

geomtesta

March 4, 2011
11:18

Contents

1	A program for generating a basic set of output suitable for use with external graphics packages	1
2	INDEX	21

1 A program for generating a basic set of output suitable for use with external graphics packages

This started out as a geometry test routine, hence the name. The code reads the primary output netCDF file as well as the other inputs. These data are mapped on to a pixel based mesh, either 2-D or 3-D, that can be easily processed by most graphics packages (e.g., VisIt or IDL). The size, location, and orientation of the pixel based mesh is specified in the input file *geometry.inp* (the name is hard coded) and is extremely flexible. For example, the axes of the pixel mesh need not be orthogonal to one another.

The original format of the files generated by this routine was HDF (more specifically, HDF Version 4). With this format, each variable is output to a separate, appropriately named file. The Silo format is now available as an alternative. The lower level implementation of the Silo files is in turn provided by either PDB or HDF5 drivers, depending on how the Silo library was compiled. The format to be used is specified by the *GRAPH_FILE* variable in the *Makefile* or *Makefile.local* as either *HDF4*, *SILO*, or *SILO_HDF5*. The Silo format naturally handles multiple variables in a single file; that file is just the name of the main output netCDF file with *.silo* replacing *.nc*. The variables that are output are hardcoded below, although some control is provided by FWEB macros to enable or disable particular variables.

The format of the input file is fixed, although comments (with #) are allowed.

1. The first line contains (x, y, z) coordinates of the origin of the pixel mesh. This need not be inside the bounds of the problem space.
2. The second line contains the (x, y, z) vector for the first axis of the basic 2-D slice; this need not be a unit vector. If desired, the number of x - y pixels can be specified at the end of this line. If absent, the default value of 40,000 is used.
3. The third line has the (x, y, z) unit vector for the second axis.
4. In the 2-D mode of operation, the fourth line contains just two real numbers corresponding to lengths of the two vectors given in the two previous lines. If, however, the code finds three reals on this line, it will assume the user wants a stack of 2-D slices in a 3-D file and that this line contains the (x, y, z) vector for the third axis. The z resolution can be controlled by the specification of a fourth *real* number on this line, *z_scale*.
5. In the 3-D mode of operation, the fifth line contains three real numbers corresponding to the lengths of the three vectors given on the previous lines.

The mesh spacing in the x and y directions is equal (square pixels), determined by dividing the area of the slice by the total number of pixels. For the z direction, this spacing is multiplied by the *z_scale* factor. If absent, *z_scale* defaults to 2.0. Some example input files can be found in the set of example runs provided with the DEGAS 2 distribution.

External zone based data can also be read in via text files and will be output in the same manner as the other variables. The names of the text files can follow the above data in the *geometry.inp* file. Each text file can have keywords “name”, “format” (as in FORTRAN output; not used in SILO mode), “units” to provide additional information on the data. If provided the name will be used to name the file and variable. If not, the name will be *ext_var_n* where n is the number of the file in the list of external files. All other lines in the file should be in the form:

```
zone    data
```

§1-§1.1 [#1-#2] **geomtesta**A program for generating a basic set of output suitable for use with external graphics packages

where *zone* is an integer between 1 and *zn_num* (inclusive) and **data** is the (real) data value.

By the definition of the mesh, mesh point (i, j, k) corresponds to the physical point x according to:

$$\vec{x} = \vec{x}_0 + (i + \frac{1}{2})\Delta_1\hat{v}_1 + (j + \frac{1}{2})\Delta_2\hat{v}_2 + (k + \frac{1}{2})\Delta_3\hat{v}_3, \quad (1)$$

where \vec{x}_0 are the coordinates of the origin, \hat{v}_1 is the unit vector describing the first axis, Δ_1 is the mesh spacing along that axis, and so on for the other two axes.

The mesh coordinate axes (x_1, x_2, x_3) are computed via:

$$x_1(i) = \vec{x}_0 \cdot \hat{v}_1 + (i + \frac{1}{2})\Delta_1, \quad (2)$$

$$x_2(j) = \vec{x}_0 \cdot \hat{v}_2 + (j + \frac{1}{2})\Delta_2, \quad (3)$$

and

$$x_3(k) = \vec{x}_0 \cdot \hat{v}_3 + (k + \frac{1}{2})\Delta_3. \quad (4)$$

Eliminating the mesh indices between these two sets of equations provides the mapping between the mesh and physical coordinates:

$$\vec{x} = \vec{x}_0 + (x_1 - \vec{x}_0 \cdot \hat{v}_1)\hat{v}_1 + (x_2 - \vec{x}_0 \cdot \hat{v}_2)\hat{v}_2 + (x_3 - \vec{x}_0 \cdot \hat{v}_3)\hat{v}_3. \quad (5)$$

For the common case in which the three unit vectors specifying the mesh axes are orthogonal to each other, this mapping simplifies to the obvious form:

$$\vec{x} = x_1\hat{v}_1 + x_2\hat{v}_2 + x_3\hat{v}_3. \quad (6)$$

```
$Id: geomtesta.web,v 1.29 2009/11/25 19:40:15 dstotler Exp $
```

```
"geomtesta.f" 1 ≡  
@m FILE 'geomtesta.web'
```

The main program.

```
"geomtesta.f" 1.1 ≡  
  program geometry_test  
    implicit_none_f77  
    implicit_none_f90  
  
    call readfilenames  
    call degas_init  
    call nc_read_output  
  
    call exercise  
  
  stop  
end
```

⟨Functions and Subroutines 1.2⟩

Exercise the locate function

```

/* Remove characters causing file name problems */

"geomtesta.f" 1.2 ≡
  @m name_clean(s, sp) sp = s
    ind_tmp = index(sp, '(')
    if (ind_tmp > 0)
      sp(ind_tmp : ind_tmp) = '_'
    ind_tmp = index(sp, ')')
    if (ind_tmp > 0)
      sp(ind_tmp : ind_tmp) = '_'
    ind_tmp = index(sp, '|')
    if (ind_tmp > 0)
      sp(ind_tmp : ind_tmp) = '_'
    ind_tmp = index(sp, '+')
    if (ind_tmp > 0)
      sp(ind_tmp : ind_tmp) = 'p'

⟨Functions and Subroutines 1.2⟩ ≡
  subroutine exercise

    define_dimen(a_ind, 0, na - 1)
    define_dimen(b_ind, 0, nb - 1)
    define_dimen(c_ind, 0, nc - 1)
    define_dimen(pixel_ind, 0, na * nb * nc - 1)
    define_dimen(ext_file_ind, n_ext_file)

    define_varp(xa, FLOAT, a_ind)
    define_varp(xb, FLOAT, b_ind)
    define_varp(xc, FLOAT, c_ind)
    define_varp(pixel_zones, INT, pixel_ind)
    define_varp(pixel_data, FLOAT, pixel_ind)
    define_varp(zone_data, FLOAT, zone_ind)
    define_varp(ext_filenames, CHAR, filenames_size, ext_file_ind)

    implicit_none_f77
    sp_common
    pr_common
    tl_common
    ou_common
    bk_common
    zn_common
    de_common
    rf_common
    implicit_none_f90
    real x3, a3, b3, c3, xp3, vz3, /* Local */
      alen, blen, clen, ha_rate, halpha_tot, ha_temp, density, pressure, max_v, clenp, dx, dz, z_scale,
      main_z_scale
    integer i, j, k, zone, test, back, iview, length, p, beg, e, ind_tmp, na, nb, nc, pixsize, main_pixsize,
      ifile, n_ext_file, open_stat
    integer index_parameters ul_index_max
    external extract_output_datum
    real extract_output_datum
    character*4 ivlab

```

```

character*1 ind_sy3
character*2 iflab
character*3 vtag, auth
character*tl_tag_length vname
character*sp_sy_len clean_sy
character*LINELEN line, ext_var_name, ext_var_units, ext_var_format
data ind_sy/'1', '2', '3'/

declare_varp(xa)
declare_varp(xb)
declare_varp(xc)
declare_varp(pixel_zones)
declare_varp(pixel_data)
declare_varp(zone_data)
declare_varp(ext_filenames)

⟨Memory allocation interface 0⟩
st_decls
tl_decls
vc_decls
@#if SILO
  include 'silo.inc'
  integer dbfile, ret
  logical write_mesh
  character*FILELEN mesh_name, silo_file
  common/silo_common/ dbfile, write_mesh, mesh_name

  e = index(filenamees_array_outputfile, '.nc')
  assert(e > 0)
  silo_file = filenamees_array_outputfile SP(: e) || 'silo'
  ret = dbcreate(trim(silo_file), string_length(silo_file), DB_CLOBBER, DB_LOCAL, DB_F77NULL,
    0, silo_format, dbfile)
  assert((ret ≠ -1) ∧ (dbfile ≠ -1))
@#endif
  open(unit = disk_in, file = 'geometry.inp', status = 'old', form = 'formatted')
  /* Switched these from free format reads to tokens. To maintain backward compatibility, have
    made the switch without adding keywords. As a consequence, the format becomes inflexible. The
    expected contents of the file are now: First line contains (x,y,z) coordinates of an origin. */
  assert(read_string(disk_in, line, length))
  assert(length ≤ len(line))
  length = parse_string(line(: length))
  p = 0
  assert(next_token(line, beg, e, p))
  x1 = read_real(line(beg : e))
  assert(next_token(line, beg, e, p))
  x2 = read_real(line(beg : e))
  assert(next_token(line, beg, e, p))
  x3 = read_real(line(beg : e)) /* Second line contains the (x,y,z) unit vector for the first axis of
    the basic 2-D slice. If desired, the number of x-y pixels can be specified after the x unit vector.
    If absent, the default value of 40,000 is used. */
  assert(read_string(disk_in, line, length))
  assert(length ≤ len(line))
  length = parse_string(line(: length))
  p = 0

```

```

assert(next_token(line, beg, e, p))
a1 = read_real(line(beg : e))
assert(next_token(line, beg, e, p))
a2 = read_real(line(beg : e))
assert(next_token(line, beg, e, p))
a3 = read_real(line(beg : e))
if (next_token(line, beg, e, p)) then
    main_pixsize = read_integer(line(beg : e))
    assert(main_pixsize > 0)
else
    main_pixsize = 40000
end if /* The third line has the (x, y, z) unit vector for the second axis. */
assert(read_string(diskin, line, length))
assert(length ≤ len(line))
length = parse_string(line(: length))
p = 0
assert(next_token(line, beg, e, p))
b1 = read_real(line(beg : e))
assert(next_token(line, beg, e, p))
b2 = read_real(line(beg : e))
assert(next_token(line, beg, e, p))
b3 = read_real(line(beg : e)) /* In the 2-D mode of operation, the fourth line contains just two
    real numbers corresponding to lengths of the two vectors given in the two previous lines. */
assert(read_string(diskin, line, length))
assert(length ≤ len(line))
length = parse_string(line(: length))
p = 0
assert(next_token(line, beg, e, p))
alen = read_real(line(beg : e))
assert(next_token(line, beg, e, p))
blen = read_real(line(beg : e))
clen = zero /* If, however, the code finds three reals on this line, it will assume the user wants a
    stack of 2-D slices in a 3-D file. As above, the z resolution can be controlled, if desired by the
    specification of a fourth real number on this line. The mesh spacing in the x and y directions is
    equal; we multiply that by this z_scale factor to get the spacing in the z direction. If absent, it
    defaults to 2.0. Whether or not z_scale is present, the fourth line contains the unit vector for
    the third direction. */
if (next_token(line, beg, e, p)) then
    c1 = alen
    c2 = blen
    c3 = read_real(line(beg : e))
    if (next_token(line, beg, e, p)) then
        main_z_scale = read_real(line(beg : e))
        assert(main_z_scale > zero)
    else
        main_z_scale = two
    end if /* The fifth line has the length of the three vectors in the 3-D case. */
assert(read_string(diskin, line, length))
assert(length ≤ len(line))
length = parse_string(line(: length))
p = 0
assert(next_token(line, beg, e, p))

```

```

    alen = read_real(line(beg : e))
    assert(next_token(line, beg, e, p))
    blen = read_real(line(beg : e))
    assert(next_token(line, beg, e, p))
    clen = read_real(line(beg : e))
end if /* Check the end of the file for additional (external) zone based data in text files. */
    n_ext_file = 0
    var_alloc(ext_filenames)
loop: continue
if (read_string(diskin, line, length)) then
    assert(length ≤ len(line))
    length = parse_string(line(: length))
    p = 0
    assert(next_token(line, beg, e, p))
    n_ext_file++
    var_realloca(ext_filenames)
    ext_filenames_n_ext_file = line(beg : e)
    go to loop
end if
close(unit = diskin)

    var_alloc(zone_data) /* High resolution 2-D plot(s) */
    pixsize = 6400000
    clenp = zero
    z_scale = zero // Not used
    call get_pixel_mesh(alen, blen, clenp, pixsize, z_scale, na, nb, nc, dx, dz)

    var_alloc(xa)
    var_alloc(xb)
    var_alloc(xc)
    var_alloc(pixel_zones)
    var_alloc(pixel_data) /* Plot the slice midway along the third direction. */
    if (clen > zero) then
        vc_unit(c, vz)
        vc_xvt(x, vz, half * clen, xp)
    else
        vc_copy(x, xp)
    end if
    call set_pixel_zones(xp, a, b, c, na, nb, nc, dx, dz, xa, xb, xc, pixel_zones)
@#if SILO
    write_mesh = T
    mesh_name = 'HighRes2d'
@#endif
    /* This is just Charles' original "zone function". I believe the purpose was to use (most of) an
       8-bit palette to color the zones with a minimum chance of color matching between adjacent
       zones (i.e., suspect that 97/252 is related to the golden mean). */
    do i = 1, zn_num
        zone_data_i = areal(3 + mod(97 * i, 255 - 3))
    end do
    call write_hdf('geomtesta.hdf', zone_data, 'zone_function', '□', 'E11.3', na, nb, nc, xa,
        xb, xc, pixel_zones, pixel_data)

    var_free(xa)
    var_free(xb)

```

```

var_free(xc)
var_free(pixel_zones)
var_free(pixel_data) /* Low resolution, 2- or 3-D plots (as specified on input) */
pixsize = main_pixsize
z_scale = main_z_scale
call get_pixel_mesh(alen, blen, clen, pixsize, z_scale, na, nb, nc, dx, dz)

var_alloc(xa)
var_alloc(xb)
var_alloc(xc)
var_alloc(pixel_zones)
var_alloc(pixel_data)

call set_pixel_zones(x, a, b, c, na, nb, nc, dx, dz, xa, xb, xc, pixel_zones)
@#if SILO
write_mesh = T
mesh_name = 'LowRes2d'
@#endif
do i = 1, zn_num
zone_data_i = areal(i)
end do
call write_hdf('zones.hdf', zone_data, 'zone_number', '□', 'E11.3', na, nb, nc, xa, xb, xc,
pixel_zones, pixel_data)

do i = 1, zn_num
zone_data_i = zn_volume(i)
end do
call write_hdf('volume.hdf', zone_data, 'zone_volume', 'm**3', 'E11.3', na, nb, nc, xa, xb,
xc, pixel_zones, pixel_data)

do i = 1, zn_num
zone_data_i = areal(zn_type(i))
end do
call write_hdf('zone_type.hdf', zone_data, 'zone_number', '□', 'E11.3', na, nb, nc, xa, xb,
xc, pixel_zones, pixel_data)

@#if 1
do i = 1, zn_num
if (zn_type(i) ≡ zn_plasma) then
zone_data_i = bk.n(1, i) // BACKGROUND DENSITY
else
zone_data_i = zero
end if
end do
call write_hdf('bk_n1.hdf', zone_data, 'electron_density', 'm**-3', 'E11.3', na, nb, nc,
xa, xb, xc, pixel_zones, pixel_data)

do i = 1, zn_num
if (zn_type(i) ≡ zn_plasma) then // BACKGROUND TEMPERATURE
zone_data_i = bk.temp(1, i) / electron_charge
else
zone_data_i = zero
end if
end do
call write_hdf('bk_t1.hdf', zone_data, 'electron_temperature', 'eV', 'E11.3', na, nb, nc,
xa, xb, xc, pixel_zones, pixel_data)

```

```

do i = 1, zn_num
  if (zn_type(i) ≡ zn_plasma) then // BACKGROUND TEMPERATURE
    zone_data_i = bk_temp(2, i) / electron_charge
  else
    zone_data_i = zero
  end if
end do
call write_hdf('bk_t2.hdf', zone_data, 'ion_temperature', 'eV', 'E11.3', na, nb, nc, xa,
  xb, xc, pixel_zones, pixel_data)

do j = 1, 3
  max_v = zero
  do i = 1, zn_num
    if (zn_type(i) ≡ zn_plasma) then // ION VELOCITY
      zone_data_i = bk_v(2, i)_j
      max_v = max(max_v, abs(zone_data_i))
    else
      zone_data_i = zero
    end if
  end do
  if (max_v > zero) then
    call write_hdf('bk_v' || ind_sy_j || '.hdf', zone_data, 'ion_velocity_' || ind_sy_j, 'm/s',
      'E11.3', na, nb, nc, xa, xb, xc, pixel_zones, pixel_data)
  end if
end do
@#endif
@#if 0 // Plots of de_zone_fraqs

do j = 1, 4
  if (j ≡ 1) then
    iview = 1
    ivlab = '1'
  else if (j ≡ 2) then
    iview = 4
    ivlab = '4'
  else if (j ≡ 3) then
    iview = 6
    ivlab = '6'
  else
    iview = 9
    ivlab = '9'
  end if
  do i = 1, zn_num
    if (zn_type(i) ≡ zn_plasma ∨ zn_type(i) ≡ zn_vacuum) then
      zone_data_i = de_zone_fraqs_i,iview
    else
      zone_data_i = zero
    end if
  end do
  call write_hdf('zn_frag_' || trim(ivlab) || '.hdf', zone_data, 'zone_frag_' || trim(ivlab),
    '1', 'E11.3', na, nb, nc, xa, xb, xc, pixel_zones, pixel_data)
end do
@#endif

```

```

@#if 1
  assert(output_old_file == TRUE)
  do j = 2, pr_test_num
    index_parameters_tl_index_test = j
    ha_temp = zero
    do i = 1, zn_num
      if (zn_type(i) == zn_plasma ∨ zn_type(i) == zn_vacuum) then
        index_parameters_tl_index_zone = i
        density = extract_output_datum(index_parameters, 1, out_post_all, o_mean,
          'neutral_density')
        zone_data_i = density // NEUTRAL DENSITY
        ha_temp += density * zn_volume(i)
      else
        zone_data_i = zero
      end if
    end do
    name_clean(sp_sy(pr_test(j)), clean_sy)
    call write_hdf('sp' || trim(clean_sy) || 'den.hdf', zone_data, 'sp' || trim(clean_sy) || '_density',
      'm**-3', 'E11.3', na, nb, nc, xa, xb, xc, pixel_zones, pixel_data)

    do i = 1, zn_num
      if (zn_type(i) == zn_plasma ∨ zn_type(i) == zn_vacuum) then
        index_parameters_tl_index_zone = i
        density = extract_output_datum(index_parameters, 1, output_all, o_var, 'neutral_density')
        zone_data_i = density // NEUTRAL DENSITY REL. STD. DEV.
      else
        zone_data_i = zero
      end if
    end do

    call write_hdf('sp' || trim(clean_sy) || 'den_rsd.hdf', zone_data,
      'sp' || trim(clean_sy) || '_density_rsd', '_', 'E11.3', na, nb, nc, xa, xb, xc,
      pixel_zones, pixel_data)

    if (tl_check(string_lookup('neutral_velocity_vector', tally_name, tl_num))) then
      vname = 'neutral_velocity_vector'
      vtag = 'vel'
    else
      assert(tl_check(string_lookup('neutral_flux_vector', tally_name, tl_num)))
      vname = 'neutral_flux_vector'
      vtag = 'flx'
    end if

    do i = 1, 3 // Component
      do zone = 1, zn_num
        if (zn_type(zone) == zn_plasma ∨ zn_type(zone) == zn_vacuum) then
          index_parameters_tl_index_zone = zone
          ha_rate = extract_output_datum(index_parameters, i, out_post_all, o_mean, trim(vname))
          zone_data_zone = ha_rate
        else
          zone_data_zone = zero
        end if
      end do
    end do

    call write_hdf('sp' || trim(clean_sy) || vtag || ind_sy_i || '.hdf', zone_data,
      trim(clean_sy) || vtag || ind_sy_i, 'm/s', 'E11.3', na, nb, nc, xa, xb, xc, pixel_zones,

```

```

        pixel_data)
    end do
    if (string_lookup('neutral_pressure', tally_name, tl_num) > 0) then
        do i = 1, zn_num
            if (zn_type(i) ≡ zn_plasma ∨ zn_type(i) ≡ zn_vacuum) then
                index_parameters_tl_index_zone = i
                pressure = extract_output_datum(index_parameters, 1, out_post_all, o_mean,
                    'neutral_pressure')
                zone_data_i = pressure / const(1.3332, -1) // NEUTRAL PRESSURE
            else
                zone_data_i = zero
            end if
        end do
        call write_hdf('sp' || trim(clean_sy) || 'prs.hdf', zone_data,
            'sp' || trim(clean_sy) || '_pressure', 'mTorr', 'E11.3', na, nb, nc, xa, xb, xc,
            pixel_zones, pixel_data)
    end if
end do
@#endif
@#if 0
    /* This is provisional. Could just test for the presence of the tally and leave this in here
    permanently. */
    j = 2 // Usually the atom species
    index_parameters_tl_index_test = j
    do k = 1, so_type_num + pr_reaction_num + pr_pmi_num
        index_parameters_tl_index_test_author = k
        ha_temp = zero
        do i = 1, zn_num
            if (zn_type(i) ≡ zn_plasma ∨ zn_type(i) ≡ zn_vacuum) then
                index_parameters_tl_index_zone = i
                density = extract_output_datum(index_parameters, 1, out_post_all, o_mean,
                    'neutral_density_by_author')
                zone_data_i = density // NEUTRAL DENSITY BY AUTHOR
                ha_temp += density * zn_volume(i)
            else
                zone_data_i = zero
            end if
        end do
        if (ha_temp > zero) then
            write(auth, '(a1,i2.2)') 'a', k
            call write_hdf('sp' || trim(sp_sy(pr_test(j))) || 'den_' || auth || '.hdf', zone_data,
                'sp' || trim(sp_sy(pr_test(j))) || '_density_' || auth, 'm**-3', 'E11.3', na, nb, nc,
                xa, xb, xc, pixel_zones, pixel_data)
        end if
    end do
@#endif
@#if 1
    do i = 1, zn_num
        zone_data_i = zero
    end do
    do test = 1, pr_test_num

```

```

if (((sp_sy(pr_test(test)) ≡ 'H') /* NEUTRAL H-ALPHA */
∨(sp_sy(pr_test(test)) ≡ 'D') ∨ (sp_sy(pr_test(test)) ≡
  'T')) ∧ string_lookup(trim(sp_sy(pr_test(test))) || 'alpha_emission_rate', tally_name,
  tl_num) > 0) then
  ha_temp = zero
  do zone = 1, zn_num
    if (zn_type(zone) ≡ zn_plasma) then
      index_parameterstl_index_zone = zone
      ha_rate = extract_output_datum(index_parameters,
        1, out_post_all, o_mean, trim(sp_sy(pr_test(test))) ||
        'alpha_emission_rate') / (const(1.8881944) * electron_charge)
      zone_datazone += ha_rate
      ha_temp += ha_rate * zn_volume(zone)
    @if 0
      else if (zn_type(zone) ≡ zn_solid) then // Can plot solid
        zone_datazone = -one // regions
    @endif
    else
      zone_datazone = zero
    end if
  end do
end if
halpha_tot = zero
do zone = 1, zn_num
  halpha_tot += zone_datazone * zn_volume(zone)
end do
if (halpha_tot > zero) then
  call write_hdf('halpha.hdf', zone_data, 'H_alpha_rate', 'photons/□(m**3□s)', 'E11.3',
    na, nb, nc, xa, xb, xc, pixel_zones, pixel_data)
end if
do i = 1, zn_num
  zone_datai = zero
end do
do test = 1, pr_test_num
  if (string_lookup('Dalpha_emission_rate_by_species', tally_name, tl_num) > 0) then
    name_clean(sp_sy(pr_test(test))), clean_sy)
    ha_temp = zero
    index_parameterstl_index_test = test
    do zone = 1, zn_num
      if (zn_type(zone) ≡ zn_plasma) then
        index_parameterstl_index_zone = zone
        ha_rate = extract_output_datum(index_parameters, 1, out_post_all, o_mean,
          'Dalpha_emission_rate_by_species') / (const(1.8881944) * electron_charge)
        zone_datazone = ha_rate
        ha_temp += ha_rate * zn_volume(zone)
      else
        zone_datazone = zero
      end if
    end do
  end if
end do
end if

```

```

    halpha_tot = zero
    do zone = 1, zn_num
        halpha_tot += zone_data_zone * zn_volume(zone)
    end do

    if (halpha_tot > zero) then
        call write_hdf('halpha' || trim(clean_sy) || '.hdf', zone_data,
            'H_alpha_rate_' || trim(clean_sy), 'photons_/_(m**3_s)', 'E11.3', na, nb, nc, xa,
            xb, xc, pixel_zones, pixel_data)
    end if
end do

@#endif
@#if 1
    do i = 1, zn_num
        zone_data_i = zero
    end do

    do test = 1, pr_test_num
        if (((sp_sy(pr_test(test)) ≡ 'He') /* NEUTRAL He 5877 line */
            ^string_lookup('He_5877_emission_rate', tally_name, tl_num) > 0)) then
            ha_temp = zero
            do zone = 1, zn_num
                if (zn_type(zone) ≡ zn_plasma) then
                    index_parameters_tl_index_zone = zone
                    ha_rate = extract_output_datum(index_parameters, 1, out_post_all, o_mean,
                        'He_5877_emission_rate') / (const(2.109566) * electron_charge)
                    zone_data_zone += ha_rate
                    ha_temp += ha_rate * zn_volume(zone)
                @#if 0
                else if (zn_type(zone) ≡ zn_solid) then // Can plot solid
                    zone_data_zone = -one // regions
                @#endif
                else
                    zone_data_zone = zero
                end if
            end do
        end if
    end do

    halpha_tot = zero
    do zone = 1, zn_num
        halpha_tot += zone_data_zone * zn_volume(zone)
    end do

    if (halpha_tot > zero) then
        call write_hdf('he_5877.hdf', zone_data, 'He_5877_rate', 'photons_/_(m**3_s)', 'E11.3',
            na, nb, nc, xa, xb, xc, pixel_zones, pixel_data)
    end if
@#endif
@#if 1
    if ((pr_reaction_num > 0) ^ (string_lookup('ion_source_rate', tally_name, tl_num) > 0)) then
        do back = 1, pr_background_num
            name_clean(sp_sy(pr_background(back)), clean_sy)
            if (trim(clean_sy) ≠ 'e') then // ION SOURCE RATE

```

```

index_parameterstl_index_problem_sp = pr_problem_sp_back(back)
ha_temp = zero
do zone = 1, zn_num
  if (zn_type(zone) ≡ zn_plasma) then
    index_parameterstl_index_zone = zone
    ha_rate = extract_output_datum(index_parameters, 1, out_post_all, o_mean,
      'ion_source_rate')
    zone_datazone = ha_rate / zn_volume(zone)
    ha_temp += ha_rate
  else
    zone_datazone = zero
  end if
end do
call write_hdf(trim(clean_sy) || 'ionizeV.hdf', zone_data,
  trim(clean_sy) || '_Ion_Source_rate', 'm**-3_s**-1', 'E11.3', na, nb, nc, xa, xb,
  xc, pixel_zones, pixel_data)

do zone = 1, zn_num
  if (zn_type(zone) ≡ zn_plasma) then
    index_parameterstl_index_zone = zone
    ha_rate = extract_output_datum(index_parameters, 1, output_all, o_var,
      'ion_source_rate')
    zone_datazone = ha_rate
  else
    zone_datazone = zero
  end if
end do
call write_hdf(trim(clean_sy) || 'ionize_rsd.hdf', zone_data,
  trim(clean_sy) || '_Ion_Source_rate_rsd', '_', 'E11.3', na, nb, nc, xa, xb, xc,
  pixel_zones, pixel_data)

end if
end do

do back = 1, pr_background_num
  name_clean(sp_sy(pr_background(back)), clean_sy)
  if (trim(clean_sy) ≠ 'e') then // MOM. SOURCE RATE
    index_parameterstl_index_problem_sp = pr_problem_sp_back(back)
    do i = 1, 3 // Component
      ha_temp = zero
      do zone = 1, zn_num
        if (zn_type(zone) ≡ zn_plasma) then
          index_parameterstl_index_zone = zone
          ha_rate = extract_output_datum(index_parameters, i, out_post_all, o_mean,
            'ion_momentum_source_vector')
          zone_datazone = ha_rate
          ha_temp += ha_rate
        else
          zone_datazone = zero
        end if
      end do
    end do
  call write_hdf(trim(clean_sy) || 'mom' || ind_syi || '.hdf', zone_data,
    trim(clean_sy) || '_Mom_' || ind_syi || '_Source_rate', 'N', 'E11.3', na, nb, nc,
    xa, xb, xc, pixel_zones, pixel_data)

```

```

end do
do i = 1, 3 // Component
do zone = 1, zn_num
if (zn_type(zone) ≡ zn_plasma) then
index_parameters_tl_index_zone = zone
ha_rate = extract_output_datum(index_parameters, i, output_all, o_var,
'ion_momentum_source_vector')
zone_data_zone = ha_rate
else
zone_data_zone = zero
end if
end do
call write_hdf(trim(clean_sy) || 'mom' || ind_sy_i || '_rsd.hdf', zone_data,
trim(clean_sy) || '_Mom_' || ind_sy_i || '_Source_rate_rsd', 'N', 'E11.3', na, nb,
nc, xa, xb, xc, pixel_zones, pixel_data)
end do
end if
end do
do back = 1, pr_background_num // ENERGY SOURCE RATE
name_clean(sp_sy(pr_background(back)), clean_sy)
index_parameters_tl_index_problem_sp = pr_problem_sp_back(back)
ha_temp = zero
do zone = 1, zn_num
if (zn_type(zone) ≡ zn_plasma) then
index_parameters_tl_index_zone = zone
ha_rate = extract_output_datum(index_parameters, 1, out_post_all, o_mean,
'ion_energy_source')
zone_data_zone = ha_rate
ha_temp += ha_rate
else
zone_data_zone = zero
end if
end do
call write_hdf(trim(clean_sy) || 'energy.hdf', zone_data,
trim(clean_sy) || '_Energy_Source_rate', 'W', 'E11.3', na, nb, nc, xa, xb, xc,
pixel_zones, pixel_data)
do zone = 1, zn_num
if (zn_type(zone) ≡ zn_plasma) then
index_parameters_tl_index_zone = zone
ha_rate = extract_output_datum(index_parameters, 1, output_all, o_var,
'ion_energy_source')
zone_data_zone = ha_rate
else
zone_data_zone = zero
end if
end do
call write_hdf(trim(clean_sy) || 'energy_rsd.hdf', zone_data,
trim(clean_sy) || '_Energy_Source_rate_rsd', 'W', 'E11.3', na, nb, nc, xa, xb, xc,
pixel_zones, pixel_data)
end do
end if /* Read in and plot external zone-based data. */

```

```

if (n_ext_file > 0) then
  do ifile = 1, n_ext_file
    do zone = 1, zn_num
      zone_data_zone = zero
    end do
    open(unit = diskin, file = ext_filenames_ifile, status = 'old', form = 'formatted',
      iostat = open_stat)
    if (open_stat ≡ 0) then
      assert(ifile < 100)
      write(iflab, '(i2.2)') ifile
      ext_var_name = 'ext_var' || iflab
      ext_var_units = '□'
      ext_var_format = 'E11.3'
    loop2: continue
    if (read_string(diskin, line, length)) then
      assert(length ≤ len(line))
      length = parse_string(line(:length))
      p = 0
      assert(next_token(line, beg, e, p))
      if (line(beg : e) ≡ 'name') then
        assert(next_token(line, beg, e, p))
        ext_var_name = line(beg : e)
      else if (line(beg : e) ≡ 'units') then
        assert(next_token(line, beg, e, p))
        ext_var_units = line(beg : e)
      else if (line(beg : e) ≡ 'format') then
        assert(next_token(line, beg, e, p))
        ext_var_format = line(beg : e)
      else
        zone = read_int_soft_fail(line(beg : e))
        if (zn_check(zone)) then
          assert(next_token(line, beg, e, p))
          zone_data_zone = read_real(line(beg : e))
        else
          write(stderr, *) 'Improper□zone□number□', zone, '□in□external□file□',
            ext_filenames_n_ext_file
          end if
        end if
        go to loop2
      end if
      close(unit = diskin)
      call write_hdf(trim(ext_var_name) || '.hdf', zone_data, trim(ext_var_name),
        trim(ext_var_units), trim(ext_var_format), na, nb, nc, xa, xb, xc, pixel_zones,
        pixel_data)
    else
      write(stderr, *) 'Cannot□open□external□file□', ext_filenames_n_ext_file,
        '□error□number□', open_stat
    end if
  end do
end if /* Plot EIRENE data if the appropriate files are present. Since we've only worked with
  2-D EIRENE files (in benchmark exercises), skip if the current pixel mesh is 3-D. */
if (clen ≡ zero) then

```

§1.2–§1.3 [#3–#4] **geomtesta**A program for generating a basic set of output suitable for use with external graphics package

```
        call plot_eirene_data(zone_data, na, nb, nc, xa, xb, xc, pixel_zones, pixel_data)
    end if

@#endif
    var_free(xa)
    var_free(xb)
    var_free(xc)
    var_free(pixel_zones)
    var_free(pixel_data)
    var_free(zone_data)
    var_free(ext_filenames)
@#if SILO
    ret = dbclose(dbfile)
    assert(ret ≠ -1)
@#endif
    return
end
```

See also sections 1.3, 1.4, and 1.5.

This code is used in section 1.1.

Read and plot EIRENE output files. To facilitate the benchmarking process, have set up this routine to read the *fort.32* and *fort.44* files output by EIRENE and then to put the data into the same HDF file format as used by the rest of *geomtesta*. Have passed down the pixel mesh and storage arrays so their creation doesn't need to be repeated.

⟨Functions and Subroutines 1.2⟩ +≡

```
subroutine plot_eirene_data(zone_data, na, nb, nc, xa, xb, xc, pixel_zones, pixel_data)
    implicit_none_f77
    zn_common
    implicit_none_f90

    integer na, nb, nc    // Input
    integer pixel_zones0:na*nb*nc-1
    real zone_data_zn_num, xa0:na-1, xb0:nb-1, xc0:nc-1, pixel_data0:na*nb*nc-1
    integer nx, ny, natmi, nmoli, nioni, nfl    // Local

    ⟨Memory allocation interface 0⟩

    open(unit = diskin, file = 'eirene44.out', status = 'old', form = 'formatted', err = eof)
    read(diskin, '(i4,2x,i4)') nx, ny

    read(diskin, '(i4,2x,i4,2x,i4)') natmi, nmoli, nioni
    nfl = 1    // No way to determine this from the files?

    call plot_eirene_continued(nx, ny, natmi, nmoli, nioni, nfl, diskin, zone_data, na, nb, nc, xa,
        xb, xc, pixel_zones, pixel_data)

eof: continue

    return
end
```

Continue set up for reading and plotting of EIRENE data. Need to start a new subroutine here so that the pointer arrays can be properly dimensioned.

⟨Functions and Subroutines 1.2⟩ +=

```

subroutine plot_eirene_continued(nx, ny, natmi, nmoli, nioni, nfl, unit_num, zone_data, na, nb,
    nc, xa, xb, xc, pixel_zones, pixel_data)

    define_dimen(ix_ind, nx)
    define_dimen(iy_ind, ny)
    define_dimen(atm_ind, natmi)
    define_dimen(mol_ind, nmoli)
    define_dimen(ion_ind, nioni)
    define_dimen(fl_ind, nfl)

    define_varp(ix_iy_zone, INT, ix_ind, iy_ind)
    define_varp(atm_name, CHAR, species_symbol_string, atm_ind)
    define_varp(mol_name, CHAR, species_symbol_string, mol_ind)
    define_varp(ion_name, CHAR, species_symbol_string, ion_ind)
    define_varp(fl_name, CHAR, species_symbol_string, fl_ind)

    implicit_none_f77
    zn_common // Common
    so_common
    implicit_none_f90

    integer nx, ny, natmi, nmoli, nioni, nfl, unit_num, /* Input */
        na, nb, nc
    integer pixel_zones0:na*nb*nc-1
    real zone_datazn_num, xa0:na-1, xb0:nb-1, xc0:nc-1, pixel_data0:na*nb*nc-1

    integer i, length, zone, ix, iy, nstra // Local
    real multiplier
    character*sp_sy_len stra_name // Same length as other strings
    character*80 line

    declare_varp(ix_iy_zone)
    declare_varp(atm_name)
    declare_varp(mol_name)
    declare_varp(ion_name)
    declare_varp(fl_name)

    ⟨Memory allocation interface 0⟩
    st_decls

    var_alloc(ix_iy_zone)

    do zone = 1, zn_num
        if ((1 ≤ zn_index(zone, 1) ∧ zn_index(zone, 1) ≤ nx) ∧ (1 ≤ zn_index(zone, 2) ∧ zn_index(zone,
            2) ≤ ny)) then
            ix_iy_zonezn_index(zone, 1), zn_index(zone, 2) = zone
        end if
    end do

    do iy = 1, ny
        do ix = 1, nx
            assert(ix_iy_zoneix, iy > 0)
        end do
    end do

```

```

var_alloc(atm_name)
var_alloc(mol_name)
var_alloc(ion_name)

do i = 1, natmi
  read(unit_num, *) atm_name_i
end do
do i = 1, nmoli
  read(unit_num, *) mol_name_i
end do
do i = 1, nioni
  read(unit_num, *) ion_name_i
end do

/* This list should correspond with that in EIRENE's subroutine wneutral. */
call eirene_file_to_hdf(unit_num, nx, ny, natmi, ix_iy_zone, one, 'eir_den_', atm_name,
  'density', 'm**-3', 'E11.3', zone_data, na, nb, nc, xa, xb, xc, pixel_zones, pixel_data)
call eirene_file_to_hdf(unit_num, nx, ny, natmi, ix_iy_zone, one / electron_charge, 'eir_temp_',
  atm_name, 'temperature', 'eV', 'E11.3', zone_data, na, nb, nc, xa, xb, xc, pixel_zones,
  pixel_data)
call eirene_file_to_hdf(unit_num, nx, ny, nmoli, ix_iy_zone, one, 'eir_den_', mol_name,
  'density', 'm**-3', 'E11.3', zone_data, na, nb, nc, xa, xb, xc, pixel_zones, pixel_data)
call eirene_file_to_hdf(unit_num, nx, ny, nmoli, ix_iy_zone, one / electron_charge, 'eir_temp_',
  mol_name, 'temperature', 'eV', 'E11.3', zone_data, na, nb, nc, xa, xb, xc, pixel_zones,
  pixel_data)
call eirene_file_to_hdf(unit_num, nx, ny, nioni, ix_iy_zone, one, 'eir_den_', ion_name,
  'density', 'm**-3', 'E11.3', zone_data, na, nb, nc, xa, xb, xc, pixel_zones, pixel_data)
call eirene_file_to_hdf(unit_num, nx, ny, nioni, ix_iy_zone, one / electron_charge, 'eir_temp_',
  ion_name, 'temperature', 'eV', 'E11.3', zone_data, na, nb, nc, xa, xb, xc, pixel_zones,
  pixel_data)

close(unit = unit_num)

open(unit = unit_num, file = 'eirene32.out', status = 'old', form = 'formatted')

nstra = 0
var_alloc(fl_name)
do i = 1, nfl
  write(fl_name_i, '(a1,i2.2)') 'f', i
end do

loop: continue
if (read_string(unit_num, line, length)) then
  nstra++
  write(stra_name, '(a1,i2.2)') 's', nstra /* EIRENE leaves plate sources per unit flux;
  others are already scaled. See subroutine infcop. */
  if (so_type(nstra) == so_recomb) then
    multiplier = one / electron_charge
  else
    multiplier = so_tot_curr(nstra)
  end if /* This list should correspond to the one in EIRENE's infcop. */
  call eirene_file_to_hdf(unit_num, nx, ny, nfl, ix_iy_zone, multiplier,
    'eir_sni_' || trim(stra_name), fl_name, 'Ion_source_rate_' || trim(stra_name),
    's**-1', 'E11.3', zone_data, na, nb, nc, xa, xb, xc, pixel_zones, pixel_data)
  call eirene_file_to_hdf(unit_num, nx, ny, nfl, ix_iy_zone, multiplier,
    'eir_smo_' || trim(stra_name), fl_name, 'Momentum_source_rate_' || trim(stra_name),

```

```
      'N', 'E11.3', zone_data, na, nb, nc, xa, xb, xc, pixel_zones, pixel_data)
  call eirene_file_to_hdf(unit_num, nx, ny, 1, ix_iy_zone,
    multiplier, 'eir_see_' || trim(stra_name), 'tot', 'Electron_energy_source_ra\
    te_' || trim(stra_name), 'W', 'E11.3', zone_data, na, nb, nc, xa, xb, xc, pixel_zones,
    pixel_data)
  call eirene_file_to_hdf(unit_num, nx, ny, 1, ix_iy_zone, multiplier,
    'eir_sei_' || trim(stra_name), 'tot', 'Ion_energy_source_rate_' || trim(stra_name),
    'W', 'E11.3', zone_data, na, nb, nc, xa, xb, xc, pixel_zones, pixel_data)
  go to loop
end if

close(unit = unit_num)

var_free(ix_iy_zone)
var_free(atm_name)
var_free(mol_name)
var_free(ion_name)
var_free(fl_name)

assert(nstra == so_grps) // Assuming strata ↔ groups!

return
end
```

Read a section of an EIRENE output file and dump the contents to an HDF file. The “read” section should look a great deal like EIRENE’s subroutine *neutr*.

⟨Functions and Subroutines 1.2⟩ +≡

```
subroutine eirene_file_to_hdf(unit_num, nx, ny, nf, ix_izone, multiplier, cfilrt, cfilext, clabel,
    cunits, cformt, zone_data, na, nb, nc, xa, xb, xc, pixel_zones, pixel_data)
```

```
    implicit none f77
```

```
    zn_common // Common
```

```
    implicit none f90
```

```
    integer unit_num, nx, ny, nf, na, nb, nc // Input
```

```
    integer ix_izonenx, ny, pixel_zones0:na*nb*nc-1
```

```
    real multiplier
```

```
    real zone_datazn_num, xa0:na-1, xb0:nb-1, xc0:nc-1, pixel_data0:na*nb*nc-1
```

```
    character*(*) cfilrt, clabel, cunits, cformt
```

```
    character*(*) cfilextnf
```

```
    character*sp_sy_len clean_sy
```

```
    integer lim, i, j, ix, iy, zone, ind_tmp // Local
```

```
⟨Memory allocation interface 0⟩
```

```
st_decls
```

```
    do zone = 1, zn_num
```

```
        zone_datazone = zero
```

```
    end do
```

```
    lim = (nx / 5) * 5 - 4
```

```
    do i = 1, nf
```

```
        do iy = 1, ny
```

```
            do ix = 1, lim, 5
```

```
                read (unit_num, '(5(e16.8))') SP(zone_dataix_izoneix-1+j, iy, j = 1, 5)
```

```
            end do
```

```
            if (lim + 4 ≠ nx) then
```

```
                read (unit_num, '(5(e16.8))') SP(zone_dataix_izoneix, iy, ix = lim + 5, nx)
```

```
            end if
```

```
        end do
```

```
    if (multiplier ≠ one) then
```

```
        do zone = 1, zn_num
```

```
            zone_datazone *= multiplier
```

```
        end do
```

```
    end if
```

```
    name_clean(cfilexti, clean_sy)
```

```
    call write_hdf(cfilrt || trim(clean_sy) || '.hdf', zone_data, trim(cfilexti) || '□' || clabel, cunits,
        cformt, na, nb, nc, xa, xb, xc, pixel_zones, pixel_data)
```

```
    end do
```

```
    return
```

```
end
```

2 INDEX

- a*: [1.2](#).
a_ind: [1.2](#).
abs: [1.2](#).
alen: [1.2](#).
areal: [1.2](#).
assert: [1.2](#), [1.4](#).
atm_ind: [1.4](#).
atm_name: [1.4](#).
auth: [1.2](#).
- b*: [1.2](#).
b_ind: [1.2](#).
back: [1.2](#).
beg: [1.2](#).
bk_common: [1.2](#).
bk_n: [1.2](#).
bk_temp: [1.2](#).
bk_v: [1.2](#).
blen: [1.2](#).
- c*: [1.2](#).
c_ind: [1.2](#).
cfilext: [1.5](#).
cfilrt: [1.5](#).
cformat: [1.5](#).
CHAR: [1.2](#), [1.4](#).
clabel: [1.5](#).
clean_sy: [1.2](#), [1.5](#).
clen: [1.2](#).
clenp: [1.2](#).
const: [1.2](#).
cunits: [1.5](#).
- DB_CLOBBER*: [1.2](#).
DB_F77NULL: [1.2](#).
DB_LOCAL: [1.2](#).
dbclose: [1.2](#).
dbcreate: [1.2](#).
dbfile: [1.2](#).
de_common: [1.2](#).
de_zone_frags: [1.2](#).
declare_varp: [1.2](#), [1.4](#).
define_dimen: [1.2](#), [1.4](#).
define_varp: [1.2](#), [1.4](#).
degas_init: [1.1](#).
density: [1.2](#).
diskin: [1.2](#), [1.3](#).
dx: [1.2](#).
dz: [1.2](#).
- e*: [1.2](#).
eirene_file_to_hdf: [1.4](#), [1.5](#).
electron_charge: [1.2](#), [1.4](#).
eof: [1.3](#).
err: [1.3](#).
exercise: [1.1](#), [1.2](#).
ext_file_ind: [1.2](#).
ext_filenames: [1.2](#).
ext_var_format: [1.2](#).
ext_var_n: [1](#).
ext_var_name: [1.2](#).
ext_var_units: [1.2](#).
extract_output_datum: [1.2](#).
- file*: [1.2](#), [1.3](#), [1.4](#).
FILE: [1](#).
FILELEN: [1.2](#).
filenames_array: [1.2](#).
filenames_size: [1.2](#).
fl_ind: [1.4](#).
fl_name: [1.4](#).
FLOAT: [1.2](#).
form: [1.2](#), [1.3](#), [1.4](#).
fort: [1.3](#).
- geometry*: [1](#).
geometry_test: [1.1](#).
geomtesta: [1.3](#).
get_pixel_mesh: [1.2](#).
GRAPH_FILE: [1](#).
- ha_rate*: [1.2](#).
ha_temp: [1.2](#).
half: [1.2](#).
halpha_tot: [1.2](#).
HDF4: [1](#).
- i*: [1.2](#), [1.4](#), [1.5](#).
ifile: [1.2](#).
iflab: [1.2](#).
implicit_none_f77: [1.1](#), [1.2](#), [1.3](#), [1.4](#), [1.5](#).
implicit_none_f90: [1.1](#), [1.2](#), [1.3](#), [1.4](#), [1.5](#).
include: [1.2](#).
ind_sy: [1.2](#).
ind_tmp: [1.2](#), [1.5](#).
index: [1.2](#).
index_parameters: [1.2](#).
infcop: [1.4](#).
inp: [1](#).
INT: [1.2](#), [1.4](#).
ion_ind: [1.4](#).
ion_name: [1.4](#).
iostat: [1.2](#).
iview: [1.2](#).
ivlab: [1.2](#).

ix: [1.4](#), [1.5](#).
ix_ind: [1.4](#).
ix-iy-zone: [1.4](#), [1.5](#).
iy: [1.4](#), [1.5](#).
iy_ind: [1.4](#).
j: [1.2](#), [1.5](#).
k: [1.2](#).
len: [1.2](#).
length: [1.2](#), [1.4](#).
lim: [1.5](#).
line: [1.2](#), [1.4](#).
LINELEN: [1.2](#).
local: [1](#).
loop: [1.2](#), [1.4](#).
loop2: [1.2](#).
main_pixsize: [1.2](#).
main_z_scale: [1.2](#).
Makefile: [1](#).
max: [1.2](#).
max_v: [1.2](#).
mesh_name: [1.2](#).
mod: [1.2](#).
mol_ind: [1.4](#).
mol_name: [1.4](#).
multiplier: [1.4](#), [1.5](#).
n_ext_file: [1.2](#).
na: [1.2](#), [1.3](#), [1.4](#), [1.5](#).
name_clean: [1.2](#), [1.5](#).
natmi: [1.3](#), [1.4](#).
nb: [1.2](#), [1.3](#), [1.4](#), [1.5](#).
nc: [1](#), [1.2](#), [1.3](#), [1.4](#), [1.5](#).
nc_read_output: [1.1](#).
neutr: [1.5](#).
next_token: [1.2](#).
nf: [1.5](#).
nfl: [1.3](#), [1.4](#).
nioni: [1.3](#), [1.4](#).
nmoli: [1.3](#), [1.4](#).
nstra: [1.4](#).
nx: [1.3](#), [1.4](#), [1.5](#).
ny: [1.3](#), [1.4](#), [1.5](#).
o_mean: [1.2](#).
o_var: [1.2](#).
one: [1.2](#), [1.4](#), [1.5](#).
open_stat: [1.2](#).
ou_common: [1.2](#).
out_post_all: [1.2](#).
output_all: [1.2](#).
output_old_file: [1.2](#).
outputfile: [1.2](#).
p: [1.2](#).
parse_string: [1.2](#).
pixel_data: [1.2](#), [1.3](#), [1.4](#), [1.5](#).
pixel_ind: [1.2](#).
pixel_zones: [1.2](#), [1.3](#), [1.4](#), [1.5](#).
pixsize: [1.2](#).
plot_eirene_continued: [1.3](#), [1.4](#).
plot_eirene_data: [1.2](#), [1.3](#).
pr_background: [1.2](#).
pr_background_num: [1.2](#).
pr_common: [1.2](#).
pr_pmi_num: [1.2](#).
pr_problem_sp_back: [1.2](#).
pr_reaction_num: [1.2](#).
pr_test: [1.2](#).
pr_test_num: [1.2](#).
pressure: [1.2](#).
read_int_soft_fail: [1.2](#).
read_integer: [1.2](#).
read_real: [1.2](#).
read_string: [1.2](#), [1.4](#).
readfilenames: [1.1](#).
ret: [1.2](#).
rf_common: [1.2](#).
set_pixel_zones: [1.2](#).
silo: [1](#).
SILO: [1](#), [1.2](#).
silo_common: [1.2](#).
silo_file: [1.2](#).
silo_format: [1.2](#).
SILO-HDF5: [1](#).
so_common: [1.4](#).
so-grps: [1.4](#).
so-recomb: [1.4](#).
so_tot_curr: [1.4](#).
so_type: [1.4](#).
so_type_num: [1.2](#).
sp: [1.2](#).
SP: [1.2](#), [1.5](#).
sp_common: [1.2](#).
sp_sy: [1.2](#).
sp_sy_len: [1.2](#), [1.4](#), [1.5](#).
species_symbol_string: [1.4](#).
st_decls: [1.2](#), [1.4](#), [1.5](#).
status: [1.2](#), [1.3](#), [1.4](#).
stderr: [1.2](#).
stra_name: [1.4](#).
string_length: [1.2](#).
string_lookup: [1.2](#).

tally_name: 1.2.
test: 1.2.
tl_check: 1.2.
tl_common: 1.2.
tl_decls: 1.2.
tl_index_max: 1.2.
tl_index_problem_sp: 1.2.
tl_index_test: 1.2.
tl_index_test_author: 1.2.
tl_index_zone: 1.2.
tl_num: 1.2.
tl_tag_length: 1.2.
trim: 1.2, 1.4, 1.5.
TRUE: 1.2.
two: 1.2.

unit: 1.2, 1.3, 1.4.
unit_num: 1.4, 1.5.

var_alloc: 1.2, 1.4.
var_free: 1.2, 1.4.
var_realloca: 1.2.
vc_copy: 1.2.
vc_decls: 1.2.
vc_unit: 1.2.
vc_xvt: 1.2.
vname: 1.2.
vtag: 1.2.
vz: 1.2.

wneutral: 1.4.
write_hdf: 1.2, 1.5.
write_mesh: 1.2.

x: 1.2.
xa: 1.2, 1.3, 1.4, 1.5.
xb: 1.2, 1.3, 1.4, 1.5.
xc: 1.2, 1.3, 1.4, 1.5.
xp: 1.2.

z_scale: 1, 1.2.
zero: 1.2, 1.5.
zn_check: 1.2.
zn_common: 1.2, 1.3, 1.4, 1.5.
zn_index: 1.4.
zn_num: 1, 1.2, 1.3, 1.4, 1.5.
zn_plasma: 1.2.
zn_solid: 1.2.
zn_type: 1.2.
zn_vacuum: 1.2.
zn_volume: 1.2.
zone: 1, 1.2, 1.4, 1.5.
zone_data: 1.2, 1.3, 1.4, 1.5.
zone_ind: 1.2.

⟨Functions and Subroutines 1.2, 1.3, 1.4, 1.5⟩ Used in section 1.1.

⟨Memory allocation interface 0⟩ Used in sections 1.5, 1.4, 1.3, and 1.2.

COMMAND LINE: "fweave -f -i! -W[-ykw700 -ytw40000 -j -n/
/u/dstotler/degas2/src/geomtesta.web".

WEB FILE: "/u/dstotler/degas2/src/geomtesta.web".

CHANGE FILE: (none).

GLOBAL LANGUAGE: FORTRAN.