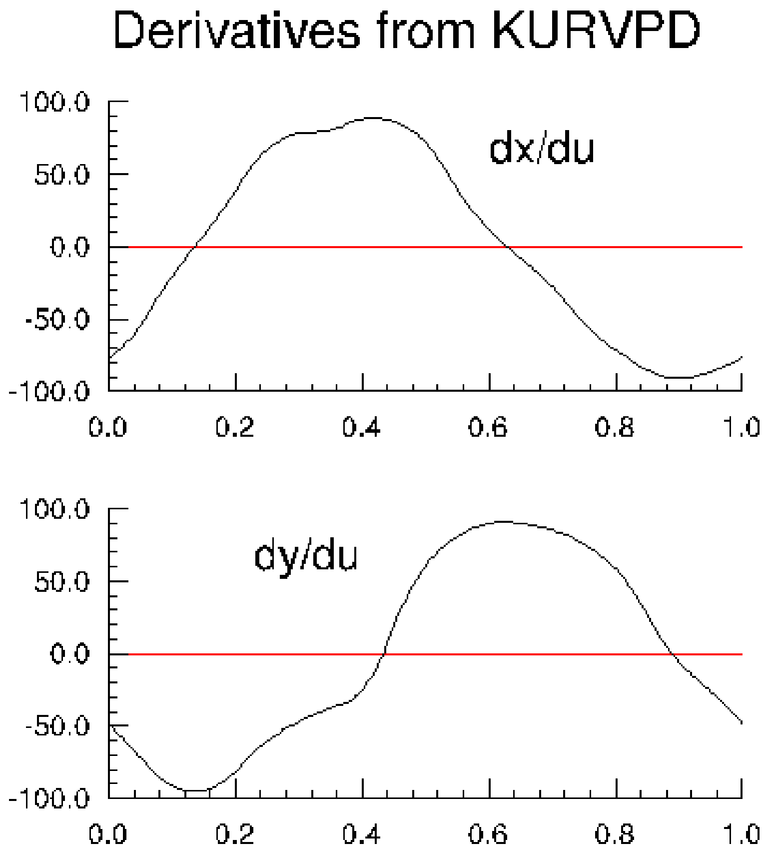
```
    "tmXBMajorLengthF": 0.015
    "tmYLMajorLengthF": 0.015
    "tmXBMinorLengthF": 0.0075
    "tmXBMajorThicknessF": 1.0
    "tmYLMajorThicknessF": 1.0
    "tmXTOn": "False"
    "tmYROn": "False"
    "tmXBLabelFont": 21
    "tmXBLabelFontHeightF": 0.025
    "tmYLLabelFont": 21
    "tmYLLabelFontHeightF": 0.025
    "tmYLMode": "Manual"
    "tmYLTickStartF": 0.
    "tmYLTickSpacingF": 10.0
    "tmYLTickEndF": 40.
    "tmXBMode": "Manual"
    "tmXBTickStartF": 0.
    "tmXBTickSpacingF": 10.0
    "tmXBTickEndF": 40.
end create

getvalues plotid_function
    "xyCoordDataSpec": dataspec
end getvalues

setvalues dataspec(0)
    "xyMarkLineMode": "markers"
    "xyMarkerColor": 4
    "xyMarkerSizeF": 0.025
end setvalues

;
; Draw a graph of the interpolated points.
;
draw(plotid_function)
drw_text("Demo for ftkurvp",0.5,0.93,0.04,xworkid)
frame(xworkid)

;
; Draw a graph of the X derivative.
;
setvalues plotid_function
    "trYMaxF" :   100.
    "trYMinF" :  -100.
    "trXMaxF" :   1.
    "trXMinF" :   0.
    "xyCoordData": dataid2
    "tmYLTickStartF": -100.
    "tmYLTickSpacingF": 50.0
    "tmYLTickEndF": 100.
    "tmXBTickStartF": 0.
    "tmXBTickSpacingF": 0.2
    "tmXBTickEndF": 1.
    "tmXBMajorLengthF": 0.020
    "tmXBMinorLengthF": 0.010
    "tmYLMajorLengthF": 0.020
    "tmYLMinorLengthF": 0.010
    "tmXBMinorPerMajor": 4
    "tmYLMinorPerMajor": 4
    "vpXF": .15
    "vpYF": .87
```

```
    "vpWidthF" :  .7
    "vpHeightF" : .32
    "tmYLLabelFontHeightF": 0.023
    "tmXBLabelFontHeightF": 0.023
end setvalues
getvalues plotid_function
    "xyCoordDataSpec": dataspec
end getvalues
setvalues dataspec(0)
    "xyLineColor": 2
end setvalues
draw(plotid_function)

;
; Draw a graph of the Y derivative.
;
setvalues plotid_function
    "trYMinF" :  -100.
    "xyCoordData": dataid3
    "tmYLTickStartF": -100.
    "vpXF": .15
    "vpYF": .42
    "vpHeightF" : .32
end setvalues
getvalues plotid_function
    "xyCoordDataSpec": dataspec
end getvalues
setvalues dataspec(0)
    "xyLineColor": 2
end setvalues

drw_text("Derivatives from ftcurvpd",0.5,0.95,0.04,xworkid)
drw_text("dx/du",0.65,0.82,0.033,xworkid)
drw_text("dy/du",0.34,0.37,0.033,xworkid)
draw(plotid_function)
frame(xworkid)

;
; End NCL script.
;
end
```

Plots for example 5 -- interpolated closed parametric curves and derivatives

---


Demo for KURVP1/KURVP2
40.0
30.0
20.0
10.0
0.0
0.0
10.0
20.0
30.0
40.0



Example 6 -- interpolated surface

```
      PROGRAM FTEX06
C
C  Example of SURF1/SURF2.
C
      PARAMETER (NXI=11,NYI=17,NXO=31,NYO=21,IDTEMP=2*NYI+NXI)
C
      DIMENSION X(NXI),Y(NYI),Z(NXI,NYI)
      DIMENSION ZX1(NYI),ZXM(NYI),ZY1(NXI),ZYN(NXI)
      DIMENSION ZP(NXI,NYI,3),TEMP(IDTEMP)
      DIMENSION XO(NXO),YO(NYO),ZO(NXO,NYO)
C
C Declare a function ZF(U,V) that defines a surface.
C
      ZF(U,V)=.5+.25*SIN(-7.*U)+.25*COS(5.*V)
C
C Define the surface to be drawn.
C
      DO 104 I=1,NXI
         X(I) = REAL(I-1)/REAL(NXI-1)
         DO 103 J=1,NYI
            Y(J) = REAL(J-1)/REAL(NYI-1)
```

```
            Z(I,J)=ZF(X(I),Y(J))
 103     CONTINUE
 104  CONTINUE
C
C  Do SURF1 setup.
C
      SIGMA = 1.
      ISF   = 255
      CALL SURF1(NXI,NYI,X,Y,Z,NXI,ZX1,ZXM,ZY1,ZYN,
     +           ZXY11,ZXYM1,ZXY1N,ZXYMN,ISF,ZP,TEMP,SIGMA,IERR)
      IF (IERR .NE. 0) THEN
        PRINT *, 'Error return from SURF =',IERR
        STOP
      ENDIF
C
C  Get interpolated points using SURF2.
C
      TINCX = 1.0/(NXO-1)
      TINCY = 1.0/(NYO-1)
      DO 20 I=1,NXO
        XO(I) = (I-1)*TINCX
        DO 10 J=1,NYO
          YO(J) = (J-1)*TINCY
          ZO(I,J) = SURF2(XO(I),YO(J),NXI,NYI,X,Y,Z,NXI,ZP,SIGMA)
   10   CONTINUE
   20 CONTINUE
C
C  Draw plot.
C
      CALL DRWFT6(NXO,NYO,XO,YO,ZO)
C
      STOP
      END
      SUBROUTINE DRWFT6(NXO,NYO,XO,YO,ZO)
C
C  Define error file, Fortran unit number, workstation type,
C  and workstation ID.
C
      PARAMETER (IERRF=6, LUNIT=2, IWTYPE=1, IWKID=1)
C
C  Open GKS, open and activate a workstation.
C
      CALL GOPKS (IERRF, ISZDM)
      CALL GOPWK (IWKID, LUNIT, IWTYPE)
      CALL GACWK (IWKID)
C
      CALL DRWTD3(NXO,NYO,XO,YO,ZO,3.85,2.75,1.65,6)
C
      CALL GDAWK (IWKID)
      CALL GCLWK (IWKID)
      CALL GCLKS
C
      RETURN
      END
      SUBROUTINE DRWTD3(NX,NY,X,Y,Z,S1,S2,S3,IST)
C
C  Procedure DRWTD3 uses the NCAR Graphics functions in Tdpack
C  to draw a surface plot of the data values in Z.
C
C  The point of observation is calculated from the 3D coordinate
C  (S1, S2, S3); the point looked at is the center of the surface.
```

```
C
C   NX     -  Dimension of the X-axis variable X.
C   NY     -  Dimension of the Y-axis variable Y.
C   X      -  An array of X-axis values.
C   Y      -  An array of Y-axis values.
C   Z      -  An array dimensioned for NX x NY containing data
C             values for each (X,Y) coordinate.
C   S1     -  X value for the eye position.
C   S2     -  Y value for the eye position.
C   S3     -  Z value for the eye position.
C   IST    -  A style index defining the colors used to shade the
C             surface as per:
C
C                    1  -  wire frame.
C                    2  -  gray shades underneath; gray shades on top.
C                    3  -  gray shades underneath; red shades on top.
C                    4  -  gray shades underneath; green shades on top.
C                    5  -  gray shades underneath; blue shades on top.
C                    6  -  gray shades underneath; cyan shades on top.
C                    7  -  gray shades underneath; magenta shades on top.
C
C             If IST is positive, then a white background is used;
C             if IST is the negative of any of the above values, then
C             a black background is used.
C
      PARAMETER (IERRF=6, LUNIT=2, IWKID=1, ITYPE=1)
      PARAMETER (MTRI=40000)
      DIMENSION X(NX),Y(NY),Z(NX,NY)
      DIMENSION RTRI(10,MTRI),RTWK(MTRI,2),ITWK(MTRI)
      DATA ANG1,ANG2,RMUL / -35.,25.,2.9 /
      COMMON /DSTDDT/ RTRI,RTWK,ITWK
C
C  Set the desired values of the shading parameters.  Values of SHDE
C  near 0 give brighter colors, and values near 1 give pastel shades.
C  Values of SHDR near 0 give a narrow range of shades, and values near
C  1 give a wide range of shades.
C
      DATA SHDE,SHDR / .1 , 0.8 /
      DATA DTOR / .017453292519943 /
C
      CALL GQOPS(ISTATE)
      IF (ISTATE .EQ. 0) THEN
        CALL GOPKS (6, 0)
        IF (ITYPE .EQ. 1) THEN
          CALL NGSETC('ME','srf.ncgm')
        ELSE IF ( (ITYPE .GE. 20) .AND. (ITYPE .LE. 31) ) THEN
          CALL NGSETC('ME','srf.ps')
        ENDIF
        CALL GOPWK (IWKID, LUNIT, ITYPE)
        CALL GACWK (IWKID)
      ENDIF
C
      IF (IST .LT. 0) THEN
        CALL GSCR(IWKID,0,0.,0.,0.)
        CALL GSCR(IWKID,1,1.,1.,1.)
      ELSE
        CALL GSCR(IWKID,0,1.,1.,1.)
        CALL GSCR(IWKID,1,0.,0.,0.)
      ENDIF
C
C  Find mins and maxs.
```

```
C
      XMIN = X(1)
      XMAX = X(1)
      DO 120 I=2,NX
        XMIN = MIN(XMIN,X(I))
        XMAX = MAX(XMAX,X(I))
  120 CONTINUE
      YMIN = Y(1)
      YMAX = Y(1)
      DO 125 I=2,NY
        YMIN = MIN(YMIN,Y(I))
        YMAX = MAX(YMAX,Y(I))
  125 CONTINUE
      ZMIN = Z(1,1)
      ZMAX = Z(1,1)
      DO 130 I=1,NX
        DO 140 J=1,NY
          ZMIN = MIN(ZMIN,Z(I,J))
          ZMAX = MAX(ZMAX,Z(I,J))
  140   CONTINUE
  130 CONTINUE
      XRNG = XMAX-XMIN
      YRNG = YMAX-YMIN
      ZRNG = ZMAX-ZMIN
      XMID = 0.5*(XMIN+XMAX)
      YMID = 0.5*(YMIN+YMAX)
      ZMID = 0.5*(ZMIN+ZMAX)
C
      CALL GSCR (IWKID,2,1.,0.,0.)
      CALL GSCR (IWKID,3,0.,1.,0.)
      CALL GSCR (IWKID,4,0.,0.,1.)
      CALL GSCR (IWKID,5,0.,1.,1.)
      CALL GSCR (IWKID,6,1.,0.,1.)
      CALL GSCR (IWKID,7,1.,1.,0.)
      CALL GSCR (IWKID,8,0.7,0.7,0.7)
C
      DO 101 ICOL=11,42
        P=1.-     REAL(ICOL-11)/31.
        Q=1.-SHDR*REAL(ICOL-11)/31.
        CALL GSCR (1,ICOL    ,     P,     P,     P)  !  gray scale
        CALL GSCR (1,ICOL+ 32,     Q,     Q,     Q)  !  white
        CALL GSCR (1,ICOL+ 64,     Q,SHDE*Q,SHDE*Q)  !  red
        CALL GSCR (1,ICOL+ 96,SHDE*Q,     Q,SHDE*Q)  !  green
        CALL GSCR (1,ICOL+128,SHDE*Q,SHDE*Q,     Q)  !  blue
        CALL GSCR (1,ICOL+160,SHDE*Q,     Q,     Q)  !  cyan
        CALL GSCR (1,ICOL+192,     Q,SHDE*Q,     Q)  !  magenta
  101 CONTINUE
C
C Define TDPACK rendering styles 0 through 7.  The indices 0-7 can
C then be used as final arguments in calls to TDITRI, TDSTRI, and TDMTRI.
C
      XSL = 0.05*XRNG
      YSL = 0.05*YRNG
      ZSL = 0.00*ZRNG
      CALL TDSTRS (1,-1, 0, -1,  0, -1, 1, 0, XSL, YSL, ZSL)
      CALL TDSTRS (2,43,74, 43, 74, 1, 1, 0, XSL, YSL, ZSL)
      CALL TDSTRS (3,43,74, 75,106, 1, 1, 0, XSL, YSL, ZSL)
      CALL TDSTRS (4,43,74,107,138, 1, 1, 0, XSL, YSL, ZSL)
      CALL TDSTRS (5,43,74,139,170, 1, 1, 0, XSL, YSL, ZSL)
      CALL TDSTRS (6,43,74,171,202, 1, 1, 0, XSL, YSL, ZSL)
      CALL TDSTRS (7,43,74,203,234, 1, 1, 0, XSL, YSL, ZSL)
```

```
C
C Create the triangle list representing a surface.
C
      NTRI=0
      CALL TDSTRI (X,NX,Y,NY,Z,NX,RTRI,MTRI,NTRI,IST)
      IF (NTRI .EQ. MTRI) THEN
        PRINT * , 'Triangle list overflow in TDITRI'
        STOP
      END IF
C
C  Determine a default eye position if none is specified.
C
      IF (S1.EQ.0. .AND. S2.EQ.0. .AND. S3.EQ.0.) THEN
        R = RMUL*SQRT(XRNG*XRNG + YRNG*YRNG + ZRNG*ZRNG)
        XEYE = XMID+R*COS(DTOR*ANG1)*COS(DTOR*ANG2)
        YEYE = YMID+R*SIN(DTOR*ANG1)*COS(DTOR*ANG2)
        ZEYE = ZMID+R*SIN(DTOR*ANG2)
      ELSE
        XEYE = S1
        YEYE = S2
        ZEYE = S3
      ENDIF
C
C Initialize TDPACK.
C
      CALL TDINIT (XEYE, YEYE, ZEYE, XMID, YMID, ZMID,
     +                   XMID, YMID, ZMID+0.1*ZRNG, 0)
C
C Order the triangles.
C
      CALL TDOTRI (RTRI,MTRI,NTRI,RTWK,ITWK,1)
      IF (NTRI .EQ. MTRI) THEN
        PRINT * , 'TRIANGLE LIST OVERFLOW IN TDOTRI'
        STOP
      END IF
C
C  Draw the triangles.
C
      CALL TDDTRI (RTRI,MTRI,NTRI,ITWK)
      CALL FRAME
C
      IF (ISTATE .EQ. 0) THEN
        CALL GDAWK (IWKID)
        CALL GCLWK (IWKID)
        CALL GCLKS
      ENDIF
C
      RETURN
      END
```

Example 6 -- interpolated surface

```
#include <stdio.h>
#include <math.h>
```

```c
#include <ncarg/ncargC.h>
#include <ncarg/gks.h>
#include <ncarg/ngmath.h>

void c_drwft6(int, int, float *, float *, float *);
void c_drwtd3(int, int, float *, float *, float *,
              float, float, float, int);

/*
 *  Example of c_ftsurf.
 */

#define NXI   11
#define NYI   17
#define NXO   31
#define NYO   21

#define IWTYPE 1
#define WKID   1

main()
{
  float x[NXI],y[NYI],z[NXI][NYI],xo[NXO],yo[NYO],zo[NXO][NYO];
  float *surface;
  int   i,j, ier;

/*
 *  Define the input surface.
 */
  for (i = 0; i < NXI; i++) {
    x[i] = (float) i / (float) (NXI-1);
    for (j = 0; j < NYI; j++) {
      y[j] = (float) j / (float) (NYI-1);
      z[i][j] = 0.5 + 0.25*sin(-7.*x[i]) + 0.25*cos(5.*y[j]);
    }
  }

/*
 *  Set up the output arrays.
 */
  for (i = 0; i < NXO; i++) {
    xo[i] = (float) i / (float) (NXO-1) ;
    for (j = 0; j < NYO; j++) {
      yo[j] = (float) j / (float) (NYO-1);
    }
  }

/*
 *  Get the interpolated surface.
 */
  surface = c_ftsurf(NXI, NYI, x, y, &z[0][0], NXO, NYO, xo, yo, &ier);

/*
 *  Draw plot.
 */
  c_drwft6(NXO,NYO,xo,yo,surface);

}

void c_drwft6(int nx,int ny,float xo[],float yo[],float zo[])
{
```

```
/*
 *  Open GKS, open and activate a workstation.
 */
  gopen_gks("stdout",0);
  gopen_ws(WKID, NULL, IWTYPE);
  gactivate_ws(WKID);

  c_drwtd3(nx,ny,xo,yo,zo,3.85,2.75,1.65,6);

/*
 *  Deactivate and close workstation, close GKS.
 */
  gdeactivate_ws(WKID);
  gclose_ws(WKID);
  gclose_gks();
}


#include <stdio.h>
#include <math.h>
#include <ncarg/ncargC.h>

#define WKTP 8
#define WKID 1
#define MTRI 40000
#define DTOR .017453292519943

#ifndef MAX
#define MAX(A,B)        (((A) > (B)) ? (A) : (B))
#endif

#ifndef MIN
#define MIN(A,B)        (((A) < (B)) ? (A) : (B))
#endif

float rtri[MTRI][10], rtwk[2][MTRI];
int   itwk[MTRI], ntri;

void c_drwtd3(int nx, int ny, float *x, float *y, float *z,
              float s1, float s2, float s3, int style)
{
/*
 * Procedure c_drwtd3 uses the NCAR Graphics functions in Tdpack to
 * draw a surface plot of the data values in z.
 *
 * The point of observation is specified by (s1, s2, s3); the point
 * looked at is the center of the surface.  If s1 = s2 = s3 = 0.,
 * then the observation point is calculated automatically.
 *
 *  nx     -  Dimension of the X-axis variable x.
 *  ny     -  Dimension of the Y-axis variable y.
 *  x      -  An array of X-axis values.
 *  y      -  An array of Y-axis values.
 *  z      -  An array dimensioned for nx X ny containing data
 *            values for each (X,Y) coordinate.
 *  s1     -  x value for the eye position.
 *  s2     -  y value for the eye position.
 *  s3     -  z value for the eye position.
 *  style  -  A style index defining the colors used to shade the
 *            surface as per:
 *
```

```
 *               1  -  wire frame.
 *               2  -  gray shades underneath; gray shades on top.
 *               3  -  gray shades underneath; red shades on top.
 *               4  -  gray shades underneath; green shades on top.
 *               5  -  gray shades underneath; blue shades on top.
 *               6  -  gray shades underneath; cyan shades on top.
 *               7  -  gray shades underneath; magenta shades on top.
 *
 *            If style is positive, then a white background is used;
 *            if style is the negative of any of the above values, then
 *            a black background is used.
 */
   float *fz;
   int   i, j;
   float ang1=-35., ang2=25., rmul=2.9, shde=0.1, shdr=0.8;
   float xmin, xmax, ymin, ymax, zmin, zmax, xrng, yrng, zrng,
         xmid, ymid, zmid;
   float p, q, r, xsl, ysl, zsl, xeye, yeye, zeye;
   Gop_st op_st;
   Gcolr_rep colval;

   ginq_op_st(&op_st);
   if (op_st == GST_GKCL) {
      gopen_gks ("stdout",0);
      if (WKTP == 1) {
        c_ngsetc("me","srf.ncgm");
      }
      else if ( (WKTP >= 20) && (WKTP <= 31) ) {
        c_ngsetc("me","srf.ps");
      }
      gopen_ws (WKID, 0, WKTP);
      gactivate_ws (WKID);
   }

/*
 *  Set foreground and background colors.
 */
   if (style >= 0) {
      colval.rgb.red = 1.;
      colval.rgb.green = 1.;
      colval.rgb.blue = 1.;
      gset_colr_rep(WKID,0,&colval);
      colval.rgb.red = 0.;
      colval.rgb.green = 0.;
      colval.rgb.blue = 0.;
      gset_colr_rep(WKID,1,&colval);
   }
   else {
      colval.rgb.red = 0.;
      colval.rgb.green = 0.;
      colval.rgb.blue = 0.;
      gset_colr_rep(WKID,0,&colval);
      colval.rgb.red = 1.;
      colval.rgb.green = 1.;
      colval.rgb.blue = 1.;
      gset_colr_rep(WKID,1,&colval);
   }

/*
 *  Find mins and maxs.
 */
```

```
xmin = x[0];
xmax = x[0];
for (i = 1; i < nx; i++) {
  xmin = MIN(xmin,x[i]);
  xmax = MAX(xmax,x[i]);
}
ymin = y[0];
ymax = y[0];
for (j = 1; j < ny; j++) {
  ymin = MIN(ymin,y[j]);
  ymax = MAX(ymax,y[j]);
}
zmin = z[0];
zmax = z[0];
for (i = 1; i < nx * ny; i++) {
  zmin = MIN(zmin, z[i]);
  zmax = MAX(zmax, z[i]);
}
xrng = xmax-xmin;
yrng = ymax-ymin;
zrng = zmax-zmin;
xmid = 0.5*(xmin+xmax);
ymid = 0.5*(ymin+ymax);
zmid = 0.5*(zmin+zmax);

colval.rgb.red = 1.;
colval.rgb.green = 0.;
colval.rgb.blue = 0.;
gset_colr_rep(WKID,2,&colval);
colval.rgb.red = 0.;
colval.rgb.green = 1.;
colval.rgb.blue = 0.;
gset_colr_rep(WKID,3,&colval);
colval.rgb.red = 0.;
colval.rgb.green = 0.;
colval.rgb.blue = 1.;
gset_colr_rep(WKID,4,&colval);
colval.rgb.red = 0.;
colval.rgb.green = 1.;
colval.rgb.blue = 1.;
gset_colr_rep(WKID,5,&colval);
colval.rgb.red = 1.;
colval.rgb.green = 0.;
colval.rgb.blue = 1.;
gset_colr_rep(WKID,6,&colval);
colval.rgb.red = 1.;
colval.rgb.green = 1.;
colval.rgb.blue = 0.;
gset_colr_rep(WKID,7,&colval);
colval.rgb.red = 0.7;
colval.rgb.green = 0.7;
colval.rgb.blue = 0.7;
gset_colr_rep(WKID,8,&colval);

for (i = 11; i < 43; i++) {
  p = 1. - (float) (i-11)/31.;
  q = 1. - shdr * (float) (i-11)/31.;
  colval.rgb.red = p;
  colval.rgb.green = p;
  colval.rgb.blue = p;
  gset_colr_rep(WKID,i,&colval);
```

```
      colval.rgb.red = q;
      colval.rgb.green = q;
      colval.rgb.blue = q;
      gset_colr_rep(WKID,i+32,&colval);
      colval.rgb.red = q;
      colval.rgb.green = shde*q;
      colval.rgb.blue = shde*q;
      gset_colr_rep(WKID,i+64,&colval);
      colval.rgb.red = shde*q;
      colval.rgb.green = q;
      colval.rgb.blue = shde*q;
      gset_colr_rep(WKID,i+96,&colval);
      colval.rgb.red = shde*q;
      colval.rgb.green = shde*q;
      colval.rgb.blue = q;
      gset_colr_rep(WKID,i+128,&colval);
      colval.rgb.red = shde*q;
      colval.rgb.green = q;
      colval.rgb.blue = q;
      gset_colr_rep(WKID,i+160,&colval);
      colval.rgb.red = q;
      colval.rgb.green = shde*q;
      colval.rgb.blue = q;
      gset_colr_rep(WKID,i+192,&colval);
    }

/*
 *  Rearrange the array, since Tdpack expects an array ordered as
 *  per Fortran.
 */
    fz = (float *) calloc(nx*ny,sizeof(float));

    for (i = 0; i < nx; i++) {
      for (j = 0; j < ny; j++) {
         fz[j*nx+i] = z[i*ny+j];
      }
    }

/*
 *  Define plotting styles.
 */
    xsl = 0.05*xrng;
    ysl = 0.05*yrng;
    zsl = 0.00*zrng;
    c_tdstrs(1, -1, 0, -1, 0, -1, 1, 0, xsl, ysl, zsl);
    c_tdstrs(1, -1, 0, -1, 0, -1, 1, 0, xsl, ysl, zsl);
    c_tdstrs(2, 43,74, 43, 74, 1, 1, 0, xsl, ysl, zsl);
    c_tdstrs(3, 43,74, 75,106, 1, 1, 0, xsl, ysl, zsl);
    c_tdstrs(4, 43,74,107,138, 1, 1, 0, xsl, ysl, zsl);
    c_tdstrs(5, 43,74,139,170, 1, 1, 0, xsl, ysl, zsl);
    c_tdstrs(6, 43,74,171,202, 1, 1, 0, xsl, ysl, zsl);
    c_tdstrs(7, 43,74,203,234, 1, 1, 0, xsl, ysl, zsl);

/*
 *  Create the triangle list representing a surface.
 */
    ntri = 0;
    c_tdstri(x,nx,y,ny,fz,nx,&rtri[0][0],MTRI,&ntri,style);
    if (ntri == MTRI) {
      printf("Triangle list overflow in c_tdstri\n");
      exit(1);
```

```
    }
    free(fz);

/*
 *  Determine a default eye position if none is specified.
 */
    if ( (s1==0.) && (s2==0.) && (s3==0.) ) {
      r = rmul * (float) sqrt( (float) (xrng*xrng + yrng*yrng + zrng*zrng));
      xeye = xmid + r * (float) cos( (double) (DTOR*ang1)) *
                 (float) cos( (double) (DTOR*ang2));
      yeye = ymid + r * (float) sin( (double) (DTOR*ang1)) *
                 (float) cos( (double) (DTOR*ang2));
      zeye = zmid + r * (float) sin( (double) (DTOR*ang2));
    }
    else {
      xeye = s1;
      yeye = s2;
      zeye = s3;
    }

/*
 *  Initialize Tdpack.
 */
    c_tdinit (xeye, yeye, zeye, xmid, ymid, zmid,
                xmid, ymid, zmid+0.1*zrng, 0);

/*
 *  Order the triangles.
 */
    c_tdotri(&rtri[0][0], MTRI, &ntri, &rtwk[0][0], itwk, 1);
    if (ntri == MTRI) {
      printf("Triangle list overflow in c_tdotri\n");
      exit(1);
    }

/*
 *  Draw the triangles.
 */
    c_tddtri(&rtri[0][0], MTRI, &ntri, itwk);
    c_frame();

    if (op_st == GST_GKCL) {
      gdeactivate_ws(WKID);
      gclose_ws(WKID);
      gclose_gks();
    }
}
```

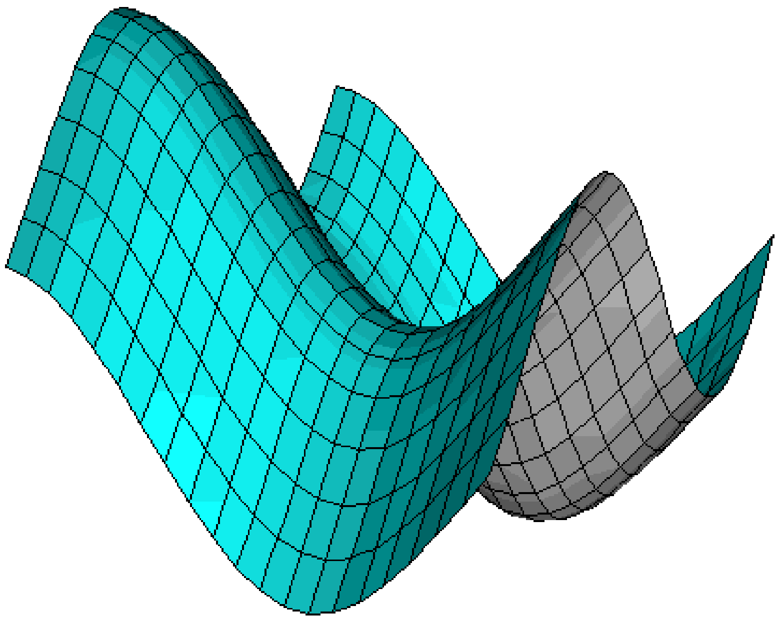
Example 6 -- interpolated surface

```
;
;   Description:  This program illustrates the use of ftsurf.
;
begin
```

```
;
;  Create the input arrays.
;
nxi = 11
nyi = 17
xi = fspan(0.,1.,nxi)
yi = fspan(0.,1.,nyi)
zi = new((/nxi,nyi/),float)
do i=0,nxi-1
  do j=0,nyi-1
    zi(i,j) = 0.5 + 0.25*sin(-7.*xi(i)) + 0.25*cos(5.*yi(j))
  end do
end do

;
;  Set up the output grid.
;
nxo = 31
nyo = 21
xo = fspan(0.,1.,nxo)
yo = fspan(0.,1.,nyo)

;
;  Interpolate.
;
zo = ftsurf(xi,yi,zi,xo,yo)

;
; Create workstation object.
;
  NCGM=1
  X11=0
  PS=0

  if (NCGM .eq. 1) then
;
; Create an ncgmWorkstation object.
;
    wid = create "dsgrid" ncgmWorkstationClass defaultapp
        "wkMetaName" : "nm06n.ncgm"
    end create
  else
    if (X11 .eq. 1) then
;
; Create an XWorkstation object.
;
      wid = create "dsgrid" xWorkstationClass defaultapp
        "wkPause" : "True"
      end create
    else
      if (PS .eq. 1) then
;
; Create a PSWorkstation object.
;
        wid = create "dsgrid" psWorkstationClass defaultapp
          "wkPSFileName" : "nm06n.ps"
        end create
      end if
     end if
  end if
```

```
  getvalues wid
    "wkGksWorkId" : gkswid
  end getvalues
;
;  Draw plot.
;
xeye = 3.85
yeye = 2.75
zeye = 1.65
drwtd2(gkswid, xo, yo, zo, xeye, yeye, zeye, -6)

end
```

Plot for example 6 -- interpolated surface



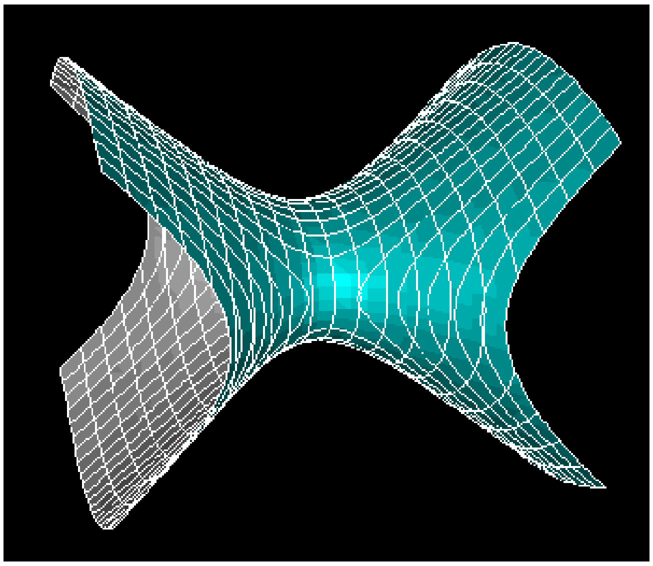
Fitgrid References

- Cline, Alan Kaylor, A Software Package for Curve and Surface Fitting Employing Splines Under

Tension, Pleasant Valley Software, 8603 Altus Cove, Austin TX 78759.

- Cline, A.K. and Renka, R.J., A Constrained Two-Dimensional Triangulation and the Solution of Closest Node Problems in the Presence of Barriers, SIAM Journal of Analysis, October, 1990, vol. 29, No. 5, pp. 1305-1321.

Csagrid, a cubic spline approximation package for random data in one, two, or three dimensions



Introduction (read this first)

Learning how to use the package

Csagrid Table of contents

Credits

Introduction

Learning how to use Csagrid

Reference charts and tables

Examples

Csagrid credits

Acknowledgments

Author: Fred Clare

Editor: Brian Bevirt.

Technical Reviewers: Tom Parker and Juli Rew

Production: Fred Clare and Brian Bevirt

Acknowledgments: The Csagrid package is based entirely on the package *SPLPACK* written by David Fulker. The user entry points have been changed and C and NCL interfaces added.

Also, thanks go to Dick Valent who told the author of the existence of *SPLPACK*.

Copyright

Trademark

Csagrid Introduction

About Csagrid

Csagrid (*C*ubic *S*pline *A*pproximation *GRIDder*) is a software package in the ngmath library that implements a cubic spline approximation algorithm to fit a function to input data. The input for the approximation is a set of randomly-spaced data. These data may be one-dimensional, two-dimensional, or three-dimensional. See the comparative analysis of the ngmath packages for details on which ngmath package is most appropriate for your problem.

Csagrid is the only package in the ngmath library that offers a fitted surface approximation for three-dimensional data. The results obtained from Csagrid will, in general, be more visually appealing than the results obtained from the weighted average algorithm used in Dsgrid. For an example of this, compare the plot for Csagrid example 5 to the plot for Dsgrid example 1.

The output is a set of values at coordinates on a user-specified grid, or at a set of user-specified points (including at a single point).

Functionally equivalent Fortran, C, and NCL interfaces are provided.

Csagrid is based on the work of David Fulker and his package *Splpack*.

For a proper understanding of the package, read the remainder of this module. If you are an impatient sort, you may go to the Overview of Csagrid procedures module and proceed at your own risk.

General background on interpolation and approximation methods

Computational interpolation and approximation methods can be divided into two basic classes: fitted function methods and weighted average methods.

The fitted function methods fit an algebraic surface to the known data and then pick the interpolated or approximated values from the fitted surface. The weighted average methods calculate interpolated or approximated values as weighted averages of known values.

The approximation method used in Csagrid

Csagrid uses a fitted-function algorithm. Given n-dimensional input data, Csagrid finds an n-dimensional cubic spline surface that is a least-squares fit to the input data. This surface will not necessarily pass through the original data.

The algorithm used in Csagrid to find an approximating cubic spline is different from cubic spline interpolation. In cubic spline interpolation, the knots occur at the input data values and the interpolated curve or surface passes through the input data. In Csagrid, the approximating curve or surface is calculated as a linear combination of spline basis functions, the number of which is equal to the user-specified value for the number of knots. The more basis functions there are, the closer the fit is to the original data. However, the more basis functions there are, the greater the cost in compute time. The plot for Example 1 illustrates the relationship between the number of basis functions and the appearance of the approximated surface. The timing table gives an indication of compute time requirements.

Options exist for weighting the input data values, for extrapolation into data sparse regions, and for taking first and second order partial derivatives in each coordinate direction.

Learning how to use Csagrid

To learn the details on using the Csagrid package, proceed to the module on learning how to use Csagrid.

Learning how to use Csagrid

The Csagrid package contains functionally equivalent interfaces for Fortran, C, and NCL.

Each interface is divided into entries for one-dimensional, two-dimensional, and three-dimensional data. For each of the three dimensionalities, there are simplified entries and expanded entries. The expanded entries are provided in the cases you want to find derivatives, weight the input data values, or extrapolate into data sparse regions. For the two-dimensional and three-dimensional entries, you also have the option of choosing output on a user-specified grid, or at a user-specified set of points.

Each of the entries in the Csagrid package has an argument for specifying the number of knots to use, hence it is necessary for you to understand that concept. See the section on knot selection in the module on the basic capabilities.

The entry points, as outlined above, are summarized in the Csagrid procedures module. To proceed, go to that summary table and click on your desired entry point for details.

Overview of Csagrid procedures

The table below summarizes all of the procedures in Csagrid for the three user interfaces:

- Fortran
- C
- NCL

Clicking on the procedure name in the tables below will link to a detailed description of that procedure.

Fortran

```
  One-dimensional input:
    CSA1S   - simple entry
    CSA1XS  - expanded entry

  Two-dimensional input:
    CSA2S   - simple entry, gridded output
    CSA2XS  - expanded entry, gridded output
    CSA2LS  - simple entry, list output
    CSA2LXS - expanded entry, list output

  Three-dimensional input:
    CSA3S   - simple entry, gridded output
    CSA3XS  - expanded entry, gridded output
    CSA3LS  - simple entry, list output.
    CSA3LXS - expanded entry, list output.
```

C

```
  One-dimensional input:
    c_csa1s   - simple entry
    c_csa1xs  - expanded entry

  Two-dimensional input:
    c_csa2s   - simple entry, gridded output
    c_csa2xs  - expanded entry, gridded output
    c_csa2ls  - simple entry, list output
    c_csa2lxs - expanded entry, list output

  Three-dimensional input:
    c_csa3s   - simple entry, gridded output
    c_csa3xs  - expanded entry, gridded output
    c_csa3ls  - simple entry, list output
    c_csa3lxs - expanded entry, list output
```

NCL

```
  One-dimensional input:
```

```
    csa1s   - simple entry
    csa1xs  - expanded entry

  Two-dimensional input:
    csa2s   - simple entry, gridded output
    csa2xs  - expanded entry, gridded output
    csa2ls  - simple entry, list output
    csa2lxs - expanded entry, list output

  Three-dimensional input:
    csa3s   - simple entry, gridded output
    csa3xs  - expanded entry, gridded output
    csa3ls  - simple entry, list output
    csa3lxs - expanded entry, list output
```

Detailed description of the procedures for csagrid

CSA1S - simple entry for one-dimensional input

CSA1S is called to find an approximating cubic spline for one-dimensional input data. If you want to weight the input data values, calculate derivatives, or handle data sparse areas specially, then you will need to use CSA1XS.

```
-------------------------------------------------------------------
            Argument | Type     |  Mode   | Dimension
-------------------------------------------------------------------
CALL CSA1S  (NI,     | Integer  | Input   |
            XI,      | Real     | Input   | NI
            YI,      | Real     | Input   | NI
            KNOTS,   | Integer  | Input   |
            NO,      | Integer  | Input   |
            XO,      | Real     | Input   | NO
            YO,      | Real     | Output  | NO
            NWRK,    | Integer  | Input   |
            WORK,    | Real     | Input   | NWRK = KNOTS * (KNOTS+3)
            IER)     | Integer  | Output  |
-------------------------------------------------------------------
```

NI
: The number of input data points. It must be that NI > 3 and, depending on the size of KNOTS below, NI may have to be larger.

XI
: An array containing the X coordinates of the input data points.

YI
: An array containing function values at the input XI values, that is, YI(L) is the value of the input function at XI(L) for L=1,NI.

KNOTS

The number of knots to be used in constructing the approximation spline. KNOTS must be at least 4. The larger the value for KNOTS, the closer the approximated curve will come to passing through the input function values.

NO
: The number of values to be calculated for the output curve.

XO
: An array containing the X coordinates of the output curve.

YO
: An array containing the calculated function values for the output curve.

NWRK
: The size of the WORK array. NWRK must be at least KNOTS*(KNOTS+3).

WORK
: A work array dimensioned for NWRK.

IER
: An error return value. If IER is returned as 0, then no errors were detected. If IER is non-zero, then refer to the error list in the error table for details.

CSA1XS - expanded entry for one-dimensional input

CSA1XS is called to find an approximating cubic spline for one-dimensional input data. CSA1XS is called if you want to weight the input data values, calculate derivatives, or handle data sparse areas specially. If you do not want to do any of these three things, then use CSA1S.

```
--------------------------------------------------------------------
             Argument | Type     |  Mode    | Dimension
--------------------------------------------------------------------
CALL CSA1XS  (NI,     | Integer  | Input    |
             XI,      | Real     | Input    | NI
             YI,      | Real     | Input    | NI
             WTS,     | Real     | Input    | NI
             KNOTS,   | Integer  | Input    |
             SMTH,    | Real     | Input    |
             NDERIV,  | Integer  | Input    |
             NO,      | Integer  | Input    |
             XO,      | Real     | Input    | NO
             YO,      | Real     | Output   | NO
             NWRK,    | Integer  | Input    |
             WORK,    | Real     | Input    | NWRK = KNOTS * (KNOTS+3)
             IER)     | Integer  | Output   |
--------------------------------------------------------------------
```

NI
: The number of input data points. It must be that NI > 3 and, depending on the size of KNOTS below, NI may have to be larger.

XI
: An array containing the X coordinates of the input data points.

YI
: An array containing function values at the input XI values, that is, YI(L) is the value of the input function at XI(L) for L=1,NI.

WTS
: An array containing weights for the YI values at the input XI values, that is, WTS(L) is a weight

for the value of YI(L) for L=1,NI. If you do not desire to weight the input YI values, then set WTS(1) to -1. The weights in the WTS array are relative and may be set to any non-negative value. When CSA1XS is called, the weights are summed and the individual weights are normalized so that the weight sum is unity.

KNOTS
: The number of knots to be used in constructing the approximation spline. KNOTS must be at least 4. The larger the value for KNOTS, the closer the approximated curve will come to passing through the input function values.

SMTH
: A parameter that controls extrapolation into data sparse regions. If SMTH is zero, then nothing special is done in data sparse regions. A good first choice for SMTH is 1.

NDERIV
: Specifies whether you want functional values (NDERIV=0), first derivative values (NDERIV=1), or second derivative values (NDERIV=2).

NO
: The number of values to be calculated in the output curve.

XO
: An array containing the X coordinates for the output curve.

YO
: An array containing the calculated function values of the output curve.

NWRK
: The size of the WORK array. NWRK must be at least KNOTS*(KNOTS+3).

WORK
: A work array dimensioned for NWRK.

IER
: An error return value. If IER is returned as 0, then no errors were detected. If IER is non-zero, then refer to the error list in the error table for details.

CSA2S - simple entry for two-dimensional input, gridded output

CSA2S is called to find an approximating cubic spline surface for two-dimensional input data. If you want to weight the input data values, calculate derivatives, or handle data sparse areas specially, then you will need to use CSA2XS.

```
-------------------------------------------------------------------
            Argument | Type     |  Mode   | Dimension
-------------------------------------------------------------------
CALL CSA2S  (NI,     | Integer  | Input   |
            XI,      | Real     | Input   | 2 x NI
            UI,      | Real     | Input   | NI
            KNOTS,   | Integer  | Input   | 2
            NXO,     | Integer  | Input   |
            NYO,     | Integer  | Input   |
            XO,      | Real     | Input   | NXO
            YO,      | Real     | Input   | NYO
            UO,      | Real     | Output  | NXO x NYO
            NWRK,    | Integer  | Input   |
            WORK,    | Real     | Input   | NWRK = NK * (NK+3) where
                     |          |         | NK = KNOTS(1) * KNOTS(2)
            IER)     | Integer  | Output  |
-------------------------------------------------------------------
```

NI
: The number of input data points. It must be that NI > 3 and, depending on the size of KNOTS below, NI may have to be larger.

XI
: An array containing the X - Y coordinates of the input data points. XI(1,L) is the X coordinate and XI(2,L) is the Y coordinate for the input domain for L=1,NI.

UI
: An array containing the function values at the input XI values, that is UI(L) is the value of the input function at the coordinate (XI(1,L),XI(2,L)) for L=1,NI.

KNOTS
: The number of knots to be used in constructing the approximation spline. KNOTS(1) specifies the number of knots in the X direction and KNOTS(2) specifies the number of knots in the Y direction. Both KNOTS(1) and KNOTS(2) must be at least 4. The larger the values for KNOTS, the closer the approximated curve will come to passing through the input function values.

NXO
: The number of X coordinate values in the output grid.

NYO
: The number of Y coordinate values in the output grid.

XO
: An array containing the X coordinates of the output surface.

YO
: An array containing the Y coordinates of the output surface.

UO
: An array containing the calculated function values for the output surface. UO(I,J) is the calculated functional value at (XO(I),YO(J)) for I=1,NXO and J=1,NYO.

NWRK
: The size of the WORK array. NWRK must be at least KNOTS(1)*KNOTS(2)*(KNOTS(1)*KNOTS(2)+3).

WORK
: A work array dimensioned for NWRK.

IER
: An error return value. If IER is returned as 0, then no errors were detected. If IER is non-zero, then refer to the error list in the error table for details.

CSA2XS - expanded entry for two-dimensional input, gridded output

CSA2XS is called if you want to weight the input data values, calculate derivatives, or handle data sparse areas specially. If you do not want to do any of these three things, then use CSA2S.

```
------------------------------------------------------------------
             Argument | Type     |  Mode   | Dimension
------------------------------------------------------------------
CALL CSA2XS (NI,      | Integer  | Input   |
             XI,      | Real     | Input   | 2 x NI
             UI,      | Real     | Input   | NI
             WTS,     | Real     | Input   | NI
             KNOTS,   | Integer  | Input   | 2
             SMTH,    | Real     | Input   |
```

```
            NDERIV,    | Integer | Input   | 2
            NXO,       | Integer | Input   |
            NYO,       | Integer | Input   |
            XO,        | Real    | Input   | NXO
            YO,        | Real    | Input   | NYO
            UO,        | Real    | Output  | NXO x NYO
            NWRK,      | Integer | Input   |
            WORK,      | Real    | Input   | NWRK = NK * (NK+3) where
                       |         |         | NK = KNOTS(1) * KNOTS(2)
            IER)       | Integer | Output  |
---------------------------------------------------------------------
```

NI
: The number of input data points. It must be that NI > 3 and, depending on the size of KNOTS below, NI may have to be larger.

XI
: An array containing the X - Y coordinates of the input data points. XI(1,L) is the X coordinate and XI(2,L) is the Y coordinate for the input domain for L=1,NI.

UI
: An array containing the function values at the input XI values, that is UI(L) is the value of the input function at the coordinate (XI(1,L),XI(2,L)) for L=1,NI.

WTS
: An array containing weights for the UI values at the input XI values, that is, WTS(L) is a weight for the value of UI(L) for L=1,NI. If you do not desire to weight the input UI values, then set WTS(1) to -1. The weights in the WTS array are relative and may be set to any non-negative value. When CSA2XS is called, the weights are summed and the individual weights are normalized so that the weight sum is unity.

KNOTS
: The number of knots to be used in constructing the approximation spline. KNOTS(1) specifies the number of knots in the X direction and KNOTS(2) specifies the number of knots in the Y direction. Both KNOTS(1) and KNOTS(2) must be at least 4. The larger the value for KNOTS, the closer the approximated curve will come to passing through the input function values.

SMTH
: A parameter that controls extrapolation into data sparse regions. If SMTH is zero, then nothing special is done in data sparse regions. A good first choice for SMTH is 1.

NDERIV
: Specifies which partial derivatives are desired. NDERIV(1) indicates if the 0 th, 1 st, or 2 nd partial in the X direction is desired; NDERIV(2) indicates if the 0 th, 1 st, or 2 nd partial in the Y direction is desired.

NXO
: The number of X coordinate values in the output grid.

NYO
: The number of Y coordinate values in the output grid.

XO
: An array containing the X coordinates of the output surface.

YO
: An array containing the Y coordinates of the output surface.

UO
: An array containing the calculated function values for the output surface. UO(I,J) is the calculated functional value at (XO(I),YO(J)) for I=1,NXO and J=1,NYO.

NWRK

The size of the WORK array. NWRK must be at least KNOTS(1)*KNOTS(2)*(KNOTS(1)*KNOTS(2)+3).

WORK

A work array dimensioned for NWRK.

IER

An error return value. If IER is returned as 0, then no errors were detected. If IER is non-zero, then refer to the error list in the error table for details.

CSA2LS - simple entry for two-dimensional input, list output

CSA2LS is called to find an approximating cubic spline surface for two-dimensional input data. If you want to weight the input data values, calculate derivatives, or handle data sparse areas specially, then you will need to use CSA2LXS.

```
------------------------------------------------------------------
             Argument | Type      |  Mode   | Dimension
------------------------------------------------------------------
CALL CSA2LS  (NI,     | Integer   | Input   |
             XI,      | Real      | Input   | 2 x NI
             UI,      | Real      | Input   | NI
             KNOTS,   | Integer   | Input   | 2
             NO,      | Integer   | Input   |
             XO,      | Real      | Input   | NO
             YO,      | Real      | Input   | NO
             UO,      | Real      | Output  | NO
             NWRK,    | Integer   | Input   |
             WORK,    | Real      | Input   | NWRK = NK * (NK+3) where
                      |           |         | NK = KNOTS(1) * KNOTS(2)
             IER)     | Integer   | Output  |
------------------------------------------------------------------
```

NI

The number of input data points. It must be that NI > 3 and, depending on the size of KNOTS below, NI may have to be larger.

XI

An array containing the X - Y coordinates of the input data points. XI(1,L) is the X coordinate and XI(2,L) is the Y coordinate for the input domain for L=1,NI.

UI

An array containing the function values at the input XI values, that is UI(L) is the value of the input function at the coordinate (XI(1,L),XI(2,L)) for L=1,NI.

KNOTS

The number of knots to be used in constructing the approximation spline. KNOTS(1) specifies the number of knots in the X direction and KNOTS(2) specifies the number of knots in the Y direction. Both KNOTS(1) and KNOTS(2) must be at least 4. The larger the value for KNOTS, the closer the approximated curve will come to passing through the input function values.

NO

The number of coordinate values in the output list. NO can be any positive number.

XO

An array containing the X coordinates of the output list.

YO

An array containing the Y coordinates of the output list.

UO

An array containing the calculated function values. UO(L) is the calculated functional value at (XO(L), YO(L)) for L=1,NO.

NWRK

The size of the WORK array. NWRK must be at least KNOTS(1)*KNOTS(2)*(KNOTS(1)*KNOTS(2)+3).

WORK

A work array dimensioned for NWRK.

IER

An error return value. If IER is returned as 0, then no errors were detected. If IER is non-zero, then refer to the error list in the error table for details.

CSA2LXS - expanded entry for two-dimensional input, list output

CSA2LXS is called if you want to weight the input data values, calculate derivatives, or handle data sparse areas specially. If you do not want to do any of these three things, then use CSA2LS.

```
---------------------------------------------------------------------
              Argument | Type      |  Mode    | Dimension
---------------------------------------------------------------------
CALL CSA2LXS  (NI,     | Integer   | Input    |
              XI,      | Real      | Input    | 2 x NI
              UI,      | Real      | Input    | NI
              WTS,     | Real      | Input    | NI
              KNOTS,   | Integer   | Input    | 2
              SMTH,    | Real      | Input    |
              NDERIV,  | Integer   | Input    | 2
              NO,      | Integer   | Input    |
              XO,      | Real      | Input    | NO
              YO,      | Real      | Input    | NO
              UO,      | Real      | Output   | NO
              NWRK,    | Integer   | Input    |
              WORK,    | Real      | Input    | NWRK = NK * (NK+3) where
                       |           |          | NK = KNOTS(1) * KNOTS(2)
              IER)     | Integer   | Output   |
---------------------------------------------------------------------
```

NI

The number of input data points. It must be that NI > 3 and, depending on the size of KNOTS below, NI may have to be larger.

XI

An array containing the X - Y coordinates of the input data points. XI(1,L) is the X coordinate and XI(2,L) is the Y coordinate for the input domain for L=1,NI.

UI

An array containing the function values at the input XI values, that is UI(L) is the value of the input function at the coordinate (XI(1,L),XI(2,L)) for L=1,NI.

WTS

An array containing weights for the UI values at the input XI values, that is, WTS(L) is a weight for the value of UI(L) for L=1,NI. If you do not desire to weight the input UI values, then set WTS(1) to -1. The weights in the WTS array are relative and may be set to any non-negative value. When CSA2LXS is called, the weights are summed and the individual weights are normalized so that the weight sum is unity.

KNOTS
: The number of knots to be used in constructing the approximation spline. KNOTS(1) specifies the number of knots in the X direction and KNOTS(2) specifies the number of knots in the Y direction. Both KNOTS(1) and KNOTS(2) must be at least 4. The larger the value for KNOTS, the closer the approximated curve will come to passing through the input function values.

SMTH
: A parameter that controls extrapolation into data sparse regions. If SMTH is zero, then nothing special is done in data sparse regions. A good first choice for SMTH is 1.

NDERIV
: Specifies which partial derivatives are desired. NDERIV(1) indicates if the 0 th, 1 st, or 2 nd partial in the X direction is desired; NDERIV(2) indicates if the 0 th, 1 st, or 2 nd partial in the Y direction is desired.

NO
: The number of coordinate values in the output list. NO can be any positive number.

XO
: An array containing the X coordinates of the output list.

YO
: An array containing the Y coordinates of the output list.

UO
: An array containing the calculated function values for the output surface. UO(L) is the calculated functional value at (XO(L), YO(L)) for L=1,NO.

NWRK
: The size of the WORK array. NWRK must be at least KNOTS(1)*KNOTS(2)*(KNOTS(1)*KNOTS(2)+3).

WORK
: A work array dimensioned for NWRK.

IER
: An error return value. If IER is returned as 0, then no errors were detected. If IER is non-zero, then refer to the error list in the error table for details.

CSA3S - simple entry for three-dimensional input, gridded output

CSA3S is called to find an approximating cubic spline for three-dimensional input data. If you want to weight the input data values, calculate derivatives, or handle data sparse areas specially, then you will need to use CSA3XS.

```
-------------------------------------------------------------------------------
             Argument | Type     |  Mode   | Dimension
-------------------------------------------------------------------------------
CALL CSA3S  (NI,      | Integer  | Input   |
             XI,      | Real     | Input   | 3 x NI
             UI,      | Real     | Input   | NI
             KNOTS,   | Integer  | Input   | 3
             NXO,     | Integer  | Input   |
             NYO,     | Integer  | Input   |
             NZO,     | Integer  | Input   |
```

XO,	Real	Input	NXO
YO,	Real	Input	NYO
ZO,	Real	Input	NZO
UO,	Real	Output	NXO x NYO x NZO
NWRK,	Integer	Input	
WORK,	Real	Input	NWRK = NK * (NK+3) where NK = KNOTS(1) * KNOTS(2) * KNOTS(3)
IER)	Integer	Output	

NI
: The number of input data points. It must be that NI > 3 and, depending on the size of KNOTS below, NI may have to be larger.

XI
: An array containing the X - Y - Z coordinates of the input data points. XI(1,L) is the X coordinate, XI(2,L) is the Y coordinate, and XI(2,L) is the Z coordinate for the input domain for L=1,NI.

UI
: An array containing the function values at the input XI values, that is, UI(L) is the value of the input function at the coordinate (XI(1,L), XI(2,L), XI(3,L)) for L=1,NI.

KNOTS
: The number of knots to be used in constructing the approximation spline. KNOTS(1) specifies the number of knots in the X direction, KNOTS(2) specifies the number of knots in the Y direction and KNOTS(3) specifies the number of knots in the Z direction. KNOTS(I) must be at least 4 for I=1,3. The larger the value for KNOTS, the closer the approximated curve will come to passing through the input function values.

NXO
: The number of X coordinate values in the output grid.

NYO
: The number of Y coordinate values in the output grid.

NZO
: The number of Z coordinate values in the output grid.

XO
: An array containing the X coordinates of the output grid.

YO
: An array containing the Y coordinates of the output grid.

ZO
: An array containing the Z coordinates of the output grid.

UO
: An array containing the calculated function values for the output function. UO(I,J,K) is the calculated functional value at (XO(I), YO(J), ZO(K)) for I=1,NXO and J=1,NYO and K=1,NZO.

NWRK
: The size of the WORK array. NWRK must be at least NK*(NK+3) where NK=KNOTS(1)*KNOTS(2)*KNOTS(3).

WORK
: A work array dimensioned for NWRK.

IER
: An error return value. If IER is returned as 0, then no errors were detected. If IER is non-zero, then refer to the error list in the error table for details.

CSA3XS - expanded entry for three-dimensional input, gridded

output

CSA3XS is called if you want to weight the input data values, calculate derivatives, or handle data sparse areas specially. If you do not want to do any of these three things, then use CSA3S.

```
------------------------------------------------------------------------------
            Argument    | Type       |  Mode    | Dimension
------------------------------------------------------------------------------
CALL CSA3XS  (NI,       | Integer    | Input    |
              XI,       | Real       | Input    | 3 x NI
              UI,       | Real       | Input    | NI
              WTS,      | Real       | Input    | NI
              KNOTS,    | Integer    | Input    | 3
              SMTH,     | Real       | Input    |
              NDERIV,   | Integer    | Input    | 3
              NXO,      | Integer    | Input    |
              NYO,      | Integer    | Input    |
              NZO,      | Integer    | Input    |
              XO,       | Real       | Input    | NXO
              YO,       | Real       | Input    | NYO
              ZO,       | Real       | Input    | NZO
              UO,       | Real       | Output   | NXO x NYO x NZO
              NWRK,     | Integer    | Input    |
              WORK,     | Real       | Input    | NWRK = NK * (NK+3) where
                        |            |          | NK = KNOTS(1) * KNOTS(2) * KNOTS(3)
              IER)      | Integer    | Output   |
------------------------------------------------------------------------------
```

NI
: The number of input data points. It must be that NI > 3 and, depending on the size of KNOTS below, NI may have to be larger.

XI
: An array containing the X - Y - Z coordinates of the input data points. XI(1,L) is the X coordinate, XI(2,L) is the Y coordinate, and XI(3,L) is the Z coordinate for the input domain for L=1,NI.

UI
: An array containing the function values at the input XI values, that is UI(L) is the value of the input function at the coordinate (XI(1,L), XI(2,L), XI(3,L)) for L=1,NI.

WTS
: An array containing weights for the UI values at the input XI values, that is, WTS(L) is a weight for the value of UI(L) for L=1,NI. If you do not desire to weight the input UI values, then set WTS(1) to -1. The weights in the WTS array are relative and may be set to any non-negative value. When CSA3XS is called, the weights are summed and the individual weights are normalized so that the weight sum is unity.

KNOTS
: The number of knots to be used in constructing the approximation spline. KNOTS(1) specifies the number of knots in the X direction, KNOTS(2) specifies the number of knots in the Y direction and KNOTS(3) specifies the number of knots in the Z direction. KNOTS(I) must be at least 4 for I=1,3. The larger the values for KNOTS, the closer the approximated curve will come to passing through the input function values.

SMTH
: A parameter that controls extrapolation into data sparse regions. If SMTH is zero, then nothing special is done in data sparse regions. A good first choice for SMTH is 1.

NDERIV

Specifies which partial derivatives are desired. NDERIV(1) indicates whether the 0 th, 1 st, or 2 nd partial in the X direction is desired; NDERIV(2) indicates whether the 0 th, 1 st, or 2 nd partial in the Y direction is desired; NDERIV(3) indicates whether the 0 th, 1 st, or 2 nd partial in the Z direction is desired.

NXO
: The number of X coordinate values in the output grid.

NYO
: The number of Y coordinate values in the output grid.

NZO
: The number of Z coordinate values in the output grid.

XO
: An array containing the X coordinates of the output grid.

YO
: An array containing the Y coordinates of the output grid.

ZO
: An array containing the Z coordinates of the output grid.

UO
: An array containing the calculated function values for the output grid. UO(I,J,K) is the calculated functional value at (XO(I), YO(J), ZO(K)) for I=1,NXO and J=1,NYO and K=1,NZO.

NWRK
: The size of the WORK array. NWRK must be at least NK*(NK+3) where NK=KNOTS(1)*KNOTS(2)*KNOTS(3)

WORK
: A work array dimensioned for NWRK.

IER
: An error return value. If IER is returned as 0, then no errors were detected. If IER is non-zero, then refer to the error list in the error table for details.

CSA3LS - simple entry for three-dimensional input, list output

CSA3LS is called to find an approximating cubic spline for three-dimensional input data. If you want to weight the input data values, calculate derivatives, or handle data sparse areas specially, then you will need to use CSA3LXS.

```
-------------------------------------------------------------------------------
             Argument | Type      |  Mode   | Dimension
-------------------------------------------------------------------------------
CALL CSA3LS  (NI,     | Integer   | Input   |
             XI,      | Real      | Input   | 3 x NI
             UI,      | Real      | Input   | NI
             KNOTS,   | Integer   | Input   | 3
             NO,      | Integer   | Input   |
             XO,      | Real      | Input   | NXO
             YO,      | Real      | Input   | NYO
             ZO,      | Real      | Input   | NZO
             UO,      | Real      | Output  | NXO x NYO x NZO
             NWRK,    | Integer   | Input   |
             WORK,    | Real      | Input   | NWRK = NK * (NK+3) where
                      |           |         | NK = KNOTS(1) * KNOTS(2) * KNOTS(3)
             IER)     | Integer   | Output  |
-------------------------------------------------------------------------------
```

NI
: The number of input data points. It must be that NI > 3 and, depending on the size of KNOTS below, NI may have to be larger.

XI
: An array containing the X - Y - Z coordinates of the input data points. XI(1,L) is the X coordinate, XI(2,L) is the Y coordinate and XI(3,L) s the Z coordinate for the input domain for L=1,NI.

UI
: An array containing the function values at the input XI values, that is UI(L) is the value of the input function at the coordinate (XI(1,L), XI(2,L), XI(3,L)) for L=1,NI.

KNOTS
: The number of knots to be used in constructing the approximation spline. KNOTS(1) specifies the number of knots in the X direction, KNOTS(2) specifies the number of knots in the Y direction and KNOTS(3) specifies the number of knots in the Z direction. KNOTS must be at least 4. The larger the value for KNOTS, the closer the approximated curve will come to passing through the input function values.

NO
: The number of coordinate values in the output list. NO can be any positive number.

XO
: An array containing the X coordinates of the output list.

YO
: An array containing the Y coordinates of the output list.

ZO
: An array containing the Y coordinates of the output list.

UO
: An array containing the calculated function values for the output function. UO(L) is the calculated functional value at (XO(L), YO(L), ZO(L)) for L=1,NO.

NWRK
: The size of the WORK array. NWRK must be at least NK*(NK+3) where NK=KNOTS(1)*KNOTS(2)*KNOTS(3).

WORK
: A work array dimensioned for NWRK.

IER
: An error return value. If IER is returned as 0, then no errors were detected. If IER is non-zero, then refer to the error list in the error table for details.

CSA3LXS - expanded entry for three-dimensional input, list output

CSA3LXS is called if you want to weight the input data values, calculate derivatives, or handle data sparse areas specially. If you do not want to do any of these three things, then use CSA3LS.

```
--------------------------------------------------------------------------------
              Argument | Type     |  Mode   | Dimension
--------------------------------------------------------------------------------
CALL CSA3LXS (NI,      | Integer  | Input   |
              XI,      | Real     | Input   | 3 x NI
              UI,      | Real     | Input   | NI
              WTS,     | Real     | Input   | NI
              KNOTS,   | Integer  | Input   | 3
```

SMTH,	Real	Input	
NDERIV,	Integer	Input	3
NO,	Integer	Input	
XO,	Real	Input	NO
YO,	Real	Input	NO
ZO,	Real	Input	NO
UO,	Real	Output	NO
NWRK,	Integer	Input	
WORK,	Real	Input	NWRK = NK * (NK+3) where NK = KNOTS(1) * KNOTS(2) * KNOTS(3)
IER)	Integer	Output	

NI
: The number of input data points. It must be that NI > 3 and, depending on the size of KNOTS below, NI may have to be larger.

XI
: An array containing the X - Y - Z coordinates of the input data points. XI(1,L) is the X coordinate, XI(2,L) is the Y coordinate, and XI(3,L) is the Z coordinate for the input domain for L=1,NI.

UI
: An array containing the function values at the input XI values, that is UI(L) is the value of the input function at the coordinate (XI(1,L), XI(2,L), XI(3,L)) for L=1,NI.

WTS
: An array containing weights for the UI values at the input XI values, that is, WTS(L) is a weight for the value of UI(L) for L=1,NI. If you do not desire to weight the input UI values, then set WTS(1) to -1. The weights in the WTS array are relative and may be set to any non-negative value. When CSA3LXS is called, the weights are summed and the individual weights are normalized so that the weight sum is unity.

KNOTS
: The number of knots to be used in constructing the approximation spline. KNOTS(1) specifies the number of knots in the X direction, KNOTS(2) specifies the number of knots in the Y direction and KNOTS(3) specifies the number of knots in the Z direction. KNOTS(I) must be at least 4 for I=1,3. The larger the value for KNOTS, the closer the approximated curve will come to passing through the input function values.

SMTH
: A parameter that controls extrapolation into data sparse regions. If SMTH is zero, then nothing special is done in data sparse regions. A good first choice for SMTH is 1.

NDERIV
: Specifies which partial derivatives are desired. NDERIV(1) indicates whether the 0 th, 1 st, or 2 nd partial in the X direction is desired; NDERIV(2) indicates whether the 0 th, 1 st, or 2 nd partial in the Y direction is desired; NDERIV(3) indicates whether the 0 th, 1 st, or 2 nd partial in the Z direction is desired.

NO
: The number of coordinate values in the output list. NO can be any positive number.

XO
: An array containing the X coordinates of the output list.

YO
: An array containing the Y coordinates of the output list.

ZO
: An array containing the Z coordinates of the output list.

UO

An array containing the calculated function values for the output surface. UO(L) is the calculated functional value at (XO(L), YO(L), ZO(L)) for L=1,NO.

NWRK

The size of the WORK array. NWRK must be at least NK*(NK+3) where NK=KNOTS(1)*KNOTS(2)*KNOTS(3).

WORK

A work array dimensioned for NWRK.

IER

An error return value. If IER is returned as 0, then no errors were detected. If IER is non-zero, then refer to the error list in the error table for details.

c_csa1s - simple entry for one-dimensional input

---

c_csa1s is called to find an approximating cubic spline for one-dimensional input data. If you want to weight the input data values, calculate derivatives, or handle data sparse areas specially, then you will need to use c_csa1xs.

Function prototype:

```
float *c_csa1s(int, float [], float [], int, int, float [], int *);
```

Return value:

c_csa1s returns a pointer to a linear array of data that is the approximated curve. That is, if *out* is declared as

```
float *out;
```

and we set:

```
out = c_csa1s(n, x, y, z, knots, no, xo, &ier);
```

then *out*[i] is the approximated function value at coordinate point *xo*[i] for 0 <= i < *no*. The space for *out* is allocated internal to c_csa1s and is *no* floats in size.

Argument description:

```
-------------------------------------------------
                Argument | Type      |  Size
-------------------------------------------------
float *c_csa1s (n,       | int       |
                xi,      | float []  | n
                yi,      | float []  | n
                knots,   | int       |
                m,       | int       |
                xo,      | float []  | m
                ier      | int *     |
               );
-------------------------------------------------
```

n

The number of input data points. It must be that n > 3 and, depending on the size of knots below, n may have to be larger.

xi
: An array containing the abscissae for the input function.

yi
: An array containing the functional values of the input function -- yi[k] is the functional value at xi[k] for k=0,n-1.

knots
: The number of knots to be used in constructing the approximation spline. knots must be at least 4. The larger the value for knots, the closer the approximated curve will come to passing through the input function values.

m
: The number of values to be calculated for the output curve.

xo
: An array containing the abscissae for the approximation output values.

ier
: An error return value. If *ier is returned as 0, then no errors were detected. If *ier is non-zero, then refer to the error list in the error table for details.

c_csa1xs - expanded entry for one-dimensional input

c_csa1xs is called to find an approximating cubic spline for one-dimensional input data. c_csa1xs is called if you want to weight the input data values, calculate derivatives, or handle data sparse areas specially. If you do not want to do any of these three things, then use c_csa1s.

Function prototype:

```
float *c_csa1xs(int, float [], float [], float [], int,
                float, int, int, float [], int *);
```

Return value:

c_csa1xs returns a pointer to a linear array of data that is the approximated curve. That is, if *out* is declared as

```
float *out;
```

and we set:

```
out = c_csa1s(n, x, y, z, knots, smth, nderiv, no, xo, &ier);
```

then *out*[i] is the approximated function value at coordinate point *xo*[i] for 0 <= i < *no*. The space for *out* is allocated internal to c_csa1xs and is *m* floats in size.

```
--------------------------------------------
                 Argument | Type     | Size
--------------------------------------------
float *c_csa1xs (n,       | int      |
                 xi,      | float [] | n
                 yi,      | float [] | n
                 wts,     | float [] | n
```

```
                  knots,     | int         |
                  smth,      | float       |
                  nderiv,    | int         |
                  m,         | int         |
                  xo,        | float []    | m
                  ier)       | int *       |
---------------------------------------------
```

n
: The number of input data points. It must be that n > 3 and, depending on the size of *knots* below, n may have to be larger.

xi
: An array containing the X coordinates of the input data points.

yi
: An array containing function values at the input xi values, that is, yi[l] is the value of the input function at xi[l] for l=0,n-1.

wts
: An array containing weights for the yi values at the input xi values, that is, wts[l] is a weight for the value of yi[l] for l=0,n-1. If you do not desire to weight the input yi values, then set wts[0] to -1. The weights in the wts array are relative and may be set to any non-negative value. When c_csa1xs is called, the weights are summed and the individual weights are normalized so that the weight sum is unity.

knots
: The number of knots to be used in constructing the approximation spline. knots must be at least 4. The larger the value for knots, the closer the approximated curve will come to passing through the input function values.

smth
: A parameter that controls extrapolation into data sparse regions. If smth is zero, then nothing special is done in data sparse regions. A good first choice for smth is 1.

nderiv
: Specifies whether you want functional values (nderiv=0), first derivative values (nderiv=1), or second derivative values (nderiv=2).

m
: The number of values to be calculated in the output curve.

xo
: An array containing the X coordinates for the output curve.

ier
: An error return value. If *ier is returned as 0, then no errors were detected. If *ier is non-zero, then refer to the error list in the error table for details.

c_csa2s - simple entry for two-dimensional input, gridded output

---

c_csa2s is called to find an approximating cubic spline surface for two-dimensional input data. If you want to weight the input data values, calculate derivatives, or handle data sparse areas specially, then you will need to use c_csa2xs.

Function prototype:

```
float *c_csa2s(int, float [], float [], float [], int [],
```

```
                int, int, float [], float [], int *);
```

Return value: c_csa2s returns a pointer to a linear array of data that is the approximated grid stored in row-major order. That is, if *out* is declared as

```
float *out;
```

and we set:

```
out = c_csa2s(n, x, y, z, knots, no, mo, xo, yo, &ier);
```

then *out*[i**mo*+j] is the approximated function value at coordinate point (*xo*[i], *yo*[j]) for 0 <= i < *no* and 0 <= j < *mo*. The space for *out* is allocated internal to c_csa2s and is *no* * *mo* floats in size.

Argument description:

```
-------------------------------------------------
                Argument | Type       |  Size
-------------------------------------------------
float *c_csa2s (n,       | int        |
                xi,      | float []   | n
                yi,      | float []   | n
                zi,      | float []   | n
                knots,   | int []     | 2
                no,      | int        |
                mo,      | int        |
                xo,      | float []   | no
                yo,      | float []   | mo
                ier      | int *      |
               );
-------------------------------------------------
```

n
: The number of input data points. It must be that n > 3 and, depending on the size of *knots* below, n may have to be larger.

xi
: An array containing the X coordinate values for the input function.

yi
: An array containing the Y coordinate values for the input function.

zi
: An array containing the functional values of the input function -- zi[k] is the functional value at (xi[k], yi[k]) for k=0,n-1.

knots
: The number of knots to be used in constructing the approximation spline. knots[0] specifies the number of knots in the X direction and knots[1] specifies the number of knots in the Y direction. knots[0] and knots[1] must each be at least 4. The larger the value for knots, the closer the approximated curve will come to passing through the input function values.

no
: The number of X coordinate values to be calculated for the output surface.

mo
: The number of Y coordinate values to be calculated for the output surface.

xo
: An array containing the X coordinate values for the output grid.

yo

An array containing the Y coordinate values for the output grid.

ier

An error return value. If *ier is returned as 0, then no errors were detected. If *ier is non-zero, then refer to the error list in the error table for details.

c_csa2xs - expanded entry for two-dimensional input, gridded output

---

c_csa2xs is called if you want to weight the input data values, calculate derivatives, or handle data sparse areas specially. If you do not want to do any of these three things, then use c_csa2s.

Function prototype:

```
float *c_csa2xs(int, float [], float [], float [], float [], int [], float,
                int [], int, int, float [], float [], int *);
```

Return value:

c_csa2xs returns a pointer to a linear array of data that is the approximated function on a grid stored in row-major order. That is, if *out* is declared as

```
float *out;
```

and we set:

```
out = c_csa2xs(ni, xi, yi, zi, wts, knots, smth, nderiv,
               no, mo, xo, yo, &ier);
```

then *out*[i**mo*+j] is the approximated function value at coordinate point (*xo*[i], *yo*[j]) for 0 <= i < *no* and 0 <= j < *mo*. The space for *out* is allocated internal to c_csa2s and is *no* * *mo* floats in size.

Argument description:

```
-------------------------------------------------
                  Argument | Type      |  Size
-------------------------------------------------
float *c_csa2xs (ni,       | int       |
                 xi,       | float []  | ni
                 yi,       | float []  | ni
                 zi,       | float []  | ni
                 wts,      | float []  | ni
                 knots,    | int []    | 2
                 smth,     | float     |
                 nderiv    | int []    | 2
                 no,       | int       |
                 mo,       | int       |
                 xo,       | float []  | no
                 yo,       | float []  | mo
                 ier       | int *     |
                );
```

---

ni
: The number of input data points. It must be that ni > 3 and, depending on the size of *knots* below, ni may have to be larger.

xi
: An array containing the X coordinate values for the input function.

yi
: An array containing the Y coordinate values for the input function.

zi
: An array containing the functional values of the input function -- zi[k] is the functional value at (xi[k],yi[k]) for k=0,n-1.

wts
: An array containing weights for the zi values at the input xi and yi values, that is, wts[k] is a weight for the value of zi[k] for k=0,ni-1. If you do not desire to weight the input yi values, then set wts[0] to -1. The weights in the wts array are relative and may be set to any non-negative value. When c_csa2xs is called, the weights are summed and the individual weights are normalized so that the weight sum is unity.

knots
: The number of knots to be used in constructing the approximation spline. knots[0] specifies the number of knots in the X direction and knots[1] specifies the number of knots in the Y direction. knots[0] and knots[1] must each be at least 4. The larger the value for knots, the closer the approximated curve will come to passing through the input function values.

smth
: A parameter that controls extrapolation into data sparse regions. If smth is 0., then nothing special is done in data sparse regions. A good first choice for smth is 1.

nderiv
: For each of the two coordinate directions, specifies whether you want functional values (nderiv=0), first derivative values (nderiv=1), or second derivative values (nderiv=2). For example, if nderiv[0]=1 and nderiv[1]=1, then the second order mixed partial would be computed.

no
: The number of X coordinate values to be calculated for the output surface.

mo
: The number of Y coordinate values to be calculated for the output surface.

xo
: An array containing the X coordinate values for the output grid.

yo
: An array containing the Y coordinate values for the output grid.

ier
: An error return value. If *ier is returned as 0, then no errors were detected. If *ier is non-zero, then refer to the error list in the error table for details.

c_csa2ls - simple entry for two-dimensional input, list output

---

c_csa2ls is called to find values of an approximating cubic spline at specified two-dimensional coordinates. If you want to weight the input data values, calculate derivatives, or handle data sparse areas specially, then you will need to use c_csa2lxs.

Function prototype:

```
float *c_csa2ls(int, float [], float [], float [], int [],
                int, float [], float [], int *);
```

Return value: c_csa2ls returns a pointer to a linear array of data that contains the approximated values calculated at the input list of coordinate values. That is, if *out* is declared as

```
  float *out;
```

and we set:

```
  out = c_csa2ls(n, x, y, z, knots, no, xo, yo, &ier);
```

then *out*[i] is the approximated function value at coordinate point (*xo*[i], *yo*[i]) for 0 <= i < *no*. The space for *out* is allocated internal to c_csa2ls and is *no* floats in size.

Argument description:

```
-------------------------------------------------
                 Argument  | Type       |  Size
-------------------------------------------------
float *c_csa2ls (n,        | int        |
                 xi,       | float []   | n
                 yi,       | float []   | n
                 zi,       | float []   | n
                 knots,    | int        | 2
                 no,       | int        |
                 xo,       | float []   | no
                 yo,       | float []   | no
                 ier       | int *      |
                 );
-------------------------------------------------
```

n
: The number of input data points. It must be that n > 3 and, depending on the size of *knots* below, n may have to be larger.

xi
: An array containing the X coordinate values for the input function.

yi
: An array containing the Y coordinate values for the input function.

zi
: An array containing the functional values of the input function -- zi[k] is the functional value at (xi[k], yi[k]) for k=0,n-1.

knots
: The number of knots to be used in constructing the approximation spline. knots[0] specifies the number of knots in the X direction and knots[1] specifies the number of knots in the Y direction. knots[0] and knots[1] must each be at least 4. The larger the value for knots, the closer the approximated curve will come to passing through the input function values.

no
: The number of X - Y coordinate values to be calculated for the output array.

xo

An array containing the X coordinate values for the output array.

yo

An array containing the Y coordinate values for the output array.

ier

An error return value. If *ier is returned as 0, then no errors were detected. If *ier is non-zero, then refer to the error list in the error table for details.

c_csa2lxs - expanded entry for two-dimensional input, list output

---

c_csa2lxs is called to find values of an approximating cubic spline at specified two-dimensional coordinates. c_csa2lxs is called if you want to weight the input data values, calculate derivatives, or handle data sparse areas specially. If you do not want to do any of these three things, then use c_csa2ls.

Function prototype:

```
float *c_csa2lxs(int, float [], float [], float [], float [], int [],
                 float, int [], int, float [], float [], int *);
```

Return value: c_csa2lxs returns a pointer to a linear array of data that contains the approximated values calculated at the input list of coordinate values. That is, if *out* is declared as

```
  float *out;
```

and we set:

```
  out = c_csa2lxs(n, x, y, z, wts, knots, smth, nderiv, no, xo, yo, &ier);
```

then *out*[i] is the approximated function value at coordinate point (*xo*[i],*yo*[i]) for 0 <= i < *no*. The space for *out* is allocated internal to c_csa2lxs and is *no* floats in size.

Argument description:

```
-------------------------------------------------
                  Argument | Type       |  Size
-------------------------------------------------
float *c_csa2lxs (n,       | int        |
                  xi,      | float []   | n
                  yi,      | float []   | n
                  zi,      | float []   | n
                  wts,     | float []   | n
                  knots,   | int []     | 2
                  smth,    | float      |
                  nderiv,  | int []     | 2
                  no,      | int        |
                  xo,      | float []   | no
                  yo,      | float []   | no
                  ier      | int *      |
                 );
-------------------------------------------------
```

n

The number of input data points. It must be that n > 3 and, depending on the size of *knots* below, n may have to be larger.

xi
: An array containing the X coordinate values for the input function.

yi
: An array containing the Y coordinate values for the input function.

zi
: An array containing the functional values of the input function -- zi[k] is the functional value at (xi[k],yi[k]) for k=0,n-1.

wts
: An array containing weights for the zi values at the input xi and yi values, that is, wts[l] is a weight for the value of zi[l] for l=0,n-1. If you do not desire to weight the input zi values, then set wts[0] to -1. The weights in the wts array are relative and may be set to any non-negative value. When c_csa2lxs is called, the weights are summed and the individual weights are normalized so that the weight sum is unity.

knots
: The number of knots to be used in constructing the approximation spline. knots[0] specifies the number of knots in the X direction and knots[1] specifies the number of knots in the Y direction. knots[0] and knots[1] must each be at least 4. The larger the value for knots, the closer the approximated curve will come to passing through the input function values.

smth
: A parameter that controls extrapolation into data sparse regions. If smth is zero, then nothing special is done in data sparse regions. A good first choice for smth is 1.

nderiv
: For each of the two coordinate direction, specifies whether you want functional values (nderiv=0), first derivative values (nderiv=1), or second derivative values (nderiv=2). For example, if nderiv[0]=1 and nderiv[1]=1, then the second order mixed partial would be computed.

no
: The number of X - Y coordinate values to be calculated for the output array.

xo
: An array containing the X coordinate values for the output array.

yo
: An array containing the Y coordinate values for the output array.

ier
: An error return value. If *ier is returned as 0, then no errors were detected. If *ier is non-zero, then refer to the error list in the error table for details.

c_csa3s - simple entry for three-dimensional input, gridded output

---

c_csa3s is called to find an approximating cubic spline for three-dimensional input data. If you want to weight the input data values, calculate derivatives, or handle data sparse areas specially, then you will need to use c_csa3xs.

Function prototype:

```
float *c_csa3s(int, float [], float [], float [], float [], int [], int, int,
               int, float [], float [], float [], int *);
```

Return value: c_csa3s returns a pointer to a linear array of data that is the approximation spline stored in row-major order. That is, if out is declared as

```
float *out;
```

and we set:

```
out = c_csa3s(n, x, y, z, u, knots, nx, ny, nz, xo, yo, zo, &ier);
```

then out[nz*ny*i + nz*j + k] is the approximation function value at coordinate point (xo[i], yo[j], zo[k]) for 0 <= i < nx, 0 <= j < ny, and 0 <= k < nz. The space for out is allocated internal to c_csa3s and is nx*ny*nz floats in size.

Argument description:

```
-------------------------------------------------
                Argument | Type       |  Size
-------------------------------------------------
float *c_csa3s (ni,      | int        |
                xi,      | float []   | ni
                yi,      | float []   | ni
                zi,      | float []   | ni
                ui,      | float []   | ni
                knots,   | int []     | 3
                nxo,     | int        |
                nyo,     | int        |
                nzo,     | int        |
                xo,      | float []   | nxo
                yo,      | float []   | nyo
                zo,      | float []   | nzo
                ier      | int *      |
               );
-------------------------------------------------
```

ni
: The number of input data points. It must be that ni > 3 and, depending on the size of *knots* below, ni may have to be larger.

xi
: An array containing the X coordinate values for the input function.

yi
: An array containing the Y coordinate values for the input function.

zi
: An array containing the Z coordinate values for the input function.

ui
: An array containing the functional values of the input function -- ui[k] is the functional value at (xi[k], yi[k], zi[k]) for k=0,ni-1.

knots
: The number of knots to be used in constructing the approximation spline. knots[0] specifies the number of knots in the X direction, knots[1] specifies the number of knots in the Y direction and knots[2] specifies the number of knots in the Z direction. knots[0], knots[1] and knots[2] must each be at least 4. The larger the value for knots, the closer the approximated curve will come to passing through the input function values.

nxo
: The number of X coordinate values to be calculated for the output grid.

nyo
: The number of Y coordinate values to be calculated for the output grid.

nzo
: The number of Z coordinate values to be calculated for the output grid.

xo
: An array containing the X coordinate values for the output grid.

yo
: An array containing the Y coordinate values for the output grid.

zo
: An array containing the Z coordinate values for the output grid.

ier
: An error return value. If *ier is returned as 0, then no errors were detected. If *ier is non-zero, then refer to the error list in the error table for details.

c_csa3xs - expanded entry for three-dimensional input, gridded output

---

c_csa3xs is called to find an approximating cubic spline surface for three-dimensional input data. c_csa3xs is called if you want to weight the input data values, calculate derivatives, or handle data sparse areas specially. If you do not want to do any of these three things, then use c_csa3s.

Function prototype:

```
float *c_csa3xs(int, float [], float [], float [], float [], float [],
                int [], float, int [], int, int, int, float [],
                float [], float [], int *);
```

Return value: c_csa3xs returns a pointer to a linear array of data that is the approximated function on a grid stored in row-major order. That is, if *out* is declared as

```
  float *out;
```

and we set:

```
out = c_csa3xs(ni, xi, yi, zi, ui, wts, knots, smth, nderiv,
               nxo, nyo, nzo, xo, yo, zo, &ier)
```

then out[nz*ny*i + nz*j + k] is the approximation function value at coordinate point (xo[i], yo[j], zo[k]) for 0 <= i < nx, 0 <= j < ny, and 0 <= k < nz. The space for out is allocated internal to c_csa3xs and is nx*ny*nz floats in size.

Argument description:

```
-------------------------------------------------
                 Argument | Type      |  Size
-------------------------------------------------
float *c_csa3xs (ni,      | int       |
```

```
                xi,        | float [] | ni
                yi,        | float [] | ni
                zi,        | float [] | ni
                ui,        | float [] | ni
                wts,       | float [] | ni
                knots,     | int []   | 3
                smth,      | float    |
                nderiv     | int []   | 3
                nxo,       | int      |
                nyo,       | int      |
                nzo,       | int      |
                xo,        | float [] | nxo
                yo,        | float [] | nyo
                yo,        | float [] | nzo
                ier        | int *    |
               );
---------------------------------------------------
```

ni
: The number of input data points. It must be that ni > 3 and, depending on the size of *knots* below, ni may have to be larger.

xi
: An array containing the X coordinate values for the input function.

yi
: An array containing the Y coordinate values for the input function.

zi
: An array containing the Z coordinate values for the input function.

ui
: An array containing the functional values of the input function -- ui[k] is the functional value at (xi[k],yi[k],zi[k]) for k=0,n-1.

wts
: An array containing weights for the ui values at the input values, that is, wts[l] is a weight for the value of ui[l] for l=0,ni-1. If you do not desire to weight the input ui values, then set wts[0] to -1. The weights in the wts array are relative and may be set to any non-negative value. When c_csa3xs is called, the weights are summed and the individual weights are normalized so that the weight sum is unity.

knots
: The number of knots to be used in constructing the approximation spline. knots[0] specifies the number of knots in the X direction, knots[1] specifies the number of knots in the Y direction and knots[2] specifies the number of knots in the Z direction. knots[0], knots[1], and knots[2] must each be at least 4. The larger the value for knots, the closer the approximated curve will come to passing through the input function values.

smth
: A parameter that controls extrapolation into data sparse regions. If smth is zero, then nothing special is done in data sparse regions. A good first choice for smth is 1.

nderiv
: For each of the three coordinate direction, specifies whether you want functional values (nderiv=0), first derivative values (nderiv=1), or second derivative values (nderiv=2). For example, if nderiv[0]=1, nderiv[1]=1, and nderiv[2]=0, then the second order mixed partial with respect to X and Y would be computed.

nxo
: The number of X coordinate values to be calculated for the output grid.

nyo
: The number of Y coordinate values to be calculated for the output grid.

nzo
: The number of Z coordinate values to be calculated for the output grid.

xo
: An array containing the X coordinate values for the output grid.

yo
: An array containing the Y coordinate values for the output grid.

zo
: An array containing the Z coordinate values for the output grid.

ier
: An error return value. If *ier is returned as 0, then no errors were detected. If *ier is non-zero, then refer to the error list in the error table for details.

c_csa3ls - simple entry for three-dimensional input, list output

---

c_csa3ls is called to find values of an approximating cubic spline at specified three-dimensional coordinates. If you want to weight the input data values, calculate derivatives, or handle data sparse areas specially, then you will need to use c_csa3lxs.

Function prototype:

```
float *c_csa3ls(int, float [], float [], float [], float [],
                int [], int, float [], float [], float[], int *);
```

Return value: c_csa3ls returns a pointer to a linear array of data that contains the approximated values calculated at the input list of coordinate values. That is, if *out* is declared as

```
  float *out;
```

and we set:

```
  out = c_csa3ls(n, x, y, z, u, knots, no, xo, yo, zo, &ier);
```

then *out*[i] is the approximated function value at coordinate point (*xo*[i],*yo*[i],*zo*[i]) for 0 <= i < *no*. The space for *out* is allocated internal to c_csa3ls and is *no* floats in size.

Argument description:

```
-------------------------------------------------
                 Argument | Type       |  Size
-------------------------------------------------
float *c_csa3ls (n,       | int        |
                 xi,      | float []   | n
                 yi,      | float []   | n
                 zi,      | float []   | n
                 ui,      | float []   | n
                 knots,   | int []     | 3
                 no,      | int        |
                 xo,      | float []   | no
                 yo,      | float []   | no
```

```
                 zo,      | float []  | no
                 ier      | int *     |
                );
-------------------------------------------------
```

n
: The number of input data points. It must be that n > 3 and, depending on the size of *knots* below, n may have to be larger.

xi
: An array containing the X coordinate values for the input function.

yi
: An array containing the Y coordinate values for the input function.

zi
: An array containing the Z coordinate values for the input function.

ui
: An array containing the functional values of the input function -- ui[k] is the functional value at (xi[k], yi[k], zi[k]) for k=0,n-1.

knots
: The number of knots to be used in constructing the approximation spline. knots[0] specifies the number of knots in the X direction, knots[1] specifies the number of knots in the Y direction and knots[2] specifies the number of knots in the Z direction. knots[0] and knots[1] must each be at least 4. The larger the value for knots, the closer the approximated curve will come to passing through the input function values.

no
: The number of X - Y - Z coordinate values to be calculated for the output array.

xo
: An array containing the X coordinate values for the output array.

yo
: An array containing the Y coordinate values for the output array.

zo
: An array containing the Z coordinate values for the output array.

ier
: An error return value. If *ier is returned as 0, then no errors were detected. If *ier is non-zero, then refer to the error list in the error table for details.

c_csa3lxs - expanded entry for three-dimensional input, list output

---

c_csa3lxs is called to find values of an approximating cubic spline at specified three-dimensional coordinates. c_csa3lxs is called if you want to weight the input data values, calculate derivatives, or handle data sparse areas specially. If you do not want to do any of these three things, then use c_csa3ls.

Function prototype:

```
float *c_csa3lxs(int, float [], float [], float [], float [],
                 float [], int [], float, int [],
                 int, float [], float [], float [], int *);
```

Return value: c_csa3lxs returns a pointer to a linear array of data that contains the approximated values calculated at the input list of coordinate values. That is, if *out* is declared as

```
float *out;
```

and we set:

```
out = c_csa3lxs(n, x, y, z, u, wts, knots, smth, nderiv, no, xo, yo, zo, &ier);
```

then *out*[i] is the approximated function value at coordinate point (*xo*[i],*yo*[i],*zo*[i]) for 0 <= i < *no*. The space for *out* is allocated internal to c_csa2lxs and is *no* floats in size.

Argument description:

```
-------------------------------------------------
                  Argument | Type      |  Size
-------------------------------------------------
float *c_csa3lxs (ni,      | int       |
                  xi,      | float []  | ni
                  yi,      | float []  | ni
                  zi,      | float []  | ni
                  ui,      | float []  | ni
                  wts,     | float []  | ni
                  knots,   | int []    | 3
                  smth,    | float     |
                  nderiv,  | int []    | 3
                  no,      | int       |
                  xo,      | float []  | no
                  yo,      | float []  | no
                  zo,      | float []  | no
                  ier      | int *     |
                 );
-------------------------------------------------
```

ni
: The number of input data points. It must be that ni > 3 and, depending on the size of *knots* below, ni may have to be larger.

xi
: An array containing the X coordinate values for the input function.

yi
: An array containing the Y coordinate values for the input function.

zi
: An array containing the Y coordinate values for the input function.

ui
: An array containing the functional values of the input function -- ui[k] is the functional value at (xi[k], yi[k], zi[k]) for k=0,n-1.

wts
: An array containing weights for the ui values at the input values, that is, wts[l] is a weight for the value of ui[l] for l=0,n-1. If you do not desire to weight the input yi values, then set wts[0] to -1. The weights in the wts array are relative and may be set to any non-negative value. When c_csa3lxs is called, the weights are summed and the individual weights are normalized so that the weight sum is unity.

knots

The number of knots to be used in constructing the approximation spline. knots[0] specifies the number of knots in the X direction, knots[1] specifies the number of knots in the Y direction and knots[2] specifies the number of knots in the Z direction. knots[0], knots[1], and knots[2] must each be at least 4. The larger the value for knots, the closer the approximated curve will come to passing through the input function values.

smth
: A parameter that controls extrapolation into data sparse regions. If smth is zero, then nothing special is done in data sparse regions. A good first choice for smth is 1.

nderiv
: For each of the three coordinate direction, specifies whether you want functional values (nderiv=0), first derivative values (nderiv=1), or second derivative values (nderiv=2). For example, if nderiv[0]=1, nderiv[1]=1, and nderiv[2]=0, then the second order mixed partial with respect to X and Y would be computed.

no
: The number of X - Y - Z coordinate values to be calculated for the output array.

xo
: An array containing the X coordinate values for the output array.

yo
: An array containing the Y coordinate values for the output array.

zo
: An array containing the Z coordinate values for the output array.

ier
: An error return value. If *ier is returned as 0, then no errors were detected. If *ier is non-zero, then refer to the error list in the error table for details.

Csagrid timing table

In the limit, the execution times for Csagrid are linearly proportional to the number of input data points (doubling the number of input data points will double the execution time) and are proportional to the product of the squares of the number of knots (doubling the number of knots in any coordinate direction will increase the execution time by a factor of four). The number of output points is not an extremely important factor in the timings.

The following table is offered as an example of execution times. The times were obtained on a Sun sparc Ultra-1 running SunOS 5.5.1 using version SC4.2 of the C compiler:

The example used to produced the times was a two-dimensional problem.

Number of input points	Number of output points	Knots	Time in Secs.
400	1000	4,4	0.15
400	4000	4,4	0.35
400	16000	4,4	1.12
400	64000	4,4	4.23
1600	1000	4,4	0.40
1600	4000	4,4	0.60
1600	1600	4,4	1.38
1600	64000	4,4	4.51
6400	1000	4,4	1.42
6400	4000	4,4	1.62
6400	16000	4,4	2.40
6400	64000	4,4	5.51
25600	1000	4,4	5.52
25600	4000	4,4	5.72
25600	16000	4,4	6.50
25600	64000	4,4	9.62
102400	1000	4,4	21.96
102400	4000	4,4	22.16
102400	16000	4,4	22.95
102400	64000	4,4	26.21

Number of input points	Number of output points	Knots	Time in Secs.
400	1000	4,8	0.30
400	4000	4,8	0.53
400	16000	4,8	1.41
400	64000	4,8	4.94
1600	1000	4,8	0.98
1600	4000	4,8	1.20
1600	1600	4,8	2.08
1600	64000	4,8	5.56
6400	1000	4,8	3.74
6400	4000	4,8	3.96
6400	16000	4,8	4.82
6400	64000	4,8	8.34
25600	1000	4,8	14.77
25600	4000	4,8	14.99
25600	16000	4,8	15.87
25600	64000	4,8	19.36
102400	1000	4,8	59.00
102400	4000	4,8	59.19
102400	16000	4,8	60.09
102400	64000	4,8	63.64

Number of input points	Number of output points	Knots	Time in Secs.
400	1000	8,4	0.29
400	4000	8,4	0.52
400	16000	8,4	1.38
400	64000	8,4	4.87
1600	1000	8,4	0.99
1600	4000	8,4	1.22
1600	1600	8,4	2.09
1600	64000	8,4	5.56
6400	1000	8,4	3.73
6400	4000	8,4	3.96
6400	16000	8,4	4.83
6400	64000	8,4	8.31
25600	1000	8,4	14.79
25600	4000	8,4	15.01
25600	16000	8,4	15.87
25600	64000	8,4	19.37
102400	1000	8,4	58.92
102400	4000	8,4	59.12
102400	16000	8,4	60.00
102400	64000	8,4	63.49

Number of input points	Number of output points	Knots	Time in Secs.
400	1000	8,8	0.81
400	4000	8,8	1.06
400	16000	8,8	2.02
400	64000	8,8	5.90
1600	1000	8,8	3.16
1600	4000	8,8	3.40
1600	1600	8,8	4.36
1600	64000	8,8	8.23
6400	1000	8,8	12.51
6400	4000	8,8	12.75
6400	16000	8,8	13.71
6400	64000	8,8	17.60
25600	1000	8,8	49.94
25600	4000	8,8	50.17
25600	16000	8,8	51.15
25600	64000	8,8	55.00
102400	1000	8,8	199.63
102400	4000	8,8	200.53
102400	16000	8,8	201.31
102400	64000	8,8	207.00

Csagrid Examples

Overview

This module contains source codes for Csagrid examples.

The following list summarizes the examples:

- Example 1 -- a one-dimensional example varying the number of knots.
- Example 2 -- a one-dimensional example illustrating input data weighting.
- Example 3 -- a one-dimensional example illustrating extrapolation into data sparse regions.
- Example 4 -- a two-dimensional example illustrating taking partial derivatives.
- Example 5 -- a three-dimensional example (sphere).
- Example 6 -- a three-dimensional example (tube).
- Example 7 -- a two-dimensional example using list output

The plotting subroutines used by the Fortran examples to plot surfaces and isosurfaces are TDEZ2D and TDEZ3D from the Tdpack package of NCAR Graphics. TDEZ2D is a simplified procedure for drawing a surface and TDEZ3D is a simplified procedure for drawing an isosurface. The C examples use the C functions, c_tdez2d.c and c_tdez3d.c from Tdpack - these are equivalent to the Fortran procedures. The NCL examples use the built-in functions tdez2d and tdez3d.

If you are not using NCAR Graphics and want to draw the plots, you will have to substitute appropriate calls from the graphics package you are using.

Creating your own executable

If you have access to NCAR Graphics, you can create executables for all of the Fortran and C examples in one of two ways. The easiest is to use the *ncargex* utility, which is documented with a man page. The other way is to use either *ncargf77* (for Fortran) or *ncargcc* (for C) to compile and execute the source codes.

If you are using NCAR Graphics and the C interface to Csagrid and you are compiling using *ncargcc*, then function prototypes for all Csagrid C functions are contained in a C header file. This can be accessed by inserting the line:

```
#include <ncarg/ngmath.h>
```

into your code.

If you are using NCAR Graphics, you can run all of the NCL examples for Csagrid using the *ng4ex* utility. The NCL examples utilize GSUN> functions.

Example 1 -- a one-dimensional example varying the number of knots

- Fortran single precision
- C single precision
- NCL
- Sample plot

Example 2 -- a one-dimensional example illustrating input data weighting

- Fortran single precision
- C single precision
- NCL
- Sample plot

Example 3 -- a one-dimensional example illustrating extrapolation into data sparse regions

- Fortran single precision
- C single precision
- NCL
- Sample plot

Example 4 -- a two-dimensional example illustrating taking partial derivatives

- Fortran single precision
- C single precision
- NCL
- Sample plot

Example 5 -- a three-dimensional example (sphere)

- Fortran single precision
- C single precision
- NCL
- Sample plot

Example 6 -- a three-dimensional example (tube)

- Fortran single precision
- C single precision
- NCL
- Sample plot

Example 7 -- a two-dimensional example using list output

- Fortran single precision
- C single precision
- NCL

- Sample plot

Example 1 - one-dimensional example varying the number of knots (C)

```
#include <stdio.h>
#include <ncarg/ncargC.h>
#include <ncarg/gks.h>
#include <ncarg/ngmath.h>

/*
 *  Function prototypes for plotting backgrounds and curves.
 */
void c_bkgft1(float, char *, float, float);
void c_drwft1(int, float [], float [], int, float [], float [],
              float [], float []);

/*
 *  The number of input data points.
 */
#define NDATA  10

/*
 *  The number of output data points.
 */
#define NPTS  101

/*
 *  The GKS workstation type (NCGM).
 */
#define IWTYPE  1

/*
 *  The GKS workstaton identifier.
 */
#define WKID    1

/*
 * This example illustrates the effects of using differing numbers
 * of knots in calls to c_csa1s with the same input data.
 */
main ()
{

/*
 *  Set up the output arrays.
 */
  float xo[NPTS],*yo4,*yo7,*yo9,xinc;

/*
 *  Define the input data.
 */
```

```
  float xi[] = {0.0, 0.1,  0.2,  0.3, 0.5,  0.6, 0.65,  0.8,  0.9, 1.};
  float yi[] = {0.0, 0.8, -0.9, -0.9, 0.9,  1.0, 0.90, -0.8, -0.8, 0.};

  int i,knots,ier;

/*
 *  Create the output X coordinate array.
 */
  xinc = 1./ (float) (NPTS-1);
  for (i = 0; i < NPTS; i++) {
    xo[i] = (float) i * xinc;
  }

/*
 *  Calculate the approximated function values using differing
 *  number of knots.
 */
  knots = 4;
  yo4 = c_csa1s(NDATA,xi,yi,knots,NPTS,xo,&ier);
  if (ier != 0) {
    printf("Error return from c_csa1s: %d\n",ier);
    exit(1);
  }
  knots = 7;
  yo7 = c_csa1s(NDATA,xi,yi,knots,NPTS,xo,&ier);
  if (ier != 0) {
    printf("Error return from c_csa1s: %d\n",ier);
    exit(1);
  }
  knots = 9;
  yo9 = c_csa1s(NDATA,xi,yi,knots,NPTS,xo,&ier);
  if (ier != 0) {
    printf("Error return from c_csa1s: %d\n",ier);
    exit(1);
  }

/*
 *  Draw plot.
 */
  c_drwft1(NDATA,xi,yi,NPTS,xo,yo4,yo7,yo9);

}

void c_drwft1(int n, float x[], float y[], int m, float xo[],
              float curve1[], float curve2[], float curve3[])
{
/*
 *  This function uses NCAR Graphics to plot three curves on
 *  the same picture showing the results from calling c_csa1x with
 *  differing number of knots.  The values for the curves are
 *  contained in arrays curve1, curve2, and curve3.
 */
  int   i;
  float yb, yt, ypos_top = 0.88;

/*
 *  Declare variables used in GKS calls.
 */
  Gcolr_rep rgb;
  Gpoint plist[NDATA];
  Gpoint_list pmk;
```

```
/*
 *  Open GKS, open and activate a workstation.
 */
  gopen_gks("stdout",0);
  gopen_ws(WKID, NULL, IWTYPE);
  gactivate_ws(WKID);

/*
 * Define a color table.
 */
  rgb.rgb.red = rgb.rgb.green = rgb.rgb.blue = 1.;
  gset_colr_rep(WKID,0,&rgb);
  rgb.rgb.red = rgb.rgb.green = rgb.rgb.blue = 0.;
  gset_colr_rep(WKID,1,&rgb);
  rgb.rgb.red = 1.;
  rgb.rgb.green = rgb.rgb.blue = 0.;
  gset_colr_rep(WKID,2,&rgb);
  rgb.rgb.red = rgb.rgb.green = 0.;
  rgb.rgb.blue = 1.;
  gset_colr_rep(WKID,3,&rgb);

/*
 * Plot the main title.
 */
  gset_clip_ind(0);
  c_plchhq(.5,.95,":F21:Demo for c_csa1s",0.035,0.,0.);

/*
 *  Draw a background grid for the first curve.
 */
  yb = -1.2;
  yt =  1.2;
  c_bkgft1(ypos_top,"knots = 4",yb,yt);
  c_gridal(5,5,4,1,1,1,10,0.0,yb);

/*
 *  Graph the approximated function values for knots=4.
 */
  c_curve(xo,curve1,m);

/*
 *  Mark the input data points.
 */
  for (i = 0; i < n; i++) {
    plist[i].x = x[i];
    plist[i].y = y[i];
  }
  gset_marker_size(2.2);
  gset_marker_colr_ind(3);
  pmk.num_points = n;
  pmk.points = plist;
  gpolymarker(&pmk);

/*
 *  Graph the approximated function values for knots=7.
 */
  yb = -1.2;
  yt =  1.2;
  c_bkgft1(ypos_top-0.3,"knots = 7",yb,yt);
  c_gridal(5,5,4,1,1,1,10,0.0,yb);
  c_curve(xo,curve2,m);
```

```
/*
 *  Mark the input data points.
 */
  for (i = 0; i < n; i++) {
    plist[i].x = x[i];
    plist[i].y = y[i];
  }
  gset_marker_size(2.2);
  gset_marker_colr_ind(3);
  pmk.num_points = n;
  pmk.points = plist;
  gpolymarker(&pmk);

/*
 *  Graph the approximated function values for knots=9.
 */
  yb = -1.2;
  yt =  1.2;
  c_bkgft1(ypos_top-0.6,"knots = 9",yb,yt);
  c_gridal(5,5,4,1,1,1,10,0.0,yb);
  c_curve(xo,curve3,m);

/*
 *  Mark the input data points.
 */
  for (i = 0; i < n; i++) {
    plist[i].x = x[i];
    plist[i].y = y[i];
  }
  gset_marker_size(2.2);
  gset_marker_colr_ind(3);
  pmk.num_points = n;
  pmk.points = plist;
  gpolymarker(&pmk);

  c_frame();

/*
 *  Deactivate and close workstation, close GKS.
 */
  gdeactivate_ws(WKID);
  gclose_ws(WKID);
  gclose_gks();
}

/*
 *  Draw a background.
 */
void c_bkgft1(float ypos, char *label, float yb, float yt) {
  c_set(0.,1.,0.,1.,0.,1.,0.,1.,1);

/*
 *   Plot the curve label using font 21 (Helvetica).
 */
  c_pcseti("fn",21);
  c_plchhq(.25,ypos-0.03,label,0.025,0.,-1.0);

/*
 *  Draw a horizontal line at Y=0. using color index 2.
 */
  c_set(0.13,0.93,ypos-0.2,ypos,0.0,1., yb, yt, 1);
```

```
  gset_line_colr_ind(2);
  c_line(0.,0.,1.,0.);
  c_sflush();
  gset_line_colr_ind(1);

/*
 *  Set Gridal parameters.
 *
 *
 *   Set lty to indicate that the Plotchar routine PLCHHQ should be used.
 */
  c_gaseti("lty",1);

/*
 *   Size and format for X axis labels.
 */
  c_gasetr("xls",0.02);
  c_gasetc("xlf","(f3.1)");

/*
 *   Size and format for X axis labels.
 */
  c_gasetr("yls",0.02);
  c_gasetc("ylf","(f5.1)");

/*
 *  Length of major tick marks for the X and Y axes.
 */
  c_gasetr("xmj",0.02);
  c_gasetr("ymj",0.02);
}
```

Example 1 - one-dimensional example varying the number of knots (Fortran)

```
      PROGRAM CSEX01
C
C  This example illustrates the effects of using differing numbers
C  of knots in calls to CSA1S with the same input data.
C
C
C  The dimensionality of the problem.
C
      PARAMETER (NDIM=1)
C
C  The number of input data points.
C
      PARAMETER (NDATA=10)
C
C  The number of output data points.
C
      PARAMETER (NPTS=101)
C
```

```
C  The maximum number of knots used in any call.
C
      PARAMETER (NCF=9)
C
C  The size of the workspace.
C
      PARAMETER (NWRK=NCF*(NCF+3))
C
C  Dimension arrays.
C
      DIMENSION XDATA(NDIM,NDATA),YDATA(NDATA),XDATAT(NDATA)
      DIMENSION WORK(NWRK)
      DIMENSION XO(NPTS),YO4(NPTS),YO7(NPTS),YO9(NPTS)
C
C  Define the original data.
C
      DATA XDATA(NDIM,1),XDATA(NDIM,2),XDATA(NDIM,3)/ 0.00, 0.1, 0.2 /
      DATA XDATA(NDIM,4),XDATA(NDIM,5),XDATA(NDIM,6)/ 0.30, 0.5, 0.6 /
      DATA XDATA(NDIM,7),XDATA(NDIM,8),XDATA(NDIM,9)/ 0.65, 0.8, 0.9 /
      DATA XDATA(NDIM,10)                           / 1.00           /
      DATA YDATA/0.0, 0.8, -0.9, -0.9, 0.9, 1.0, 0.90, -0.8, -0.8, 0./
C
C  Create the output X coordinate array.
C
      XINC = 1./(NPTS-1)
      DO 10 I=1,NPTS
        XO(I) = (I-1)*XINC
   10 CONTINUE
C
C  Calculate the approximated function values using differing
C  number of knots.
C
      KNOTS = 4
      CALL CSA1S (NDATA,XDATA,YDATA,KNOTS,NPTS,XO,YO4,NWRK,WORK,IER)
      IF (IER .NE. 0) THEN
        WRITE(6,520) IER
  520   FORMAT(' Error ',I3,' returned from CSA1S')
        STOP
      ENDIF
C
      KNOTS = 7
      CALL CSA1S (NDATA,XDATA,YDATA,KNOTS,NPTS,XO,YO7,NWRK,WORK,IER)
      IF (IER .NE. 0) THEN
        WRITE(6,520) IER
        STOP
      ENDIF
C
      KNOTS = 9
      CALL CSA1S (NDATA,XDATA,YDATA,KNOTS,NPTS,XO,YO9,NWRK,WORK,IER)
      IF (IER .NE. 0) THEN
        WRITE(6,520) IER
        STOP
      ENDIF
C
C  Draw a plot of the approximation functions and mark the original points.
C
      DO 34 I=1,NDATA
        XDATAT(I) = XDATA(1,I)
   34 CONTINUE
C
      CALL DRWFT1(NDATA,XDATAT,YDATA,NPTS,XO,YO4,YO7,YO9)
```

```
C
      STOP
      END
      SUBROUTINE DRWFT1(NUMO,X,Y,IO,XO,CURVE1,CURVE2,CURVE3)
C
C  This subroutine uses NCAR Graphics to plot three curves on
C  the same picture showing the results from calling CSA1X with
C  differing number of knots.  The values for the curves are
C  contained in arrays CURVE1, CURVE2, and CURVE3.
C
      DIMENSION XO(IO),CURVE1(IO),CURVE2(IO),CURVE3(IO)
C
C  Define error file, Fortran unit number, workstation type
C  and workstation ID.
C
      PARAMETER (IERRF=6, LUNIT=2, IWTYPE=1, IWKID=1)
C
C  Vertical position for initial curve.
C
      DATA YPOS_TOP/0.88/
C
C  Open GKS, open and activate a workstation.
C
      CALL GOPKS (IERRF, ISZDM)
      CALL GOPWK (IWKID, LUNIT, IWTYPE)
      CALL GACWK (IWKID)
C
C  Define a color table.
C
      CALL GSCR(IWKID, 0, 1.0, 1.0, 1.0)
      CALL GSCR(IWKID, 1, 0.0, 0.0, 0.0)
      CALL GSCR(IWKID, 2, 1.0, 0.0, 0.0)
      CALL GSCR(IWKID, 3, 0.0, 0.0, 1.0)
      CALL GSCLIP(0)
C
C  Plot the main title.
C
      CALL PLCHHQ(0.50,0.95,':F21:Demo for csa1s',0.035,0.,0.)
C
C  Draw a background grid for the first curve.
C
      YB = -1.2
      YT =  1.2
      CALL BKGFT1(YPOS_TOP,'knots = 4',YB,YT)
      CALL GRIDAL(5,5,4,1,1,1,10,0.0,YB)
C
C  Mark the original data points.
C
      CALL GSMKSC(2.2)
      CALL GSPMCI(3)
      CALL GSLWSC(1.)
      CALL GPM(NUMO,X,Y)
C
C  Graph the approximated function values for KNOTS=4.
C
      CALL GSPLCI(1)
      CALL GPL(IO,XO,CURVE1)
C
C  Graph the approximated function values for KNOTS=7.
C
      CALL BKGFT1(YPOS_TOP-0.3,'knots = 7',YB,YT)
```

```
      CALL GRIDAL(5,5,4,1,1,1,10,0.0,YB)
      CALL GPM(NUMO,X,Y)
      CALL GPL(IO,XO,CURVE2)
      CALL GSPLCI(1)
C
C  Graph the approximated function values for KNOTS=9.
C
      CALL BKGFT1(YPOS_TOP-0.6,'knots = 9',YB,YT)
      CALL GRIDAL(5,5,4,1,1,1,10,0.0,YB)
      CALL GPM(NUMO,X,Y)
      CALL GPL(IO,XO,CURVE3)
      CALL GSPLCI(1)
      CALL FRAME
C
      CALL GDAWK(IWKID)
      CALL GCLWK(IWKID)
      CALL GCLKS
C
      RETURN
      END
      SUBROUTINE BKGFT1(YPOS,LABEL,YB,YT)
C
C  This subroutine draws a background grid.
C
      DIMENSION XX(2),YY(2)
      CHARACTER*(*) LABEL
C
      CALL SET(0.,1.,0.,1.,0.,1.,0.,1.,1)
C
C  Plot the curve label using font 21 (Helvetica).
C
      CALL PCSETI('FN',21)
      CALL PLCHHQ(0.25,YPOS - 0.03,LABEL,0.025,0.,-1.)
      CALL SET(0.13,0.93,YPOS-0.2,YPOS,0.0,1., YB, YT, 1)
C
C  Draw a horizontal line at Y=0. using color index 2.
C
      XX(1) = 0.
      XX(2) = 1.
      YY(1) = 0.
      YY(2) = 0.
      CALL GSPLCI(2)
      CALL GPL(2,XX,YY)
      CALL GSPLCI(1)
C
C  Set Gridal parameters.
C
C
C   Set LTY to indicate that the Plotchar routine PLCHHQ should be used.
C
      CALL GASETI('LTY',1)
C
C   Size and format for X axis labels.
C
      CALL GASETR('XLS',0.02)
      CALL GASETC('XLF','(F3.1)')
C
C   Size and format for Y axis labels.
C
      CALL GASETR('YLS',0.02)
      CALL GASETC('YLF','(F5.1)')
```

```
C
C   Length of major tick marks for the X and Y axes.
C
      CALL GASETR('XMJ',0.02)
      CALL GASETR('YMJ',0.02)
C
      RETURN
      END
```

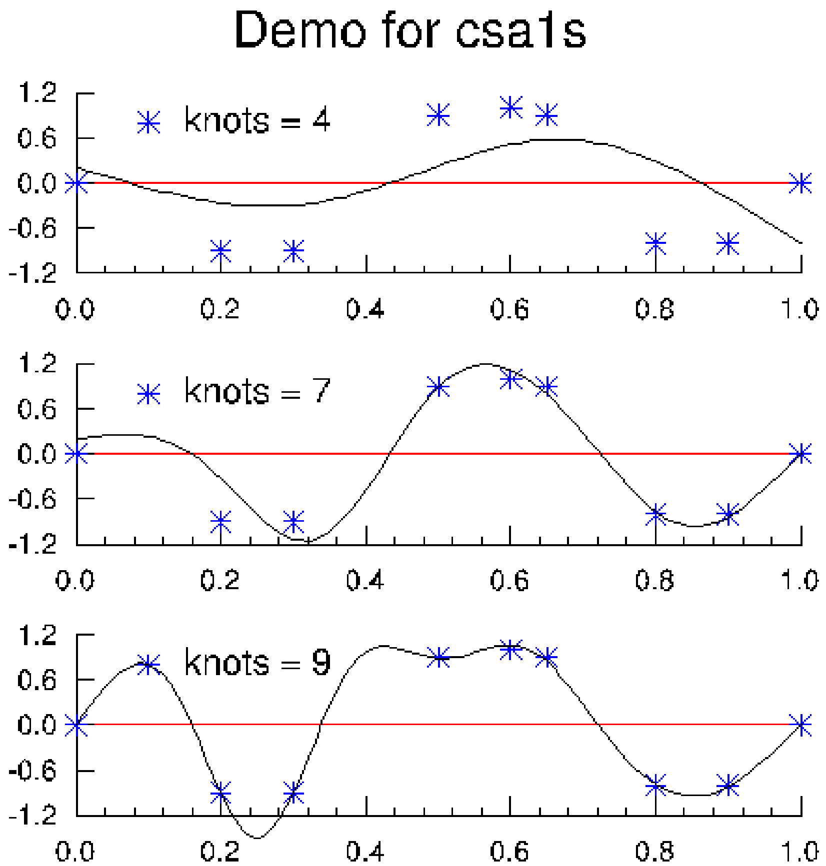
Example 1 - one-dimensional example varying the number of knots (NCL)

```
load "$NCARG_ROOT/lib/ncarg/nclex/gsun/gsn_code.ncl"

begin

;----------- Define original data --------------------------------------

  xi = (/0.0, 0.1,  0.2,  0.3, 0.5,  0.6, 0.65,  0.8,  0.9, 1./)
  yi = (/0.0, 0.8, -0.9, -0.9, 0.9,  1.0, 0.90, -0.8, -0.8, 0./)

  npts   = 101
  xo     = fspan(0.0,1.0,npts)  ; Create the output X coordinate array.

  xx = (/0.0, 1.0/) ; Create data for a polyline for marking
  yy = (/0.0, 0.0/) ; the Y = 0.0 line in each graph.

;--- Calculate approximated function values using differing number of knots.

  knots = 4
  yo4 = csa1s(xi,yi,knots,xo)
  knots = 7
  yo7 = csa1s(xi,yi,knots,xo)
  knots = 9
  yo9 = csa1s(xi,yi,knots,xo)

;---- Open workstation, define color map --------------------------

  NCGM=1
  X11=0
  PS=0

  if (NCGM .eq. 1) then
    wks = gsn_open_wks("ncgm","nm13n")
  end if
  if (X11 .eq. 1) then
    wks = gsn_open_wks("x11","nm13n")
  end if
  if (PS .eq. 1) then
    wks = gsn_open_wks("ps","nm13n")
  end if

  cmap = (/                 \
```

```
           (/1.,1.,1./),   \
           (/0.,0.,0./),   \
           (/1.,0.,0./),   \
           (/0.,0.,1./)    \
        /)
  gsn_define_colormap(wks,cmap)

;---- Specify resources ---------------------------

  txres = True   ; Set up some variables for
  xyres = True   ; creating three different
  gsres = True   ; resource lists.

  xyres@gsnFrame               = False     ; Don't advance the frame.

  xyres@tmXTBorderOn           = False     ; Don't draw top axis.
  xyres@tmXTOn                 = False     ; Don't draw top axis tick marks.
  xyres@tmBorderThicknessF     = 1.0       ; Default thickness is 2.0

  xyres@tmXBLabelFont          = 21        ; Change X label font.
  xyres@tmXBLabelFontHeightF   = 0.025     ; Change font size.
  xyres@tmXBMajorLengthF       = 0.015     ; Default is 0.02.
  xyres@tmXBMajorThicknessF    = 1.0       ; Default is 2.
  xyres@tmXBMinorLengthF       = 0.0075    ; Default is 0.01.
  xyres@tmXBMinorPerMajor      = 4         ; # of minor tick marks per major.
  xyres@tmXBMode               = "Manual"  ; Set tick mark mode.
  xyres@tmXBTickStartF         = 0.0
  xyres@tmXBTickEndF           = 1.0
  xyres@tmXBTickSpacingF       = 0.2
  xyres@tmXMajorGridThicknessF = 1.0       ; Default is 2.0

  xyres@tmYLLabelFont          = 21        ; See explanations for X axes
  xyres@tmYLLabelFontHeightF   = 0.025     ; resources.
  xyres@tmYLMajorLengthF       = 0.015
  xyres@tmYLMajorThicknessF    = 1.0
  xyres@tmYLMinorPerMajor      = 0
  xyres@tmYLMode               = "Manual"
  xyres@tmYLTickStartF         = -1.2
  xyres@tmYLTickEndF           =  1.2
  xyres@tmYLTickSpacingF       =  0.6
  xyres@tmYRBorderOn           = False     ; Don't draw right axis.
  xyres@tmYROn                 = False     ; Don't draw right axis tick marks.

  xyres@trXMaxF   =  1.0                   ; Specify data limits.
  xyres@trXMinF   =  0.0
  xyres@trYMaxF   =  1.2
  xyres@trYMinF   = -1.2

  xyres@vpHeightF = 0.20                   ; Specify size and location of plot.
  xyres@vpWidthF  = 0.80
  xyres@vpXF      = 0.13
  xyres@vpYF      = 0.88

;---------- First graph ----------------------------------------

  xy = gsn_xy(wks,xo,yo4,xyres)      ; Plot the approximated curve created
                                     ; using 4 knots.

  xyres@xyMarkLineMode = "Markers"   ; Set line mode to "Markers".
  xyres@xyMarkerColor  = 3           ; Set markers to blue.
  xyres@xyMarkerSizeF  = 0.025       ; Make markers larger.
```

```
  xy = gsn_xy(wks,xi,yi,xyres)       ; Mark the original points with blue Xs.

  txres@txFont        = 21           ; Change the default font.
  txres@txFontHeightF = 0.03         ; Set the font height.
  gsn_text(wks,xy,"knots = 4",0.25,1.0,txres) ; Label the plot.

  gsres@gsLineColor = 2              ; Set polyline color to red.
  gsn_polyline(wks,xy,xx,yy,gsres)   ; Draw polyline at Y=0.

;----------- Second graph, same picture --------------------------------

  xyres@vpYF              = 0.58      ; Set Y location of plot.
  xyres@xyMarkLineMode   = "Lines"   ; Set line mode to "Lines".

  xy = gsn_xy(wks,xo,yo7,xyres)      ; Plot the approximated curve
                                     ; created using 7 knots.

  xyres@xyMarkLineMode   = "Markers" ; Set line mode to "Markers".
  xy = gsn_xy(wks,xi,yi,xyres)       ; Mark original points.

  txres@txFontHeightF = 0.03         ; Set font height.
  gsn_text(wks,xy,"knots = 7",0.25,1.0,txres) ; Label the plot.

  gsn_polyline(wks,xy,xx,yy,gsres)   ; Draw polyline at Y=0.


;----------- Third graph, same picture --------------------------------

  xyres@vpYF              = 0.28      ; Set Y location of plot.
  xyres@xyMarkLineMode   = "Lines"   ; Set line mode.

  xy = gsn_xy(wks,xo,yo9,xyres)      ; Plot the approximated curve created
                                     ; using 9 knots.

  xyres@xyMarkLineMode   = "Markers" ; Set line mode.
  xy = gsn_xy(wks,xi,yi,xyres)       ; Plot the original points with markers.

  txres@txFontHeightF = 0.03         ; Set font height.
  gsn_text(wks,xy,"knots = 9",0.25,1.0,txres) ; Label the plot.

  gsn_polyline(wks,xy,xx,yy,gsres)   ; Draw polyline at Y=0.

;------------- Main title -------------------------------------------

  txres@txFontHeightF = 0.04         ; Change the font height.
  gsn_text_ndc(wks,"Demo for csa1s",.5,.95,txres)

;------------- Plot and clean up  -----------------------------------
  frame(wks)

  delete(xy)
  delete(txres)
  delete(gsres)
  delete(xyres)

end
```

Plot for example 1 - one-dimensional example varying the number of knots



Example 2 - one-dimensional example illustrating input data weighting (C)

```
#include <stdio.h>
#include <ncarg/ncargC.h>
#include <ncarg/gks.h>
#include <ncarg/ngmath.h>

/*
 *  Function prototypes for plotting backgrounds and curves.
 */
void c_bkgft1(float, char *, float, float);
void c_drwft1(int, float [], float [], int, float [], float [],
```

```
              float [], float []);

/*
 *  The number of input data points.
 */
#define NDATA   6

/*
 *  The number of output data points.
 */
#define NPTS  101

/*
 *  The GKS workstation type (NCGM).
 */
#define IWTYPE  1

/*
 *  The GKS workstaton identifier.
 */
#define WKID    1

/*
 *  Demo of the effect of weighting the input points using c_csa1xs.
 */
main ()
{
/*
 *  Set up the output arrays.
 */
  float xo[NPTS],*yo1,*yo2,*yo3,xinc;

/*
 *  Specify the input data and initial weighting array.
 */
  float  xi[] = {0.0, 0.2,  0.4,  0.6, 0.8,  1.0};
  float  yi[] = {0.0, 1.0, -0.7, -0.2, -0.1, 0.0};
  float wts[] = {1.0, 1.0,  1.0,  1.0,  1.0, 1.0};

  float smth=0.;
  int i,knots=5,nderiv=0,ier;

/*
 *  Specify the output coordinates.
 */
  xinc = 1./ (float) (NPTS-1);
  for (i = 0; i < NPTS; i++) {
    xo[i] = (float) i * xinc;
  }

/*
 *  Calculate the approximating curve with all weights equal.
 */
  yo1 = c_csa1xs(NDATA,xi,yi,wts,knots,smth,nderiv,NPTS,xo,&ier);
  if (ier != 0) {
    printf("Error return from c_csa1xs: %d\n",ier);
    exit(1);
  }

/*
```

```
 *  Calculate the approximating curve with the second input coordinate
 *  weighted as half the other weights.
 */
  wts[1] = 0.5;
  yo2 = c_csa1xs(NDATA,xi,yi,wts,knots,smth,nderiv,NPTS,xo,&ier);
  if (ier != 0) {
    printf("Error return from c_csa1xs: %d\n",ier);
    exit(1);
  }

/*
 *  Calculate the approximating curve with the second input coordinate
 *  ignored (i.e. with a weight of zero).
 */
  wts[1] = 0.0;
  yo3 = c_csa1xs(NDATA,xi,yi,wts,knots,smth,nderiv,NPTS,xo,&ier);
  if (ier != 0) {
    printf("Error return from c_csa1xs: %d\n",ier);
    exit(1);
  }

/*
 *  Draw plot.
 */
  c_drwft1(NDATA,xi,yi,NPTS,xo,yo1,yo2,yo3);

}

void c_drwft1(int n, float x[], float y[], int m, float xo[],
              float curve1[], float curve2[], float curve3[])
{

/*
 *  This function uses NCAR Graphics to plot curves.
 */

  int   i;
  float yb, yt, ypos_top = 0.88;

/*
 *  Declare variables used in GKS calls.
 */
  Gcolr_rep rgb;
  Gpoint plist[NDATA];
  Gpoint_list pmk;

/*
 *  Open GKS, open and activate a workstation.
 */
  gopen_gks("stdout",0);
  gopen_ws(WKID, NULL, IWTYPE);
  gactivate_ws(WKID);

/*
 * Define a color table.
 */
  rgb.rgb.red = rgb.rgb.green = rgb.rgb.blue = 1.;
  gset_colr_rep(WKID,0,&rgb);
  rgb.rgb.red = rgb.rgb.green = rgb.rgb.blue = 0.;
  gset_colr_rep(WKID,1,&rgb);
  rgb.rgb.red = 1.;
```

```
  rgb.rgb.green = rgb.rgb.blue = 0.;
  gset_colr_rep(WKID,2,&rgb);
  rgb.rgb.red = rgb.rgb.green = 0.;
  rgb.rgb.blue = 1.;
  gset_colr_rep(WKID,3,&rgb);

/*
 * Plot the main title.
 */
  gset_clip_ind(0);
  c_plchhq(.5,.95,":F21:Effect of data weights",0.035,0.,0.);

/*
 *  Draw a background grid for the first curve.
 */
  yb = -1.0;
  yt =  1.0;
  c_bkgft1(ypos_top,"Weights = (1., 1., 1., 1., 1., 1.)",yb,yt);
  c_gridal(5,5,4,1,1,1,10,0.0,yb);

/*
 *  Graph the approximation curve.
 */
  c_curve(xo,curve1,m);

/*
 *  Mark the input data points.
 */
  for (i = 0; i < n; i++) {
    plist[i].x = x[i];
    plist[i].y = y[i];
  }
  gset_marker_size(2.2);
  gset_marker_colr_ind(3);
  pmk.num_points = n;
  pmk.points = plist;
  gpolymarker(&pmk);

/*
 *  Graph the approximation curve where the second input coordinate was
 *  given a weight of 0.5.
 */
  c_bkgft1(ypos_top-0.3,"Weights = (1., .5, 1., 1., 1., 1.)",yb,yt);
  c_gridal(5,5,4,1,1,1,10,0.0,yb);
  c_curve(xo,curve2,m);
  gpolymarker(&pmk);

/*
 *  Graph the approximation curve where the second input coordinate was
 *  given a weight of 0.0 .
 */
  c_bkgft1(ypos_top-0.6,"Weights = (1., 0., 1., 1., 1., 1.)",yb,yt);
  c_gridal(5,5,4,1,1,1,10,0.0,yb);
  c_curve(xo,curve3,m);
  gpolymarker(&pmk);

  c_frame();

/*
 *  Deactivate and close workstation, close GKS.
 */
```

```
  gdeactivate_ws(WKID);
  gclose_ws(WKID);
  gclose_gks();
}

/*
 *  Draw a background.
 */
void c_bkgft1(float ypos, char *label, float yb, float yt) {
  c_set(0.,1.,0.,1.,0.,1.,0.,1.,1);

/*
 *   Plot the curve label using font 21 (Helvetica).
 */
  c_pcseti("fn",21);
  c_plchhq(.65,ypos-0.03,label,0.025,0.,0.0);

/*
 *  Draw a horizontal line at Y=0. using color index 2.
 */
  c_set(0.13,0.93,ypos-0.2,ypos,0.0,1., yb, yt, 1);
  gset_line_colr_ind(2);
  c_line(0.,0.,1.,0.);
  c_sflush();
  gset_line_colr_ind(1);

/*
 *  Set Gridal parameters.
 *
 *
 *   Set lty to indicate that the Plotchar routine PLCHHQ should be used.
 */
  c_gaseti("lty",1);

/*
 *   Size and format for X axis labels.
 */
  c_gasetr("xls",0.02);
  c_gasetc("xlf","(f3.1)");

/*
 *   Size and format for X axis labels.
 */
  c_gasetr("yls",0.02);
  c_gasetc("ylf","(f5.1)");

/*
 *  Length of major tick marks for the X and Y axes.
 */
  c_gasetr("xmj",0.02);
  c_gasetr("ymj",0.02);
}
```

Example 2 - one-dimensional example illustrating input data weighting (Fortran)

```
      PROGRAM CSEX02
C
C  Show the effects of changing the weighting array.
C
C  The number of input data points.
C
      PARAMETER (NDATA=6)
C
C  The number of output data points.
C
      PARAMETER (NX=101)
C
C  The maximum number of knots used in any call.
C
      PARAMETER (NCF=5)
C
C  The size of the workspace.
C
      PARAMETER (NWRK=NCF*(NCF+3))
C
C  Array dimensions.
C
      DIMENSION XDATA(1,NDATA),YDATA(NDATA),XDATAT(NDATA)
      DIMENSION WTS(NDATA),WORK(NWRK)
      DIMENSION XO(NX),Y1(NX),Y2(NX),Y3(NX)
C
C  Initialize the weight array so that all input points are
C  weighted equally.
C
      DATA WTS/1.0, 1.0, 1.0, 1.0, 1.0, 1.0/
C
C  Specify the input data.
C
      DATA XDATA(1,1),XDATA(1,2),XDATA(1,3)/0.0, 0.2, 0.4/
      DATA XDATA(1,4),XDATA(1,5),XDATA(1,6)/0.6, 0.8, 1.0/
      DATA YDATA/0.0, 1.0, -0.7, -0.2, -0.1, 0.0/
C
C  Define the output coordinates.
C
      XINC = 1./REAL(NX-1)
      DO 10 I=1,NX
        XO(I) = (I-1)*XINC
   10 CONTINUE
C
C  Calculate the approximating curve with all weights equal.
C
      SSMTH = 0.
      NDERIV = 0
      CALL CSA1XS (NDATA,XDATA,YDATA,WTS,NCF,
     +             SSMTH,NDERIV,NX,XO,Y1,NWRK,WORK,IER)
      IF (IER .NE. 0) THEN
        WRITE(6,520) IER
  520   FORMAT(' Error ',I3,' returned from CSA1XS')
        STOP
      ENDIF
C
C  Calculate the approximating curve with the second input coordinate
C  weighted as half the other weights.
```

```
C
      WTS(2) = 0.5
      CALL CSA1XS (NDATA,XDATA,YDATA,WTS,NCF,
     +             SSMTH,NDERIV,NX,XO,Y2,NWRK,WORK,IER)
      IF (IER .NE. 0) THEN
        WRITE(6,520) IER
        STOP
      ENDIF
C
C  Calculate the approximating curve with the second input coordinate
C  ignored (i.e. with a weight of zero).
C
      WTS(2) = 0.0
      CALL CSA1XS (NDATA,XDATA,YDATA,WTS,NCF,
     +             SSMTH,NDERIV,NX,XO,Y3,NWRK,WORK,IER)
      IF (IER .NE. 0) THEN
        WRITE(6,520) IER
        STOP
      ENDIF
C
C  Draw a plot of the approximation curves and mark the original points.
C
      DO 30 I=1,NDATA
        XDATAT(I) = XDATA(1,I)
   30 CONTINUE
      CALL DRWFT1(NDATA,XDATAT,YDATA,NX,XO,Y1,Y2,Y3)
C
      STOP
      END
      SUBROUTINE DRWFT1(NUMIN,X,Y,NUMOUT,XO,Y1,Y2,Y3)
C
C  This subroutine uses NCAR Graphics to plot curves.
C
C  Define error file, Fortran unit number, workstation type,
C  and workstation ID.
C
      PARAMETER (IERRF=6, LUNIT=2, IWTYPE=1, IWKID=1)
C
C  Vertical position for initial curve.
C
      DATA YPOS_TOP/0.88/
C
C  Open GKS, open and activate a workstation.
C
      CALL GOPKS (IERRF, ISZDM)
      CALL GOPWK (IWKID, LUNIT, IWTYPE)
      CALL GACWK (IWKID)
C
C  Define a color table.
C
      CALL GSCR(IWKID, 0, 1.0, 1.0, 1.0)
      CALL GSCR(IWKID, 1, 0.0, 0.0, 0.0)
      CALL GSCR(IWKID, 2, 1.0, 0.0, 0.0)
      CALL GSCR(IWKID, 3, 0.0, 1.0, 0.0)
      CALL GSCR(IWKID, 4, 0.0, 0.0, 1.0)
      CALL GSCLIP(0)
C
C  Plot the main title.
C
      CALL PLCHHQ(0.50,0.95,':F25:Effect of data weights',
     +            0.030,0.,0.)
```

```
C
C  Draw a background grid for the first curve.
C
      YB = -1.0
      YT =  1.0
      CALL BKGFT1(YPOS_TOP,'Weights = (1., 1., 1., 1., 1., 1.)',YB,YT)
      CALL GRIDAL(5,5,4,1,1,1,10,0.0,YB)
C
C  Graph the approximation curve where all input coordinates were
C  wieghted equally.
C
      CALL GPL(NUMOUT,XO,Y1)
C
C  Mark the original data points.
C
      CALL GSMKSC(2.)
      CALL GSPMCI(4)
      CALL GPM(NUMIN,X,Y)
C
C  Graph the approximation curve where the second input coordinate was
C  given a weight of 0.5.
C
      CALL BKGFT1(YPOS_TOP-0.3,
     +            'Weights = (1., .5, 1., 1., 1., 1.)',YB,YT)
      CALL GRIDAL(5,5,4,1,1,1,10,0.0,YB)
      CALL GPL(NUMOUT,XO,Y2)
      CALL GPM(NUMIN,X,Y)
C
C  Graph the approximation curve where the second input coordinate was
C  given a weight of 0.0 .
C
      CALL BKGFT1(YPOS_TOP-0.6,
     +            'Weights = (1., 0., 1., 1., 1., 1.)',YB,YT)
      CALL GRIDAL(5,5,4,1,1,1,10,0.0,YB)
      CALL GPL(NUMOUT,XO,Y3)
      CALL GPM(NUMIN,X,Y)
      CALL FRAME
C
      CALL GDAWK(IWKID)
      CALL GCLWK(IWKID)
      CALL GCLKS
C
      RETURN
      END
      SUBROUTINE BKGFT1(YPOS,LABEL,YB,YT)
C
C  This subroutine draws a background grid.
C
      DIMENSION XX(2),YY(2)
      CHARACTER*(*) LABEL
C
      CALL SET(0.,1.,0.,1.,0.,1.,0.,1.,1)
C
C  Plot the curve label using font 21 (Helvetica).
C
      CALL PCSETI('FN',21)
      CALL PLCHHQ(0.65,YPOS - 0.03,LABEL,0.023,0.,0.)
      CALL SET(0.13,0.93,YPOS-0.2,YPOS,0.0,1., YB, YT, 1)
C
C  Draw a horizontal line at Y=0. using color index 2.
C
```

```
      XX(1) = 0.
      XX(2) = 1.
      YY(1) = 0.
      YY(2) = 0.
      CALL GSPLCI(2)
      CALL GPL(2,XX,YY)
      CALL GSPLCI(1)
C
C  Set Gridal parameters.
C
C
C   Set LTY to indicate that the Plotchar routine PLCHHQ should be used.
C
      CALL GASETI('LTY',1)
C
C   Size and format for X axis labels.
C
      CALL GASETR('XLS',0.02)
      CALL GASETC('XLF','(F3.1)')
C
C   Size and format for Y axis labels.
C
      CALL GASETR('YLS',0.02)
      CALL GASETC('YLF','(F5.1)')
C
C   Length of major tick marks for the X and Y axes.
C
      CALL GASETR('XMJ',0.02)
      CALL GASETR('YMJ',0.02)
C
      RETURN
      END
```

Example 2 - one-dimensional example illustrating input data weighting (NCL)

```
load "$NCARG_ROOT/lib/ncarg/nclex/gsun/gsn_code.ncl"

begin

;----------- Define original data and initial weighting array -----------

  xi =  (/0.0, 0.2,  0.4,  0.6, 0.8,  1.0/)
  yi =  (/0.0, 1.0, -0.7, -0.2, -0.1, 0.0/)
  wts = (/1.0, 1.0,  1.0,  1.0,  1.0, 1.0/)

  npts   = 101
  xo     = fspan(0.0,1.0,npts)  ; Create the output X coordinate array.

  xx = (/0.0, 1.0/) ; Create data for a polyline for marking
  yy = (/0.0, 0.0/) ; the Y = 0.0 line in each graph.

;--- Calculate approximated function values using different weights.
```

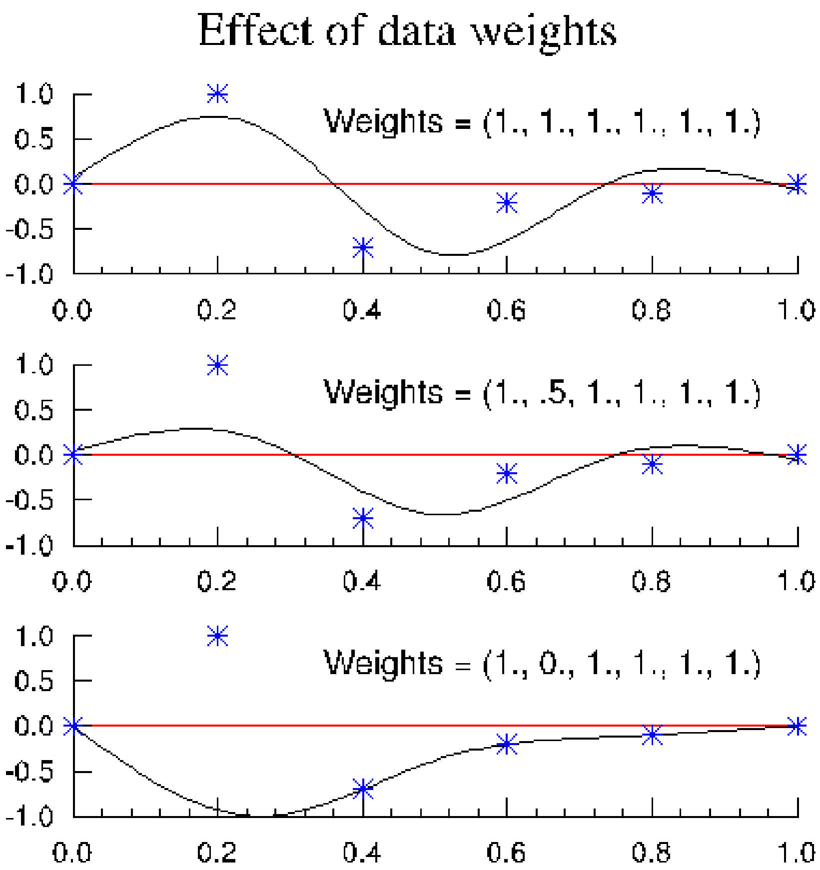
```
  yo1 = csa1xs(xi,yi,wts,5,0.0,0,xo)
  wts(1) = 0.5
  yo2 = csa1xs(xi,yi,wts,5,0.0,0,xo)
  wts(1) = 0.0
  yo3 = csa1xs(xi,yi,wts,5,0.0,0,xo)

;---- Open workstation, define color map ---------------------------

  NCGM=1
  X11=0
  PS=0

  if (NCGM .eq. 1) then
    wks = gsn_open_wks("ncgm","nm14n")
  end if
  if (X11 .eq. 1) then
    wks = gsn_open_wks("x11","nm14n")
  end if
  if (PS .eq. 1) then
    wks = gsn_open_wks("ps","nm14n")
  end if

  cmap = (/                  \
           (/1.,1.,1./),     \
           (/0.,0.,0./),     \
           (/1.,0.,0./),     \
           (/0.,0.,1./)      \
         /)
  gsn_define_colormap(wks,cmap)

;---- Specify resources --------------------------

  txres = True   ; Set up some variables for
  xyres = True   ; creating three different
  gsres = True   ; resource lists.

  xyres@gsnFrame                = False     ; Don't advance the frame.

  xyres@tmXTBorderOn            = False     ; Don't draw top axis.
  xyres@tmXTOn                  = False     ; Don't draw top axis tick marks.
  xyres@tmBorderThicknessF      = 1.0       ; Default thickness is 2.0

  xyres@tmXBLabelFont           = 21        ; Change X label font.
  xyres@tmXBLabelFontHeightF    = 0.025     ; Change font size.
  xyres@tmXBMajorLengthF        = 0.015     ; Default is 0.02.
  xyres@tmXBMajorThicknessF     = 1.0       ; Default is 2.
  xyres@tmXBMinorLengthF        = 0.0075    ; Default is 0.01.
  xyres@tmXBMinorPerMajor       = 4         ; # of minor tick marks per major.
  xyres@tmXBMode                = "Manual"  ; Set tick mark mode.
  xyres@tmXBTickStartF          = 0.0
  xyres@tmXBTickEndF            = 1.0
  xyres@tmXBTickSpacingF        = 0.2
  xyres@tmXMajorGridThicknessF = 1.0        ; Default is 2.0

  xyres@tmYLLabelFont           = 21        ; See explanations for X axes
  xyres@tmYLLabelFontHeightF    = 0.025     ; resources.
  xyres@tmYLMajorLengthF        = 0.015
  xyres@tmYLMajorThicknessF     = 1.0
  xyres@tmYLMinorPerMajor       = 0
  xyres@tmYLMode                = "Manual"
```

```
  xyres@tmYLTickStartF          = -1.0
  xyres@tmYLTickEndF            =  1.0
  xyres@tmYLTickSpacingF        =  0.5
  xyres@tmYRBorderOn            = False     ; Don't draw right axis.
  xyres@tmYROn                  = False     ; Don't draw right axis tick marks.

  xyres@trXMinF   =  0.0
  xyres@trXMaxF   =  1.0                    ; Specify data limits.
  xyres@trYMaxF   =  1.0
  xyres@trYMinF   = -1.0

  xyres@vpHeightF = 0.20                    ; Specify size and location of plot.
  xyres@vpWidthF  = 0.80
  xyres@vpXF      = 0.13
  xyres@vpYF      = 0.88

;----------- First graph --------------------------------------------

  xy = gsn_xy(wks,xo,yo1,xyres)      ; Plot the approximated curve created
                                     ; with equal weights.

  xyres@xyMarkLineMode = "Markers"   ; Set line mode to "Markers".
  xyres@xyMarkerColor  = 3           ; Set markers to blue.
  xyres@xyMarkerSizeF  = 0.025       ; Make markers larger.

  xy = gsn_xy(wks,xi,yi,xyres)       ; Mark the original points with blue Xs.

  txres@txFont        = 21           ; Change the default font.
  txres@txFontHeightF = 0.03         ; Set the font height.
  gsn_text(wks,xy,"Weights = (1., 1., 1., 1., 1., 1.)",0.65,0.7,txres)

  gsres@gsLineColor = 2              ; Set polyline color to red.
  gsn_polyline(wks,xy,xx,yy,gsres)   ; Draw polyline at Y=0.

;----------- Second graph, same picture --------------------------------

  xyres@vpYF             = 0.58      ; Set Y location of plot.
  xyres@xyMarkLineMode   = "Lines"   ; Set line mode to "Lines".

  xy = gsn_xy(wks,xo,yo2,xyres)      ; Plot the approximated curve
                                     ; created using a weight of 0.5
                                     ; to the second input point.

  xyres@xyMarkLineMode   = "Markers" ; Set line mode to "Markers".
  xy = gsn_xy(wks,xi,yi,xyres)       ; Mark original points.

  txres@txFontHeightF = 0.03         ; Set font height.
  gsn_text(wks,xy,"Weights = (1., .5, 1., 1., 1., 1.)",0.65,0.7,txres)

  gsn_polyline(wks,xy,xx,yy,gsres)   ; Draw polyline at Y=0.


;----------- Third graph, same picture --------------------------------

  xyres@vpYF             = 0.28      ; Set Y location of plot.
  xyres@xyMarkLineMode   = "Lines"   ; Set line mode.

  xy = gsn_xy(wks,xo,yo3,xyres)      ; Plot the approximated curve created
                                     ; using a zero weight for the second
                                     ; input point.
```

```
  xyres@xyMarkLineMode   = "Markers" ; Set line mode.
  xy = gsn_xy(wks,xi,yi,xyres)       ; Plot the original points with markers.

  txres@txFontHeightF = 0.03         ; Set font height.
  gsn_text(wks,xy,"Weights = (1., 0., 1., 1., 1., 1.)",0.65,0.7,txres)

  gsn_polyline(wks,xy,xx,yy,gsres)   ; Draw polyline at Y=0.

;------------- Main title ------------------------------------------

  txres@txFontHeightF = 0.04         ; Change the font height.
  gsn_text_ndc(wks,":F21:Effect of data weights",.5,.95,txres)

;------------- Plot and clean up  -----------------------------------
  frame(wks)

  delete(xy)
  delete(txres)
  delete(gsres)
  delete(xyres)

end
```

Plot for example 2 - one-dimensional example illustrating input data weighting

---



Example 3 - one-dimensional example illustrating extrapolation into data sparse regions (C)

```
#include <stdio.h>
#include <ncarg/ncargC.h>
#include <ncarg/gks.h>
#include <ncarg/ngmath.h>

/*
 *  Function prototypes for plotting backgrounds and curves.
 */
void c_bkgft1(float, char *, float, float);
void c_drwft1(int, float [], float [], int, float [], float [],
              float []);

/*
 *  The number of input data points.
```

```
 */
#define NDATA   9

/*
 *  The number of output data points.
 */
#define NPTS  101

/*
 *  The GKS workstation type (NCGM).
 */
#define IWTYPE  1

/*
 *  The GKS workstaton identifier.
 */
#define WKID    1

/*
 *  Demo of extrapolation into data sparse regions using c_csa1xs.
 */
main ()
{

/*
 *  Set up the output arrays.
 */
  float xo[NPTS],*yo1,*yo2,xinc;

/*
 *  Specify the input data and initial weighting array.
 */
  float  xi[] = {0.0, 0.1, 0.2,  0.3,  0.4, 0.5, 0.6, 0.9, 1.0};
  float  yi[] = {0.0, 0.2, 0.4, -0.4, -1.0, 0.2, 0.5, 0.4, 0.0};

  float smth=0.,wts=-1.;
  int i,knots=8,nderiv=0,ier;

/*
 *  Specify the output coordinates.
 */
  xinc = 1./ (float) (NPTS-1);
  for (i = 0; i < NPTS; i++) {
    xo[i] = (float) i * xinc;
  }

/*
 *  Calculate the approximating curve with no extrapolation.
 *  Note the unexpected oscillation in the curve.
 */
  yo1 = c_csa1xs(NDATA,xi,yi,&wts,knots,smth,nderiv,NPTS,xo,&ier);
  if (ier != 0) {
    printf("Error return from c_csa1xs: %d\n",ier);
    exit(1);
  }

/*
 *  Calculate the approximating curve with extrapolation into
 *  data sparse regions.
 */
  smth = 1.;
```

```
  yo2 = c_csa1xs(NDATA,xi,yi,&wts,knots,smth,nderiv,NPTS,xo,&ier);
  if (ier != 0) {
    printf("Error return from c_csa1xs: %d\n",ier);
    exit(1);
  }

/*
 *  Draw plot.
 */
  c_drwft1(NDATA,xi,yi,NPTS,xo,yo1,yo2);

}

void c_drwft1(int n, float x[], float y[], int m, float xo[],
              float curve1[], float curve2[])
{

/*
 *  This function uses NCAR Graphics to plot curves.
 */

  int   i;
  float yb, yt, ypos_top = 0.86;

/*
 *  Declare variables used in GKS calls.
 */
  Gcolr_rep rgb;
  Gpoint plist[NDATA];
  Gpoint_list pmk;

/*
 *  Open GKS, open and activate a workstation.
 */
  gopen_gks("stdout",0);
  gopen_ws(WKID, NULL, IWTYPE);
  gactivate_ws(WKID);

/*
 * Define a color table.
 */
  rgb.rgb.red = rgb.rgb.green = rgb.rgb.blue = 1.;
  gset_colr_rep(WKID,0,&rgb);
  rgb.rgb.red = rgb.rgb.green = rgb.rgb.blue = 0.;
  gset_colr_rep(WKID,1,&rgb);
  rgb.rgb.red = 1.;
  rgb.rgb.green = rgb.rgb.blue = 0.;
  gset_colr_rep(WKID,2,&rgb);
  rgb.rgb.red = rgb.rgb.green = 0.;
  rgb.rgb.blue = 1.;
  gset_colr_rep(WKID,3,&rgb);

/*
 * Plot the main title.
 */
  gset_clip_ind(0);
  c_plchhq(.5,.95,":F21:Data sparse areas",0.03,0.,0.);

/*
 *  Draw a background grid for the first curve.
 */
```

```
  yb = -2.0;
  yt =  1.0;
  c_bkgft1(ypos_top,"smth = 0.",yb,yt);
  c_gridal(5,5,3,1,1,1,10,0.0,yb);

/*
 *  Graph the approximation curve.
 */
  c_curve(xo,curve1,m);

/*
 *  Mark the input data points.
 */
  for (i = 0; i < n; i++) {
    plist[i].x = x[i];
    plist[i].y = y[i];
  }
  gset_marker_size(2.2);
  gset_marker_colr_ind(3);
  pmk.num_points = n;
  pmk.points = plist;
  gpolymarker(&pmk);

/*
 *  Graph the approximation curve.
 */
  c_bkgft1(ypos_top-0.45,"smth = 1.",yb,yt);
  c_gridal(5,5,3,1,1,1,10,0.0,yb);
  c_curve(xo,curve2,m);
  gpolymarker(&pmk);

  c_frame();

/*
 *  Deactivate and close workstation, close GKS.
 */
  gdeactivate_ws(WKID);
  gclose_ws(WKID);
  gclose_gks();
}

/*
 *  Draw a background.
 */
void c_bkgft1(float ypos, char *label, float yb, float yt) {

  c_set(0.,1.,0.,1.,0.,1.,0.,1.,1);

/*
 *   Plot the curve label using font 21 (Helvetica).
 */
  c_pcseti("fn",21);
  c_plchhq(0.17,ypos-0.2,label,0.03,0.,-1.0);

/*
 *  Draw a horizontal line at Y=0. using color index 2.
 */
  c_set(0.13,0.93,ypos-0.28,ypos,0.0,1., yb, yt, 1);
  gset_line_colr_ind(2);
  c_line(0.,0.,1.,0.);
  c_sflush();
```

```
  gset_line_colr_ind(1);

/*
 *  Set Gridal parameters.
 *
 *
 *   Set lty to indicate that the Plotchar routine PLCHHQ should be used.
 */
  c_gaseti("lty",1);
  c_pcseti("fn",21);

/*
 *   Size and format for X axis labels.
 */
  c_gasetr("xls",0.02);
  c_gasetc("xlf","(f3.1)");

/*
 *   Size and format for X axis labels.
 */
  c_gasetr("yls",0.02);
  c_gasetc("ylf","(f5.1)");

/*
 *  Length of major tick marks for the X and Y axes.
 */
  c_gasetr("xmj",0.02);
  c_gasetr("ymj",0.02);
}
```

Example 3 - one-dimensional example illustrating extrapolation into data sparse regions (Fortran)

```
      PROGRAM CSEX03
C
C  Demo of extrapolation into data sparse regions using CSA1XS.
C
C  The number of input data points.
C
      PARAMETER (NDATA=9)
C
C  The number of output data points.
C
      PARAMETER (NX=101)
C
C  The maximum number of knots used in any call.
C
      PARAMETER (NCF=8)
C
```

```
C  The size of the workspace.
C
      PARAMETER (NWRK=NCF*(NCF+3))
C
C  Dimension the arrays.
C
      DIMENSION XDATA(1,NDATA),YDATA(NDATA),XDATAT(NDATA)
      DIMENSION WORK(NWRK)
      DIMENSION XO(NX),Y1(NX),Y2(NX)
C
C  Specify the input data.
C
      DATA XDATA(1,1),XDATA(1,2),XDATA(1,3)/0.0, 0.1, 0.2/
      DATA XDATA(1,4),XDATA(1,5),XDATA(1,6)/0.3, 0.4, 0.5/
      DATA XDATA(1,7),XDATA(1,8),XDATA(1,9)/0.6, 0.90, 1.0/
      DATA YDATA/0.0, 0.2, 0.4, -0.4, -1.0, 0.2, 0.5, 0.4, 0.0/
C
C  Define the output coordinates.
C
      XINC = 1./REAL(NX-1)
      DO 10 I=1,NX
        XO(I) = (I-1)*XINC
   10 CONTINUE
C
C  Calculate the approximating curve with no extrapolation.  Note
C  the unexpected oscillation in the approximation curve.
C
      SSMTH = 0.
      NDERIV = 0
      WTS = -1.
      CALL CSA1XS (NDATA,XDATA,YDATA,WTS,NCF,
     +             SSMTH,NDERIV,NX,XO,Y1,NWRK,WORK,IER)
      IF (IER .NE. 0) THEN
        WRITE(6,520) IER
  520   FORMAT(' Error ',I3,' returned from CSA1XS')
        STOP
      ENDIF
C
C  Calculate the approximating curve with the extrapolation parameter
C  set to 1.0.
C
      SSMTH = 1.0
      CALL CSA1XS (NDATA,XDATA,YDATA,WTS,NCF,
     +             SSMTH,NDERIV,NX,XO,Y2,NWRK,WORK,IER)
      IF (IER .NE. 0) THEN
        WRITE(6,520) IER
        STOP
      ENDIF
C
C  Draw a plot of the approximation curves and mark the original points.
C
      DO 30 I=1,NDATA
        XDATAT(I) = XDATA(1,I)
   30 CONTINUE
      CALL DRWFT1(NDATA,XDATAT,YDATA,NX,XO,Y1,Y2)
C
      STOP
      END
      SUBROUTINE DRWFT1(NUMIN,X,Y,NUMOUT,XO,Y1,Y2)
C
C  This subroutine uses NCAR Graphics to plot curves.
```

```
C
C  Define error file, Fortran unit number, workstation type,
C  and workstation ID.
C
      PARAMETER (IERRF=6, LUNIT=2, IWTYPE=1, IWKID=1)
C
C  Vertical position for initial curve.
C
      DATA YPOS_TOP/0.86/
C
C  Open GKS, open and activate a workstation.
C
      CALL GOPKS (IERRF, ISZDM)
      CALL GOPWK (IWKID, LUNIT, IWTYPE)
      CALL GACWK (IWKID)
C
C  Define a color table.
C
      CALL GSCR(IWKID, 0, 1.0, 1.0, 1.0)
      CALL GSCR(IWKID, 1, 0.0, 0.0, 0.0)
      CALL GSCR(IWKID, 2, 1.0, 0.0, 0.0)
      CALL GSCR(IWKID, 3, 0.0, 1.0, 0.0)
      CALL GSCR(IWKID, 4, 0.0, 0.0, 1.0)
      CALL GSCLIP(0)
C
C  Plot the main title.
C
      CALL PLCHHQ(0.50,0.95,':F25:Data sparse areas',
     +            0.030,0.,0.)
C
C  Graph the approximation curve with extrapolation into data sparse
C  regions turned off.  Note the unexpected oscillation in the curve.
C
C
C  Draw a background grid for the first curve.
C
      YB = -2.0
      YT =  1.0
      CALL BKGFT1(YPOS_TOP,'SMTH = 0.',YB,YT)
      CALL GRIDAL(5,5,5,1,1,1,10,0.0,YB)
C
C  Graph the approximation curve with no extrapolation into data
C  sparse areas.
C
      CALL GPL(NUMOUT,XO,Y1)
C
C  Mark the original data points.
C
      CALL GSMKSC(2.)
      CALL GSPMCI(4)
      CALL GPM(NUMIN,X,Y)
C
C  Graph the approximation curve with extrapolation parameter set to 1.
C
      CALL BKGFT1(YPOS_TOP-0.45,'SMTH = 1.',YB,YT)
      CALL GRIDAL(5,5,5,1,1,1,10,0.0,YB)
      CALL GPL(NUMOUT,XO,Y2)
      CALL GPM(NUMIN,X,Y)
      CALL FRAME
C
      CALL GDAWK(IWKID)
```

```
      CALL GCLWK(IWKID)
      CALL GCLKS
C
      RETURN
      END
      SUBROUTINE BKGFT1(YPOS,LABEL,YB,YT)
C
C  This subroutine draws a background grid.
C
      DIMENSION XX(2),YY(2)
      CHARACTER*(*) LABEL
C
      CALL SET(0.,1.,0.,1.,0.,1.,0.,1.,1)
C
C  Plot the curve label using font 21 (Helvetica).
C
      CALL PCSETI('FN',21)
      CALL PLCHHQ(0.20,YPOS - 0.2,LABEL,0.023,0.,-1.)
      CALL SET(0.13,0.93,YPOS-0.28,YPOS,0.0,1., YB, YT, 1)
C
C  Draw a horizontal line at Y=0. using color index 2.
C
      XX(1) = 0.
      XX(2) = 1.
      YY(1) = 0.
      YY(2) = 0.
      CALL GSPLCI(2)
      CALL GPL(2,XX,YY)
      CALL GSPLCI(1)
C
C  Set Gridal parameters.
C
C
C   Set LTY to indicate that the Plotchar routine PLCHHQ should be used.
C
      CALL GASETI('LTY',1)
C
C   Size and format for X axis labels.
C
      CALL GASETR('XLS',0.02)
      CALL GASETC('XLF','(F3.1)')
C
C   Size and format for Y axis labels.
C
      CALL GASETR('YLS',0.02)
      CALL GASETC('YLF','(F5.1)')
C
C   Length of major tick marks for the X and Y axes.
C
      CALL GASETR('XMJ',0.02)
      CALL GASETR('YMJ',0.02)
C
      RETURN
      END
```

Example 3 - one-dimensional example

illustrating extrapolation into data sparse regions (NCL)

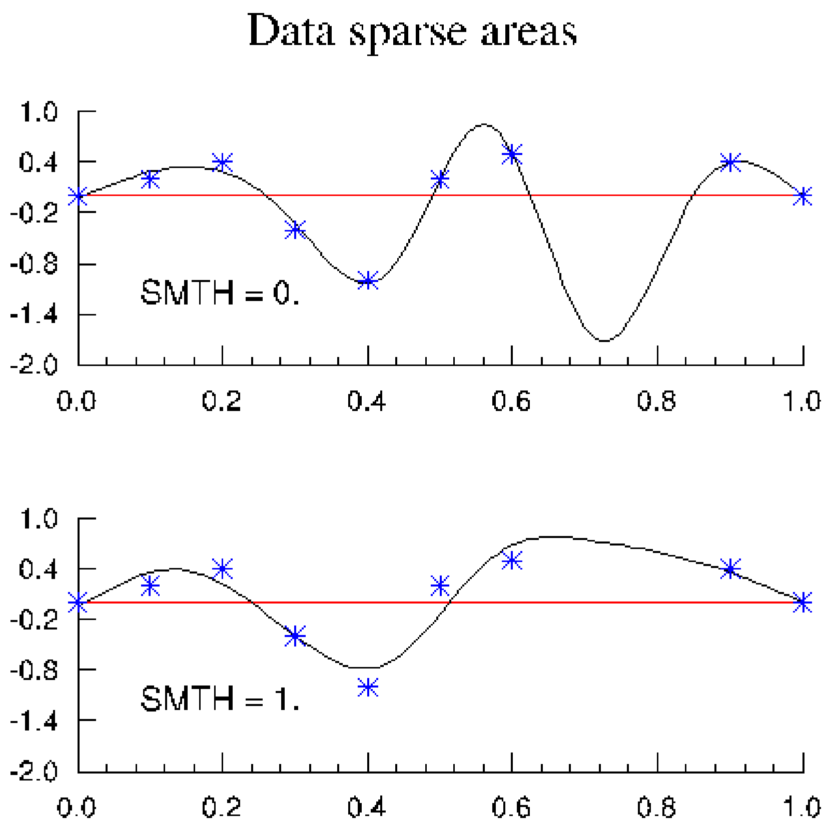
```
load "$NCARG_ROOT/lib/ncarg/nclex/gsun/gsn_code.ncl"

begin

;----------- Define original data and initial weighting array -----------

  xi = (/0.0, 0.1, 0.2,  0.3,  0.4, 0.5, 0.6, 0.9, 1.0/)
  yi = (/0.0, 0.2, 0.4, -0.4, -1.0, 0.2, 0.5, 0.4, 0.0/)

  npts   = 101
  xo     = fspan(0.0,1.0,npts)  ; Create the output X coordinate array.

  xx = (/0.0, 1.0/) ; Create data for a polyline for marking
  yy = (/0.0, 0.0/) ; the Y = 0.0 line in each graph.

;--- Calculate approximated function values using different weights.

  smth = 0.0
  yo1 = csa1xs(xi,yi,-1.,8,smth,0,xo)
  smth = 1.0
  yo2 = csa1xs(xi,yi,-1.,8,smth,0,xo)

;---- Open workstation, define color map ---------------------------

  NCGM=1
  X11=0
  PS=0

  if (NCGM .eq. 1) then
    wks = gsn_open_wks("ncgm","nm15n")
  end if
  if (X11 .eq. 1) then
    wks = gsn_open_wks("x11","nm15n")
  end if
  if (PS .eq. 1) then
    wks = gsn_open_wks("ps","nm15n")
  end if

  cmap = (/                 \
            (/1.,1.,1./),   \
            (/0.,0.,0./),   \
            (/1.,0.,0./),   \
            (/0.,0.,1./)    \
         /)
  gsn_define_colormap(wks,cmap)

;---- Specify resources ---------------------------

  txres = True   ; Set up some variables for
  xyres = True   ; creating three different
  gsres = True   ; resource lists.
```

```
  xyres@gsnFrame                = False     ; Don't advance the frame.

  xyres@tmXTBorderOn            = False     ; Don't draw top axis.
  xyres@tmXTOn                  = False     ; Don't draw top axis tick marks.
  xyres@tmBorderThicknessF      = 1.0       ; Default thickness is 2.0

  xyres@tmXBLabelFont           = 21        ; Change X label font.
  xyres@tmXBLabelFontHeightF    = 0.025     ; Change font size.
  xyres@tmXBMajorLengthF        = 0.015     ; Default is 0.02.
  xyres@tmXBMajorThicknessF     = 1.0       ; Default is 2.
  xyres@tmXBMinorLengthF        = 0.0075    ; Default is 0.01.
  xyres@tmXBMinorPerMajor       = 4         ; # of minor tick marks per major.
  xyres@tmXBMode                = "Manual"  ; Set tick mark mode.
  xyres@tmXBTickStartF          = 0.0
  xyres@tmXBTickEndF            = 1.0
  xyres@tmXBTickSpacingF        = 0.2
  xyres@tmXMajorGridThicknessF = 1.0       ; Default is 2.0

  xyres@tmYLLabelFont           = 21        ; See explanations for X axes
  xyres@tmYLLabelFontHeightF    = 0.025     ; resources.
  xyres@tmYLMajorLengthF        = 0.015
  xyres@tmYLMajorThicknessF     = 1.0
  xyres@tmYLMinorPerMajor       = 0
  xyres@tmYLMode                = "Manual"
  xyres@tmYLTickStartF          = -2.0
  xyres@tmYLTickEndF            =  1.0
  xyres@tmYLTickSpacingF        =  1.0
  xyres@tmYRBorderOn            = False     ; Don't draw right axis.
  xyres@tmYROn                  = False     ; Don't draw right axis tick marks.

  xyres@trXMinF   =  0.0
  xyres@trXMaxF   =  1.0                    ; Specify data limits.
  xyres@trYMinF   = -2.0
  xyres@trYMaxF   =  1.0

  xyres@vpHeightF = 0.28                    ; Specify size and location of plot.
  xyres@vpWidthF  = 0.80
  xyres@vpXF      = 0.13
  xyres@vpYF      = 0.86

;---------- First graph -------------------------------------------

  xy = gsn_xy(wks,xo,yo1,xyres)       ; Plot the approximated curve created
                                      ; with equal weights.

  xyres@xyMarkLineMode = "Markers"    ; Set line mode to "Markers".
  xyres@xyMarkerColor  = 3            ; Set markers to blue.
  xyres@xyMarkerSizeF  = 0.025        ; Make markers larger.

  xy = gsn_xy(wks,xi,yi,xyres)        ; Mark the original points with blue Xs.

  txres@txFont        = 21            ; Change the default font.
  txres@txFontHeightF = 0.038         ; Set the font height.
  gsn_text(wks,xy,"smth = 0.",0.17,-1.20,txres)

  gsres@gsLineColor = 2               ; Set polyline color to red.
  gsn_polyline(wks,xy,xx,yy,gsres)    ; Draw polyline at Y=0.

;---------- Second graph, same picture --------------------------------

  xyres@vpYF            = 0.43      ; Set Y location of plot.
```

```
  xyres@xyMarkLineMode   = "Lines"   ; Set line mode to "Lines".

  xy = gsn_xy(wks,xo,yo2,xyres)      ; Plot the approximated curve
                                     ; created using a weight of 0.5
                                     ; to the second input point.

  xyres@xyMarkLineMode   = "Markers" ; Set line mode to "Markers".
  xy = gsn_xy(wks,xi,yi,xyres)       ; Mark original points.

  txres@txFontHeightF = 0.038        ; Set font height.
  gsn_text(wks,xy,"smth = 1.",0.17,-1.20,txres)

  gsn_polyline(wks,xy,xx,yy,gsres)   ; Draw polyline at Y=0.

;------------- Main title ------------------------------------------

  txres@txFontHeightF = 0.04         ; Change the font height.
  gsn_text_ndc(wks,":F21:Data sparse areas",.5,.95,txres)

;------------- Plot and clean up  -----------------------------------
  frame(wks)

  delete(xy)
  delete(txres)
  delete(gsres)
  delete(xyres)

end
```

Plot for example 3 - one-dimensional example illustrating extrapolation into data sparse regions

---



Example 4 - a two-dimensional example illustrating taking partial derivatives (C)

```
#include <stdio.h>
#include <stdlib.h>
#include <ncarg/ncargC.h>
#include <ncarg/gks.h>
#include <ncarg/ngmath.h>

/*
 *  The dimensionality of the problem.
 */
#define NDIM    2

/*
 *  The number of output data points in the X coordinate direction.
 */
#define NX      29

/*
 *  The number of output data points in the Y coordinate direction.
 */
```

```
#define NY     25

/*
 *  The number of input data points.
 */
#define NI    1000

/*
 *  The number of knots in the X direction.
 */
#define N1      4

/*
 *  The number of knots in the Y direction.
 */
#define N2      4

/*
 *  Data limits.
 */
#define XMIN   -1.0
#define XMAX    1.0
#define YMIN   -1.0
#define YMAX    1.0

/*
 *  The GKS workstation type (NCGM).
 */
#define IWTYPE 1

/*
 *  The GKS workstaton identifier.
 */
#define WKID   1

/*
 *  Function prototype for the random number generator.
 */
float dsrnd1();

/*
 *  Do a 2D approximation and find mixed second partial derivatives.
 */
main ()
{

  float xi[NI],yi[NI],zi[NI],xo[NX],yo[NY],*function,*derivs,wts[1];
  float ssmth=0.0,*zor,t1,t2;
  int   i,j,knots[2],nderiv[2]={1,1},ier;

/*
 *  Generate input data using the functiuon f(x,y) = y**2 - 0.5*y*x**2
 */
  for (i = 0; i < NI; i++) {
    xi[i] = XMIN+(XMAX-XMIN)*dsrnd1();
    yi[i] = YMIN+(YMAX-YMIN)*dsrnd1();
    zi[i] = yi[i]*yi[i] - 0.5*xi[i]*xi[i]*yi[i];
  }

/*
 *  Create the output grid.
```

```
 */
  for (i = 0; i < NX; i++) {
    xo[i] = XMIN+((float)i/(float)(NX-1))*(XMAX-XMIN);
  }
  for (j = 0; j < NY; j++) {
    yo[j] = YMIN+((float)j/(float)(NY-1))*(YMAX-YMIN);
  }

/*
 *  Specify the numbers of knots in each coordinate direction.
 */
  knots[0] = N1;
  knots[1] = N2;

/*
 *  Calculate the approximated functuion values.
 */
  function = c_csa2s(NI,xi,yi,zi,knots,NX,NY,xo,yo,&ier);
  if (ier != 0) {
    printf("Error return from c_csa2s: %d\n",ier);
    exit(1);
  }

/*
 *  Calculate the second order mixed partial.
 */
  nderiv[0] = 1;
  nderiv[1] = 1;
  wts[0] = -1.;
  derivs = c_csa2xs(NI,xi,yi,zi,wts,knots,ssmth,nderiv,NX,NY,xo,yo,&ier);
  if (ier != 0) {
    printf("Error return from c_csa2xs: %d\n",ier);
    exit(1);
  }

/*
 *  Open GKS, open and activate a workstation.
 */
  gopen_gks("stdout",0);
  gopen_ws(WKID, NULL, IWTYPE);
  gactivate_ws(WKID);

/*
 *  Rearrange the C array "function" for input to c_tdez2d.
 */
  zor = (float *) calloc(NX*NY, sizeof(float));
  if (zor == NULL) {
    printf("Unable to allocate temp space\n");
    exit(1);
  }
  for (i = 0; i < NX; i++) {
    for (j = 0; j < NY; j++) {
      zor[j*NX+i] = function[i*NY+j];
    }
  }

/*
 *  Draw plot of approximated function.
 */
  c_tdez2d(NX, NY, xo, yo, zor, 2.7, 45., 78., 6);
  c_set(0.,1.,0.,1.,0.,1.,0.,1.,1);
```

```
  c_plchhq(0.5,0.85,
            ":F25:z = f(x,y) = y:S:2"
            ":E:  - -:H-10::S:1:E::B::V-6:2:E:  y:V-6:*:V+6:x:S:2:E:",
      0.04,0.,0.);
  c_frame();

/*
 *  Rearrange the C array "derivs" for input to c_tdez2d.
 */
  for (i = 0; i < NX; i++) {
    for (j = 0; j < NY; j++) {
      zor[j*NX+i] = derivs[i*NY+j];
    }
  }

/*
 *  Draw plot of approximated function.
 */
  c_tdez2d(NX, NY, xo, yo, zor, 2.7, 45., 78., 6);
  c_set(0.,1.,0.,1.,0.,1.,0.,1.,1);
  c_plchhq(0.5,0.85,
      ":F25:z =  :F34::S::H8:6:F25::S:2:E::E::F34:>:B::F34::H-35::"
      "V-6:6:F25:x:F34:6:F25:y:E:  f(x,y) = - x",
      0.04,0.,0.);
  c_frame();

/*
 *  Deactivate and close workstation, close GKS.
 */
  gdeactivate_ws(WKID);
  gclose_ws(WKID);
  gclose_gks();
}

float dsrnd1()
{

/*
 *  Random number generator returns float.
 */
  return (((float) rand()/ (float) RAND_MAX));
}
```

Example 4 - a two-dimensional example illustrating taking partial derivatives (Fortran)

```
      PROGRAM CSEX04
C
C  Do a 2D approximation and find mixed second order partial derivatives.
C
C
C  The dimensionality of the problem.
C
```

```
      PARAMETER (NDIM=2)
C
C  The number of input data points.
C
      PARAMETER (NDATA=1000)
C
C  The number of output data points in the X coordinate direction.
C
      PARAMETER (NX=29)
C
C  The number of output data points in the Y coordinate direction.
C
      PARAMETER (NY=25)
C
C  Specifty the number of knots in the X direction.
C
      PARAMETER (N1=4)
C
C  Specifty the number of knots in the Y direction.
C
      PARAMETER (N2=4)
C
C  The size of the workspace.
C
      PARAMETER (NCF=N1*N2, NWRK=NCF*(NCF+3))
C
C  Define error file, Fortran unit number, workstation type,
C  and workstation ID.
C
      PARAMETER (IERRF=6, LUNIT=2, IWTYPE=1, IWKID=1)
C
C  Dimension the arrays.
C
      DIMENSION XDATA(NDIM,NDATA),YDATA(NDATA),KNOTS(NDIM),WORK(NWRK),
     +          NDERIV(2)
      DATA XMIN,YMIN,XMAX,YMAX / -1.0, -1.0, 1.0, 1.0/
      DATA NDERIV(1),NDERIV(2) / 0, 0/
      REAL XO(NX), YO(NY), FUNC(NX,NY),FUNCD(NX,NY)
C
C  Generate input data using the functiuon f(x,y) = y**2 - 0.5*y*x**2
C
      INDX = NDATA
      DO 10 I=1,INDX
        XDATA(1,I) = XMIN+(XMAX-XMIN)*DSRND1()
        XDATA(2,I) = YMIN+(YMAX-YMIN)*DSRND1()
        XX = XDATA(1,I)
        YY = XDATA(2,I)
        YDATA(I) = YY*YY - 0.5*YY*XX*XX
   10 CONTINUE
C
C  Create the output grid.
C
      DO 102 I=1,NX
        XO(I) = XMIN+(REAL(I-1)/REAL(NX-1))*(XMAX-XMIN)
  102 CONTINUE
      DO 103 J =1,NY
        YO(J)= YMIN+(REAL(J-1)/REAL(NY-1))*(YMAX-YMIN)
  103 CONTINUE
C
C  Specify the numbers of knots in each coordinate direction.
C
```

```
      KNOTS(1) = N1
      KNOTS(2) = N2
C
C  Calculate the approximated functuion values.
C
      CALL CSA2S (INDX,XDATA,YDATA,KNOTS,NX,NY,XO,YO,FUNC,NWRK,WORK,IER)
C
C  Calculate the second order mixed partial derivative.
C
      NDERIV(1) = 1
      NDERIV(2) = 1
      CALL CSA2XS (INDX,XDATA,YDATA,-1.,KNOTS,0.,NDERIV,NX,NY,
     +                   XO,YO,FUNCD,NWRK,WORK,IER)
C
C  Plot a surface.
C
C  Open GKS and define the foreground and background color.
C
      CALL GOPKS (IERRF, ISZDM)
      CALL GOPWK (IWKID, LUNIT, IWTYPE)
      CALL GACWK (IWKID)
C
C  Approximated function.
C
      CALL TDEZ2D(NX, NY, XO, YO, FUNC, 2.7, 45., 78., 6)
      CALL SET(0.,1.,0.,1.,0.,1.,0.,1.,1)
      CALL PLCHHQ(0.5,0.85,
     +   ':F25:z = f(x,y) = y:S:2:E:  - -:H-10::S:1:E::B::V-6:2:E:  y:V-
     +6:*:V+6:x:S:2:E:',0.04,0.,0.)
      CALL FRAME()
C
C  Mixed partial.
C
      CALL TDEZ2D(NX, NY, XO, YO, FUNCD, 2.7, 45., 78., 6)
      CALL SET(0.,1.,0.,1.,0.,1.,0.,1.,1)
      CALL PLCHHQ(0.5,0.88,
     + ':F25:z =  :F34::S::H8:6:F25::S:2:E::E::F34:>:B::F34::H-35::V-6:6
     +:F25:x:F34:6:F25:y:E:  f(x,y) = - x',0.04,0.,0.)
      CALL FRAME()
C
C Close GKS.
C
      CALL GDAWK (IWKID)
      CALL GCLWK (IWKID)
      CALL GCLKS
C
      STOP
      END
      REAL FUNCTION DSRND1()
C
C  Random number generator.
C
      DATA ISEED/1/
      SAVE ISEED
C
      ISEED = ISEED*1103515245 + 12345
      IT = IAND(ISHIFT(ISEED,-16),32767)
C
      DSRND1 = REAL(IT)/32767.
C
      RETURN
```

```
      END
```

Example 4 - a two-dimensional example illustrating taking partial derivatives (NCL)

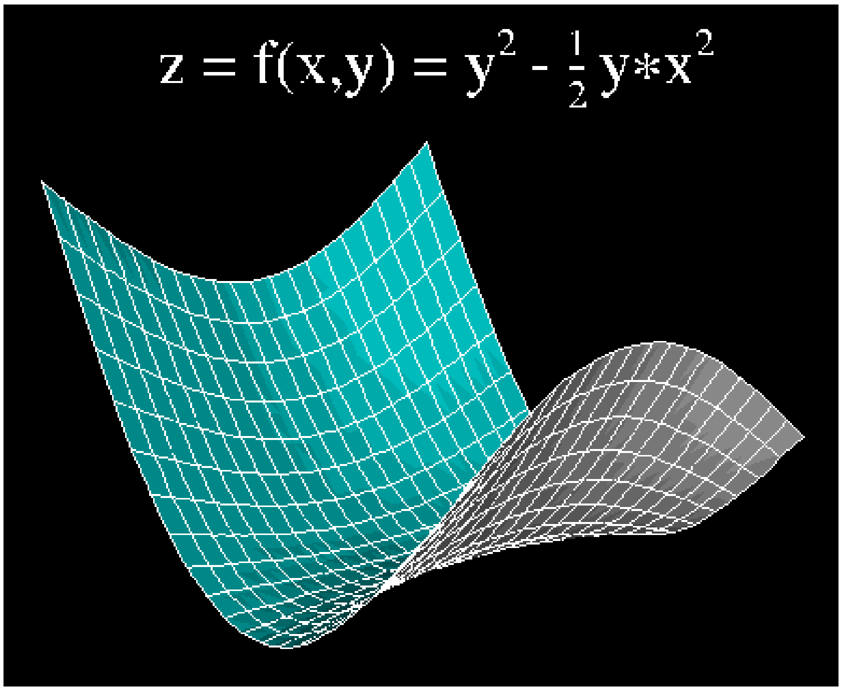
```
load "$NCARG_ROOT/lib/ncarg/nclex/gsun/gsn_code.ncl"

begin

;
;  Create the input arrays.
;
  xmin = -1.
  xmax =  1.
  ymin = -1.
  ymax =  1.

  nx = 29
  ny = 25

  ndata = 1000
  xi = new(ndata,float)
  yi = new(ndata,float)
  zi = new(ndata,float)

;
;  Generate input data using the function f(x,y) = y**2 - 0.5*y*x**2
;
  do i=0,ndata-1
      xi(i) = xmin + (xmax-xmin)*rand()/32767.
      yi(i) = ymin + (ymax-ymin)*rand()/32767.
      zi(i) = yi(i)*yi(i) - 0.5*xi(i)*xi(i)*yi(i)
  end do

;
;  Set up the output grid.
;
  xo = fspan(xmin,xmax,nx)
  yo = fspan(ymin,ymax,ny)

  knots = (/4,4/)
;
;  Calculate the approximated function values.
;
  func = csa2s(xi,yi,zi,knots,xo,yo)

;
;  Calculate the approximated second order mixed partial.
;
  wts = (/-1./)
  smth = 0.0
  nderiv = (/1,1/)
  funcd = csa2xs(xi,yi,zi,wts,knots,smth,nderiv,xo,yo)
```

```
;
; Create workstation object.
;
  NCGM=1
  X11=0
  PS=0

  if (NCGM .eq. 1) then
    wid = gsn_open_wks("ncgm","nm16n")
  end if
  if (X11 .eq. 1) then
    wid = gsn_open_wks("x11","nm16n")
  end if
  if (PS .eq. 1) then
    wid = gsn_open_wks("ps","nm16n")
  end if

;
;  Draw plot of approximated function.
;
  rho = 2.7
  theta = 45.
  phi = 78.
  tdez2d(wid, xo, yo, func, rho, theta, phi, 6)
  txres = True
  txres@txFontHeightF = 0.04
  gsn_text_ndc(wid,                                                      \
     ":F25:z = f(x,y) = y:S:2:E:  - -:H-10::S:1:E::B::V-6:2:E:  x:S:2:E:",  \
     0.5,0.85,txres)
  frame(wid)

;
;  Draw plot of second order mixed partial.
;
  tdez2d(wid, xo, yo, funcd, rho, theta, phi, 6)
  gsn_text_ndc(wid,                                      \
    ":F25:z =  :F34::S::H8:6:F25::S:2:E::E::F34:>:B::F34::H-35::V-6:6:F25:x:F34:6:F2
    0.5,0.85,txres)
  frame(wid)

end
```

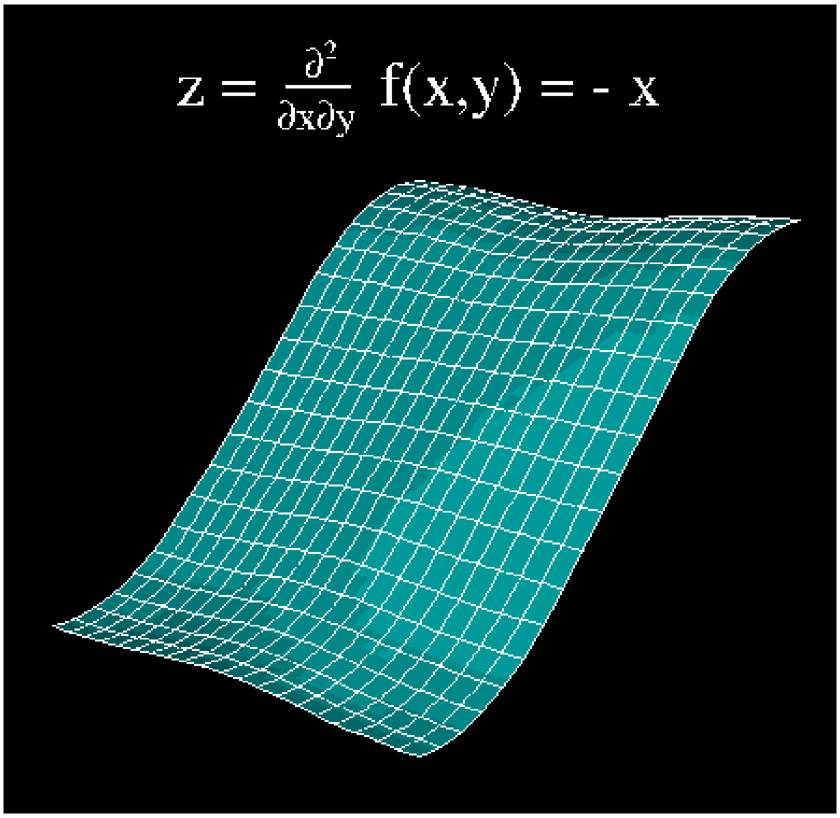
Plots for example 4 - two-dimensional example illustrating taking partial derivatives

---

Original function



Second order mixed partial



Example 5 - a three-dimensional example (sphere) (C)

```
#include <stdio.h>
#include <stdlib.h>
#include <ncarg/ncargC.h>
#include <ncarg/gks.h>
#include <ncarg/ngmath.h>

/*
 *  The number of output data points in the X coordinate direction.
 */
#define NX     21

/*
 *  The number of output data points in the Y coordinate direction.
 */
#define NY     21

/*
 *  The number of output data points in the Z coordinate direction.
 */
#define NZ     21

/*
 *  The number of input data points.
 */
#define NI    1000

/*
 *  The number of knots in the X direction.
 */
#define N1      4

/*
 *  The number of knots in the Y direction.
 */
#define N2      4

/*
 *  The number of knots in the Z direction.
 */
#define N3      4

/*
 *  Data limits.
 */
#define XMIN   -2.0
#define XMAX    2.0
#define YMIN   -2.0
#define YMAX    2.0
#define ZMIN   -2.0
#define ZMAX    2.0
```

```
/*
 *  The GKS workstation type (NCGM).
 */
#define IWTYPE 1

/*
 *  The GKS workstaton identifier.
 */
#define WKID   1

/*
 *  Function prototype for the random number generator.
 */
float dsrnd1();

/*
 * A 3D approximation using function c_csa3s with a sphere as an isosurface.
 */
main ()
{
  float xi[NI],yi[NI],zi[NI],ui[NI],xo[NX],yo[NY],zo[NZ],*surface;
  float *zor,t1,t2;
  int   it1,it2,i,j,k,knots[3],ier;

/*
 *  Generate input data.
 */
  for (i = 0; i < NI; i++) {
    xi[i] = XMIN+(XMAX-XMIN)*dsrnd1();
    yi[i] = YMIN+(YMAX-YMIN)*dsrnd1();
    zi[i] = ZMIN+(ZMAX-ZMIN)*dsrnd1();
    ui[i] = xi[i]*xi[i] + yi[i]*yi[i] + zi[i]*zi[i];
  }

/*
 *  Create the output grid.
 */
  for (i = 0; i < NX; i++) {
    xo[i] = XMIN+((float)i/(float)(NX-1))*(XMAX-XMIN);
  }
  for (j = 0; j < NY; j++) {
    yo[j] = YMIN+((float)j/(float)(NY-1))*(YMAX-YMIN);
  }
  for (k = 0; k < NZ; k++) {
    zo[k] = ZMIN+((float)k/(float)(NZ-1))*(ZMAX-ZMIN);
  }

/*
 *  Specify the numbers of knots in each coordinate direction.
 */
  knots[0] = N1;
  knots[1] = N2;
  knots[2] = N3;

/*
 *  Calculate the approximated functuion values.
 */
  surface = c_csa3s(NI,xi,yi,zi,ui,knots,NX,NY,NZ,xo,yo,zo,&ier);
  if (ier != 0) {
    printf("Error return from c_csa3s: %d\n",ier);
    exit(1);
```

```
  }

/*
 *  Rearrange the C array "surface" for input to c_tdez3d.
 */
  zor = (float *) calloc(NX*NY*NZ, sizeof(float));
  if (zor == NULL) {
    printf("Unable to allocate temp space\n");
    exit(1);
  }
  for (i = 0; i < NX; i++) {
    for (j = 0; j < NY; j++) {
      for (k = 0; k < NZ; k++) {
        it1 = k*NX*NY + j*NX + i;
        it2 = i*NZ*NY + j*NZ + k;
        zor[k*NX*NY + j*NX + i] = surface[i*NZ*NY + j*NZ + k];
      }
    }
  }

/*
 *  Draw plot.
 */
/*
 *  Open GKS, open and activate a workstation.
 */
  gopen_gks("stdout",0);
  gopen_ws(WKID, NULL, IWTYPE);
  gactivate_ws(WKID);

  c_tdez3d(NX, NY, NZ, xo, yo, zo, zor, 3., 2., -35., 65., 6);
  c_frame();

/*
 *  Deactivate and close workstation, close GKS.
 */
  gdeactivate_ws(WKID);
  gclose_ws(WKID);
  gclose_gks();
}

float dsrnd1()
{

/*
 *  Random number generator returns float.
 */
  return (((float) rand()/ (float) RAND_MAX));
}
```

Example 5 - a three-dimensional example (sphere) (Fortran)

```
      PROGRAM CSEX05
C
C  Do a 3D approximation of a sphere.
C
C
C  The dimensionality of the problem.
C
      PARAMETER (NDIM=3)
C
C  The number of input data points.
C
      PARAMETER (NDATA=1000)
C
C  The number of output data points in the X coordinate direction.
C
      PARAMETER (NX=21)
C
C  The number of output data points in the Y coordinate direction.
C
      PARAMETER (NY=21)
C
C  The number of output data points in the Z coordinate direction.
C
      PARAMETER (NZ=21)
C
C  Specifty the number of knots in the X direction.
C
      PARAMETER (N1=4)
C
C  Specifty the number of knots in the Y direction.
C
      PARAMETER (N2=4)
C
C  Specifty the number of knots in the Z direction.
C
      PARAMETER (N3=4)
C
C  The size of the workspace.
C
      PARAMETER (NCF=N1*N2*N3, NWRK=NCF*(NCF+3))
C
C  Dimension the arrays.
C
      DIMENSION XDATA(NDIM,NDATA),YDATA(NDATA),KNOTS(NDIM),WORK(NWRK)
C
C  Define error file, Fortran unit number, and workstation type,
C  and workstation ID (these values are used by GKS).
C
      PARAMETER (IERRF=6, LUNIT=2, IWTYPE=1, IWKID=1)
C
C  Define arrays for use in plotting the isosurface.
C
      REAL XO(NX), YO(NY), ZO(NZ), OUTPUT(NX,NY,NZ)
      INTEGER I, J, K, IER
      DATA XMIN,YMIN,ZMIN,XMAX,YMAX,ZMAX / -2., -2., -2., 2.,  2., 2./
C
C  Define the input data on random coordinates bounded by the
C  above values.
C
      DO 10 I=1,NDATA
        XDATA(1,I) = XMIN+(XMAX-XMIN)*DSRND1()
```

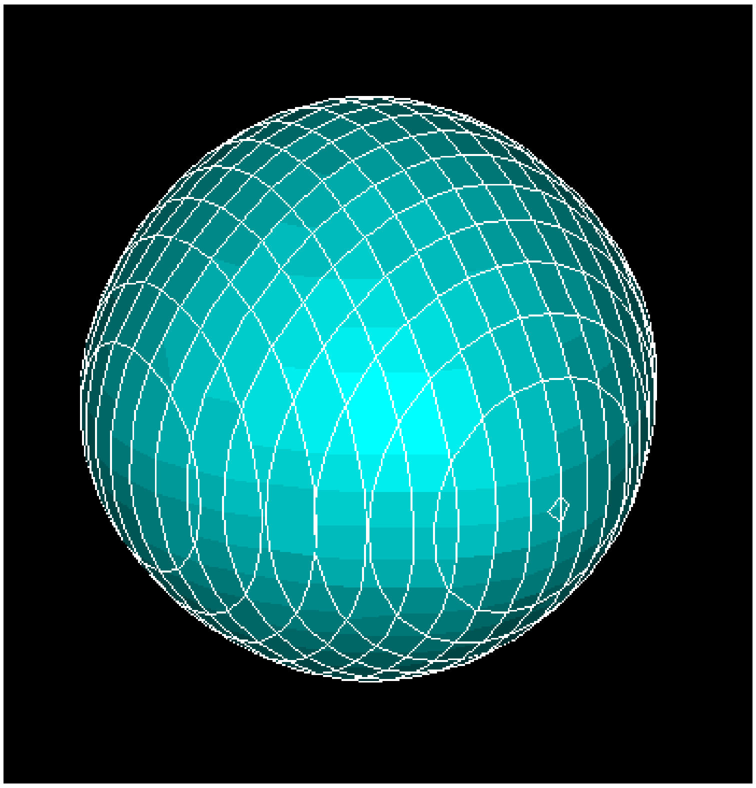
```
        XDATA(2,I) = YMIN+(YMAX-YMIN)*DSRND1()
        XDATA(3,I) = ZMIN+(ZMAX-ZMIN)*DSRND1()
        YDATA(I)  = XDATA(1,I)**2 + XDATA(2,I)**2 + XDATA(3,I)**2
   10 CONTINUE
C
C  Create the output grid.
C
      DO 102 I=1,NX
        XO(I) = XMIN+(REAL(I-1)/REAL(NX-1))*(XMAX-XMIN)
  102 CONTINUE
      DO 103 J =1,NY
        YO(J)= YMIN+(REAL(J-1)/REAL(NY-1))*(YMAX-YMIN)
  103 CONTINUE
      DO 104 K=1,NZ
        ZO(K) = ZMIN+(REAL(K-1)/REAL(NZ-1))*(ZMAX-ZMIN)
  104 CONTINUE
C
C  Specify the numbers of knots in each coordinate.
C
      KNOTS(1) = N1
      KNOTS(2) = N2
      KNOTS(3) = N3
C
C  Find the approximating function values on the output grid.
C
      CALL CSA3S (NDATA,XDATA,YDATA,KNOTS,NX,NY,NZ,
     +            XO,YO,ZO,OUTPUT,NWRK,WORK,IER)
C
C  Plot an isosurface.
C
C  Open GKS and define the foreground and background color.
C
      CALL GOPKS (IERRF, ISZDM)
      CALL GOPWK (IWKID, LUNIT, IWTYPE)
      CALL GACWK (IWKID)
C
      CALL TDEZ3D(NX, NY, NZ, XO, YO, ZO, OUTPUT, 3., 2., -35., 65., 6)
      CALL FRAME()
C
C Close GKS.
C
      CALL GDAWK (IWKID)
      CALL GCLWK (IWKID)
      CALL GCLKS
C
      STOP
      END
      REAL FUNCTION DSRND1()
C
C  Random number generator.
C
      DATA ISEED/1/
      SAVE ISEED
C
      ISEED = ISEED*1103515245 + 12345
      IT = IAND(ISHIFT(ISEED,-16),32767)
C
      DSRND1 = REAL(IT)/32767.
C
      RETURN
      END
```

Example 5 - a three-dimensional example (sphere) (NCL)

```
load "$NCARG_ROOT/lib/ncarg/nclex/gsun/gsn_code.ncl"

begin

;
;  Create the input arrays.
;
  xmin = -2.
  xmax =  2.
  ymin = -2.
  ymax =  2.
  zmin = -2.
  zmax =  2.

  nx = 21
  ny = 21
  nz = 21

  ndata = 1000
  xi = new(ndata,float)
  yi = new(ndata,float)
  zi = new(ndata,float)
  ui = new(ndata,float)

  do i=0,ndata-1
      xi(i) = xmin + (xmax-xmin)*rand()/32767.
      yi(i) = ymin + (ymax-ymin)*rand()/32767.
      zi(i) = zmin + (zmax-zmin)*rand()/32767.
      ui(i) = xi(i)*xi(i) + yi(i)*yi(i) + zi(i)*zi(i)
  end do

;
;  Set up the output grid.
;
  xo = fspan(xmin,xmax,nx)
  yo = fspan(ymin,ymax,ny)
  zo = fspan(zmin,zmax,nz)

;
;  Calculate the values for the approximating cubic spline.
;
  knots = (/4,4,4/)
  uo = csa3s(xi,yi,zi,ui,knots,xo,yo,zo)

;
; Create workstation object.
;

  NCGM=1
```

```
  X11=0
  PS=0

  if (NCGM .eq. 1) then
    wid = gsn_open_wks("ncgm","nm17n")
  end if
  if (X11 .eq. 1) then
    wid = gsn_open_wks("x11","nm17n")
  end if
  if (PS .eq. 1) then
    wid = gsn_open_wks("ps","nm17n")
  end if

;
;  Draw an isosurface plot of the approximated function at isovalue = 3.
;
  rho = 2.
  theta = -35.
  phi = 65.
  tdez3d(wid, xo, yo, zo, uo, 3.0, rho, theta, phi, 6)
  frame(wid)

end
```

Plot for example 5 - a three-dimensional example (sphere)



Example 6 - a three-dimensional example (tube) (C)

```
#include <stdio.h>
#include &ls;stdlib.h>
#include <ncarg/ncargC.h>
#include <ncarg/gks.h>
#include <ncarg/ngmath.h>

/*
 *  The number of output data points in the X coordinate direction.
 */
#define NX      21

/*
 *  The number of output data points in the Y coordinate direction.
 */
#define NY      31

/*
 *  The number of output data points in the Z coordinate direction.
 */
#define NZ      41

/*
```

```
 *  The number of input data points.
 */
#define NI    1000

/*
 *  The number of knots in the X direction.
 */
#define N1      4

/*
 *  The number of knots in the Y direction.
 */
#define N2      4

/*
 *  The number of knots in the Z direction.
 */
#define N3      4

/*
 *  Data limits.
 */
#define XMIN   -2.0
#define XMAX    2.0
#define YMIN   -2.0
#define YMAX    2.0
#define ZMIN   -2.0
#define ZMAX    2.0

/*
 *  The GKS workstation type (NCGM).
 */
#define IWTYPE 1

/*
 *  The GKS workstaton identifier.
 */
#define WKID   1

/*
 *  Function prototype for the random number generator.
 */
float dsrnd1();

/*
 * Test 3D function c_csa3s.
 */
main ()
{
  float xi[NI],yi[NI],zi[NI],ui[NI],xo[NX],yo[NY],zo[NZ],*surface;
  float *zor,t1,t2;
  int   it1,it2,i,j,k,knots[3],ier;

/*
 *  Generate input data.
 */
  for (i = 0; i < NI; i++) {
    xi[i] = XMIN+(XMAX-XMIN)*dsrnd1();
    yi[i] = YMIN+(YMAX-YMIN)*dsrnd1();
    zi[i] = ZMIN+(ZMAX-ZMIN)*dsrnd1();
    ui[i] = 0.75*xi[i]*xi[i] - 1.6*yi[i]*yi[i] + 2.*zi[i]*zi[i];
```

```
  }

/*
 *  Create the output grid.
 */
  for (i = 0; i < NX; i++) {
    xo[i] = XMIN+((float)i/(float)(NX-1))*(XMAX-XMIN);
  }
  for (j = 0; j < NY; j++) {
    yo[j] = YMIN+((float)j/(float)(NY-1))*(YMAX-YMIN);
  }
  for (k = 0; k < NZ; k++) {
    zo[k] = ZMIN+((float)k/(float)(NZ-1))*(ZMAX-ZMIN);
  }

/*
 *  Specify the numbers of knots in each coordinate direction.
 */
  knots[0] = N1;
  knots[1] = N2;
  knots[2] = N3;

/*
 *  Calculate the approximated functuion values.
 */
  surface = c_csa3s(NI,xi,yi,zi,ui,knots,NX,NY,NZ,xo,yo,zo,&ier);
  if (ier != 0) {
    printf("Error return from c_csa3s: %d\n",ier);
    exit(1);
  }

/*
 *  Rearrange the C array "surface" for input to c_tdez3d.
 */
  zor = (float *) calloc(NX*NY*NZ, sizeof(float));
  if (zor == NULL) {
    printf("Unable to allocate temp space\n");
    exit(1);
  }
  for (i = 0; i < NX; i++) {
    for (j = 0; j < NY; j++) {
      for (k = 0; k < NZ; k++) {
        it1 = k*NX*NY + j*NX + i;
        it2 = i*NZ*NY + j*NZ + k;
        zor[k*NX*NY + j*NX + i] = surface[i*NZ*NY + j*NZ + k];
        fflush(stdout);
      }
    }
  }

/*
 *  Draw plot.
 */
/*
 *  Open GKS, open and activate a workstation.
 */
  gopen_gks("stdout",0);
  gopen_ws(WKID, NULL, IWTYPE);
  gactivate_ws(WKID);

  c_tdez3d(NX, NY, NZ, xo, yo, zo, zor, 0.7, 2.3, -13., 75., 6);
```

```
  c_frame();

/*
 *  Deactivate and close workstation, close GKS.
 */
  gdeactivate_ws(WKID);
  gclose_ws(WKID);
  gclose_gks();
}

float dsrnd1()
{

/*
 *  Random number generator returns float.
 */
  return (((float) rand()/ (float) RAND_MAX));
}
```

Example 6 - a three-dimensional example (tube) (Fortran)

```
      PROGRAM CSEX06
C
C  Do a 3D approximation and plot an isosurface.
C
C
C  The dimensionality of the problem.
C
      PARAMETER (NDIM=3)
C
C  The number of input data points.
C
      PARAMETER (NDATA=1000)
C
C  The number of output data points in the X coordinate direction.
C
      PARAMETER (NX=21)
C
C  The number of output data points in the Y coordinate direction.
C
      PARAMETER (NY=31)
C
C  The number of output data points in the Z coordinate direction.
C
      PARAMETER (NZ=41)
C
C  Specifty the number of knots in the X direction.
C
      PARAMETER (N1=4)
C
C  Specifty the number of knots in the Y direction.
C
```

```
      PARAMETER (N2=4)
C
C  Specifty the number of knots in the Z direction.
C
      PARAMETER (N3=4)
C
C  The size of the workspace.
C
      PARAMETER (NCF=N1*N2*N3, NWRK=NCF*(NCF+3))
C
C  Dimension the arrays.
C
      DIMENSION XDATA(NDIM,NDATA),YDATA(NDATA),KNOTS(NDIM),WORK(NWRK)
C
C  Define error file, Fortran unit number, and workstation type,
C  and workstation ID (these values are used by GKS).
C
      PARAMETER (IERRF=6, LUNIT=2, IWTYPE=1, IWKID=1)
C
C  Define arrays for use in plotting the isosurface.
C
      REAL XO(NX),  YO(NY),  ZO(NZ),  OUTPUT(NX,NY,NZ)
      DATA XMIN,YMIN,ZMIN,XMAX,YMAX,ZMAX / -2., -2., -2., 2.,  2., 2./
C
C  Define the input data on random coordinates bounded by the
C  above values.
C
      DO 10 I=1,NDATA
        XDATA(1,I) = XMIN+(XMAX-XMIN)*DSRND1()
        XDATA(2,I) = YMIN+(YMAX-YMIN)*DSRND1()
        XDATA(3,I) = ZMIN+(ZMAX-ZMIN)*DSRND1()
        YDATA(I)  =  0.75*XDATA(1,I)**2 - 1.6*XDATA(2,I)**2
     +                    + 2.0*XDATA(3,I)**2.
   10 CONTINUE
C
C  Create the output grid.
C
      DO 102 I=1,NX
        XO(I) = XMIN+(REAL(I-1)/REAL(NX-1))*(XMAX-XMIN)
  102 CONTINUE
      DO 103 J =1,NY
        YO(J)= YMIN+(REAL(J-1)/REAL(NY-1))*(YMAX-YMIN)
  103 CONTINUE
      DO 104 K=1,NZ
        ZO(K) = ZMIN+(REAL(K-1)/REAL(NZ-1))*(ZMAX-ZMIN)
  104 CONTINUE
C
C  Specify the numbers of nodes in each coordinate.
C
      KNOTS(1) = N1
      KNOTS(2) = N2
      KNOTS(3) = N3
C
C  Calculate the approximating function.
C
      CALL CSA3S (NDATA,XDATA,YDATA,KNOTS,NX,NY,NZ,
     +            XO,YO,ZO,OUTPUT,NWRK,WORK,IER)
C
C  Plot an isosurface.
C
C Open GKS and define the foreground and background color.
```

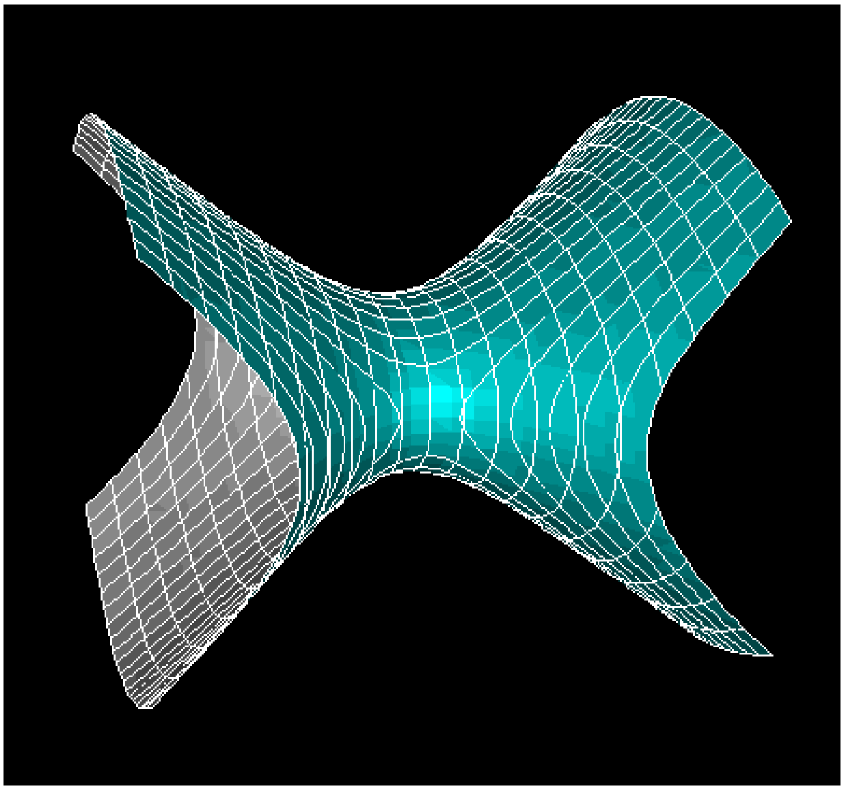
```
C
      CALL GOPKS (IERRF, ISZDM)
      CALL GOPWK (IWKID, LUNIT, IWTYPE)
      CALL GACWK (IWKID)
      CALL TDEZ3D(NX, NY, NZ, XO, YO, ZO, OUTPUT, .7,
     +             2.3, -13., 75., 6)
      CALL FRAME()
C
C Close GKS.
C
      CALL GDAWK (IWKID)
      CALL GCLWK (IWKID)
      CALL GCLKS
C
      STOP
      END
      REAL FUNCTION DSRND1()
C
C  Random number generator.
C
      DATA ISEED/1/
      SAVE ISEED
C
      ISEED = ISEED*1103515245 + 12345
      IT = IAND(ISHIFT(ISEED,-16),32767)
C
      DSRND1 = REAL(IT)/32767.
C
      RETURN
      END
```

Example 6 - a three-dimensional example (tube) (NCL)

```
load "$NCARG_ROOT/lib/ncarg/nclex/gsun/gsn_code.ncl"

begin

;
;  Create the input arrays.
;
  xmin = -2.
  xmax =  2.
  ymin = -2.
  ymax =  2.
  zmin = -2.
  zmax =  2.

  nx = 21
  ny = 31
  nz = 41

  ndata = 1000
```

```
  xi = new(ndata,float)
  yi = new(ndata,float)
  zi = new(ndata,float)
  ui = new(ndata,float)

  do i=0,ndata-1
      xi(i) = xmin + (xmax-xmin)*rand()/32767.
      yi(i) = ymin + (ymax-ymin)*rand()/32767.
      zi(i) = zmin + (zmax-zmin)*rand()/32767.
      ui(i) = 0.75*xi(i)*xi(i) - 1.6*yi(i)*yi(i) + 2.*zi(i)*zi(i)
  end do

;
;  Set up the output grid.
;
  xo = fspan(xmin,xmax,nx)
  yo = fspan(ymin,ymax,ny)
  zo = fspan(zmin,zmax,nz)

;
;  Calculate the values for the approximating cubic spline.
;
  knots = (/4,4,4/)
  uo = csa3s(xi,yi,zi,ui,knots,xo,yo,zo)

;
; Create workstation object.
;
  NCGM=1
  X11=0
  PS=0

  if (NCGM .eq. 1) then
    wid = gsn_open_wks("ncgm","nm18n")
  end if
  if (X11 .eq. 1) then
    wid = gsn_open_wks("x11","nm18n")
  end if
  if (PS .eq. 1) then
    wid = gsn_open_wks("ps","nm18n")
  end if

;
;  Draw plot of approximated function.
;
  rho = 2.3
  theta = -13.
  phi = 75.
  tdez3d(wid, xo, yo, zo, uo, 0.7, rho, theta, phi, 6)
  frame(wid)

end
```

Plot for example 6 - a three-dimensional example (tube)



Example 7 - a two-dimensional example using list output (C)

```
#include <stdio.h>
#include <stlib.h>
#include <ncarg/ncargC.h>
#include <ncarg/gks.h>
#include <ncarg/ngmath.h>

/*
 *  The number of input data points.
 */
#define NDATA   500

/*
 *  The number of output data points in the X coordinate direction.
 */
#define NX      29

/*
 *  The number of output data points in the Y coordinate direction.
 */
#define NY      25
```

```
/*
 *  The number of output data points.
 */
#define NO     NX*NY

/*
 *  The number of knots in the X direction.
 */
#define N1      10

/*
 *  The number of knots in the Y direction.
 */
#define N2      10

/*
 *  Data limits.
 */
#define XMIN   -1.4
#define XMAX    1.4
#define YMIN   -1.2
#define YMAX    1.2

/*
 *  The GKS workstation type (NCGM).
 */
#define IWTYPE  1

/*
 *  The GKS workstaton identifier.
 */
#define WKID    1

/*
 *  Function prototype for the random number generator.
 */
float dsrnd1();

/*
 *  Do a 2D approximation using a list of output coordinates.
 */
main ()
{
  float xi[NDATA],yi[NDATA],zi[NDATA],xo[NO],yo[NO],*surface,t1,t2;
  float xp[NX],yp[NY];
  int   i,j,indx=0,knots[2],ier;

/*
 *  Create the data array for the surface.
 */
  for (i = 0; i < NDATA; i++) {
    xi[i] = XMIN+(XMAX-XMIN)*dsrnd1();
    yi[i] = YMIN+(YMAX-YMIN)*dsrnd1();
    zi[i] = xi[i] + yi[i];
    t1 = 1.0/(pow(fabs(xi[i]-0.1),2.75) + pow(fabs(yi[i]),2.75) + 0.09);
    t2 = 1.0/(pow(fabs(xi[i]+0.1),2.75) + pow(fabs(yi[i]),2.75) + 0.09);
    zi[i] = 0.3*(zi[i]+t1-t2);
  }
  for (j = 0; j < NY; j++) {
    for (i = 0; i < NX; i++) {
      xo[indx] = XMIN+((float)i/(float)(NX-1))*(XMAX-XMIN);
```

```
      yo[indx] = YMIN+((float)j/(float)(NY-1))*(YMAX-YMIN);
      indx++;
    }
  }

  knots[0] = N1;
  knots[1] = N2;

  surface = c_csa2ls(NDATA,xi,yi,zi,knots,NO,xo,yo,&ier);
  if (ier != 0) {
    printf("Error return from c_csa2xs: %d\n",ier);
    exit(1);
  }

/*
 *  Draw plot.
 */
/*
 *  Open GKS, open and activate a workstation.
 */
  gopen_gks("stdout",0);
  gopen_ws(WKID, NULL, IWTYPE);
  gactivate_ws(WKID);

/*
 *  Create the output grid for plotting.
 */
  for (i = 0; i < NX; i++) {
    xp[i] = XMIN+((float)(i)/(float)(NX-1))*(XMAX-XMIN);
  }
  for (j = 0; j < NY; j++) {
    yp[j] = YMIN+((float)(j)/(float)(NY-1))*(YMAX-YMIN);
  }

/*
 *  Draw the 2D surface.  Since "surface" is already in column
 *  dominate order, we need not rearrange it for input to c_tdez2d.
 */
  c_tdez2d(NX, NY, xp, yp, surface, 2.5, -154., 80., 6);
  c_frame();

/*
 *  Deactivate and close workstation, close GKS.
 */
  gdeactivate_ws(WKID);
  gclose_ws(WKID);
  gclose_gks();
}

float dsrnd1()
{

/*
 *  Random number generator returns float.
 */
  return (((float) rand()/ (float) RAND_MAX));
}
```

Example 7 - a two-dimensional example using list output (Fortran)

```
      PROGRAM CSEX07
C
C  Do a 2D approximation using a list of output coordinates.
C
C
C  The dimensionality of the problem.
C
      PARAMETER (NDIM=2)
C
C  The number of input data points.
C
      PARAMETER (NDATA=500)
C
C  The number of output data points in the X coordinate direction.
C
      PARAMETER (NX=29)
C
C  The number of output data points in the Y coordinate direction.
C
      PARAMETER (NY=25)
C
C  The number of output data points.
C
      PARAMETER (NO=NX*NY)
C
C  The number of knots in the X direction.
C
      PARAMETER (N1=10)
C
C  The number of knots in the Y direction.
C
      PARAMETER (N2=10)
C
C  The size of the workspace.
C
      PARAMETER (NCF=N1*N2, NWRK=NCF*(NCF+3))
C
C  Define error file, Fortran unit number, workstation type,
C  and workstation ID.
C
      PARAMETER (IERRF=6, LUNIT=2, IWTYPE=1, IWKID=1)
C
C  Dimension the arrays.
C
      DIMENSION XDATA(NDIM,NDATA),YDATA(NDATA),NODES(NDIM),WORK(NWRK)
      DATA XMIN,YMIN,XMAX,YMAX / -1.4, -1.2, 1.4, 1.2/
      REAL XO(NO),  YO(NO), OUTPUT(NO)
      REAL XP(NX),  YP(NY), OPLOT(NX,NY)
C
C  Create the data array for the surface.
C
      DO 10 I=1,NDATA
```

```
        XDATA(1,I) = XMIN+(XMAX-XMIN)*DSRND1()
        XDATA(2,I) = YMIN+(YMAX-YMIN)*DSRND1()
        YDATA(I)   = XDATA(1,I) + XDATA(2,I)
        T1 = 1.0/((ABS(XDATA(1,I)-0.1))**2.75 +
     +                     ABS(XDATA(2,I))**2.75+0.09)
        T2 = 1.0/((ABS(XDATA(1,I)+0.1))**2.75 +
     +                     ABS(XDATA(2,I))**2.75+0.09)
        YDATA(I) = 0.3*(YDATA(I)+T1-T2)
   10 CONTINUE
C
C  Create the output arrays.
C
      INDX = 0
      DO 103 J=1,NY
        DO 104 I=1,NX
          INDX = INDX+1
          XO(INDX) = XMIN+(REAL(I-1)/REAL(NX-1))*(XMAX-XMIN)
          YO(INDX) = YMIN+(REAL(J-1)/REAL(NY-1))*(YMAX-YMIN)
  104   CONTINUE
  103 CONTINUE
C
C  Specify the numbers of nodes in each coordinate.
C
      NODES(1) = N1
      NODES(2) = N2
C
      CALL CSA2LS (NDATA,XDATA,YDATA,NODES,NO,XO,YO,
     +             OUTPUT,NWRK,WORK,IER)
C
C  Plot the 2D surface approximation.
C
C Open GKS and define the foreground and background color.
C
      CALL GOPKS (IERRF, ISZDM)
      CALL GOPWK (IWKID, LUNIT, IWTYPE)
      CALL GACWK (IWKID)
C
C  Convert the linear output array to a 2D array.
C
      DO 21 I=1,NX
        DO 22 J=1,NY
            OPLOT(I,J) = OUTPUT((J-1)*NX + I)
   22   CONTINUE
   21 CONTINUE
C
C  Create the output grid for plotting.
C
      DO 105 I=1,NX
        XP(I) = XMIN+(REAL(I-1)/REAL(NX-1))*(XMAX-XMIN)
  105 CONTINUE
      DO 106 J=1,NY
        YP(J) = YMIN+(REAL(J-1)/REAL(NY-1))*(YMAX-YMIN)
  106 CONTINUE
C
C  Plot the surface.
C
      CALL TDEZ2D(NX, NY, XP, YP, OPLOT, 2.5, -154., 80., 6)
      CALL FRAME()
C
C Close GKS.
C
```

```
      CALL GDAWK (IWKID)
      CALL GCLWK (IWKID)
      CALL GCLKS
C
      STOP
      END
      REAL FUNCTION DSRND1()
C
C  Random number generator.
C
      DATA ISEED/1/
      SAVE ISEED
C
      ISEED = ISEED*1103515245 + 12345
      IT = IAND(ISHIFT(ISEED,-16),32767)
C
      DSRND1 = REAL(IT)/32767.
C
      RETURN
      END
```

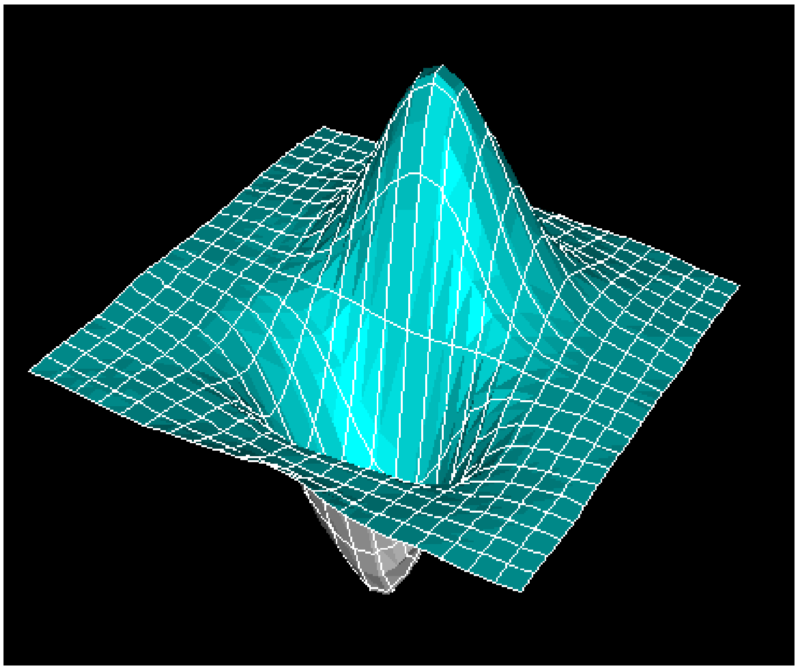
Example 7 - a two-dimensional example using list output (NCL)

```
load "$NCARG_ROOT/lib/ncarg/nclex/gsun/gsn_code.ncl"

begin

;
;  Create the input arrays.
;
xmin = -1.4
xmax =  1.4
ymin = -1.2
ymax =  1.2

ndata = 500
xi = new(ndata,float)
yi = new(ndata,float)
zi = new(ndata,float)

do i=0,ndata-1
    xi(i) = xmin + (xmax-xmin)*rand()/32767.
    yi(i) = ymin + (ymax-ymin)*rand()/32767.
    zi(i) = xi(i) + yi(i)
    t1 = 1.0/((fabs(xi(i)-0.1))^2.75 + fabs(yi(i))^2.75 + 0.09)
    t2 = 1.0/((fabs(xi(i)+0.1))^2.75 + fabs(yi(i))^2.75 + 0.09)
    zi(i) = 0.3*(zi(i)+t1-t2)
end do

;
;  Set up a list of output points that corresponds to a 2D grid.
;
```

```
nx = 29
ny = 25
no = nx*ny
xo = new(no,float)
yo = new(no,float)
scalex = 1./(nx-1)*(xmax-xmin)
do i=0,nx-1
  istart = i*ny
  iend   = istart+ny-1
  xo(istart:iend) = xmin + i*scalex
  yo(istart:iend) = fspan(ymin,ymax,ny)
end do

;
;  Find the approximation surface.
;
knots = (/10,10/)
zo = csa2ls(xi,yi,zi,knots,xo,yo)

;
;  Convert to a 2D array for plotting.
;
zo2 = onedtond(zo,(/nx,ny/))

;
; Create workstation object.
;
  NCGM=1
  X11=0
  PS=0

  if (NCGM .eq. 1) then
    wid = gsn_open_wks("ncgm","nm19n")
  end if
  if (X11 .eq. 1) then
    wid = gsn_open_wks("x11","nm19n")
  end if
  if (PS .eq. 1) then
    wid = gsn_open_wks("ps","nm19n")
  end if

;
;  Draw plot.
;
rho = 2.5
theta = -154.
phi = 80.
xo2 = fspan(xmin,xmax,nx)
yo2 = fspan(ymin,ymax,ny)
tdez2d(wid, xo2, yo2, zo2, rho, theta, phi, 6)
frame(wid)

end
```

Plot for example 7 - a two-dimensional example

using list output



Error table

The following table lists all of the error numbers and the associated error messages in the csagrid package:

Error number	Error message
33	Internal error in matrix computation, contact a consultant.
34	Failure in matrix computation, usually indicates insufficient number of input data.
35	Internal error in matrix computation, contact a consultant.
102	Number of knots is less than 4.
103	Input data range is zero.
104	Partial derivative order out of range zero to two.
105	Insufficient number of input data.
106	Insufficient work space (Fortran entry only).
107	Failure in matrix computation, usually indicates insufficient number of input data.
202	Number of knots is less than 4 in some coordinate direction.
203	Insufficient workspace.