pdsol reference manual

Özgür D. Gürcan
Laboratoire de Physique des Plasmas, Ecole Polytechnique, CNRS, 91128, Palaiseau, France

February 9, 2010

Contents

1 pdsol

1.1 Introduction

pdsol is a minimalistic partial differential equation solver written in C. It is based on the gnu scientific library (GSL) which uses xml for defining the problem and solver parameters, and hdf as the input/output data format. It runs on linux/unix or in fact any other environment where the dependencies can be satisfied.

1.1.1 How to Compile

pdsol uses linux standard autoconf/automake system. Thus, it can be compiled simply by typing

```
./configure
make
```

Try

```
./configure --help
```

for more information and details. In particular --enable-debug option turns of compiler optimizations and permits debugging. Note that the configure script will naturally fail if the required dependencies are not satisfied.

1.1.2 Requirements

pdsol uses various open source libraries and as a result is dependent on them. This means the following libraries has to be installed in order to compile and rund pdsol:

- Gnu Scientific Library `http://www.gnu.org/software/gsl/`
- Mini-XML Library `http://www.minixml.org`
- Hierarchical Data Format (HDF5) Library `http://www.hdfgroup.org/HDF5/`
- Octave `http://www.gnu.org/software/octave/` is needed for initialization and data analysis (Matlab may be used as well).

1.1.3 How to Run an Example

The example problems are found in the "examples" directory of the pdsol library. These examples can be run in a simple way, and their parameters can be changed easily without the need for recompiling the library or the source code of the example problem.

In general, when the code is compiled, all the example files will be compiled automatically.

each example file contains three essential files.

- The "init.xml" file where the problem and integration parameters are defined.
- The "init_fields.m" file, which is an octave file that defines the initial condition.
- The "<problem_name>.c" file where the problem itself is defined.

The task of a programmer who wants to write a solver that solves a new system of partial differential equations is to write these three files.

here are some examples:

- The Heat Equation
- Angular Momentum Transport Module

2 The Heat Equation

Let us consider the simplest example, the heat equation:

$$\frac{\partial T}{\partial t} - \frac{\partial^2 T}{\partial x^2} = 0;$$

It is found in the directory "examples/heateqn". In this directory, you can find three files that define the problem.

- The init.xml file where the problem and integration parameters are defined.

```
<?xml version="1.0"?>
<solver nfields="1" naux="1" dt="1e-8" dtout="0.01" tmax="0.1" name="heateqn">

  <gsolver
    method="rkf45"
    step="adaptive"
    eps_abs="1e-6"
    eps_rel="1e-8">
  </gsolver>

  <field
    index="1"
    name="T"
    init="T_init.h5">
  </field>

  <mesh
    x0="0.0"
    dx="0.005"
    nx="201">
  </mesh>

  <params
    D="1.0">
  </params>

</solver>
```

- The "init_fields.m" file, which is an octave file that defines the initial condition.

```
dx=0.005;
xx=0.0:dx:1.0;
T=exp(-(xx-0.5).^2/0.01);
save('-hdf5',"T_init.h5","T");
```

- The "heat.c" file where the equation itself is defined.

```
#include "mesh.h"
#include "solver.h"
#include <stdio.h>
#include <mxml.h>
#include <gsl/gsl_errno.h>
#include <gsl/gsl_odeiv.h>

int fcal_diffn(double t, const double y[], double dydt[], void *params){
  pdsol_rsolver* sv = (pdsol_rsolver *)params;
  pdsol_mesh *rm = sv->rm;
  const double *T;
  double dxp,dxm,D;
  double *DTdt;
  double *S;
  int l;
  T=y;
  DTdt=dydt;
  D=0.1;//sv->params[0].val;
  for(l=1;l<rm->nx-1;l++){
    dxp=rm->xx[l+1]-rm->xx[l];
    dxm=rm->xx[l]-rm->xx[l-1];
    DTdt[l]=((T[l+1]-T[l])/dxp-(T[l]-T[l-1])/dxm)/((dxp+dxm)/2);
  }
  DTdt[0]=0;
  DTdt[rm->nx-1]=0;
  return GSL_SUCCESS;
}

int main (void){
  double t=0;
  pdsol_rsolver *sv;
  int l;
  sv=pdsol_init_rsolver("init.xml");
  sv->gsv->sys->function=fcal_diffn;
  pdsol_gslrun(sv);
  return 0;
}
```

If you want to simply run this example by changing some parameters. You can do so by typing:

```
octave init_fields.m
./heat
```

if the source code (heat.c) is modified one needs to re-make it by calling:

```
make clean;
make
```

3 Angular Momentum Transport Module

The angular momentum transport module that incorporates in particular the effect of $E \times B$ shear driven residual stress, is written as an example program to the pdsol library. it can be found in the examples/angular directory.

The system of equations that are being solved can be written as:

$$\frac{\partial n}{\partial t}+\frac{1}{r}\frac{\partial}{\partial r}\left(r\Gamma_n\right)=S_n \tag{1a}$$
$$\frac{\partial P}{\partial t}+\frac{1}{r}\frac{\partial}{\partial r}\left(rQ\right)=H \tag{1b}$$
$$\frac{\partial L_\phi}{\partial t}+\frac{1}{r}\frac{\partial}{\partial r}\left(r\Pi_{r,\phi}\right)=\tau_\phi\ . \tag{1c}$$

Here *n* is the plasma density, *P* is the pressure and L_ϕ is the angular momentum density. The fluxes are defined as:

$$\Gamma_n=-D_0\frac{\partial n}{\partial r}-D_1\varepsilon\left(\frac{\partial n}{\partial r}+V_{rn}n\right) \tag{2a}$$
$$\Pi_{r,\phi}=-\nu_0\frac{\partial L_\phi}{\partial r}-\nu_1\varepsilon\left[\frac{\partial L_\phi}{\partial r}+V_{rL}L_\phi\right]+S \tag{2b}$$
$$\text{where}\quad S=-\varepsilon\alpha\left(r\right)n\left(r\right)\left(1-\frac{\sigma}{P_0}\frac{\partial P}{\partial r}\right)\frac{\partial v_{Ey}}{\partial r} \tag{2c}$$
$$Q=-\chi_0\frac{\partial P}{\partial r}-\chi_1\varepsilon\frac{\partial P}{\partial r}\ . \tag{2d}$$

We use a reduced force balance equation, a simple formula for turbulence reduction and relate the flux surface average of the flow to angular momentum:

$$\frac{\partial v_{Ey}}{\partial r} = \left(\frac{\rho_*}{n_0 P_0}\right) \frac{\partial n}{\partial r}\frac{\partial P}{\partial r} + \cdots \tag{3a}$$

$$\varepsilon = \frac{\varepsilon_0}{\left[1 + \beta \left(\partial v_{Ey}/\partial r\right)^2\right]} \tag{3b}$$

$$v_\phi \left(r,t\right) = \left[\frac{L_\phi \left(r,t\right)}{n \left(r,t\right)}\right] \lambda_2 \left(r\right) \ . \tag{3c}$$

We also impose the boundary conditions:

$$L_\phi(a) = v_\phi(a) = P(a) = 0$$
$$n(a) = n_a \ .$$

The description of the problem requires 3 evolving fields. 3 auxillary fields are used to store the fluxes. One extra output field is used to output the velocity field (instead of angular momentum). The model can be found in the directory examples/angular. In this directory three files are found:

- The init.xml file where the problem and integration parameters are defined.

```
<?xml version="1.0"?>
<solver nfields="3" naux="3" nexout="1" dt="1e-8" dtout="0.1" tmax="1.0" name="an
      gular">

  <gsolver
    method="rkf45"
    step="adaptive"
    eps_abs="1e-6"
    eps_rel="1e-8">
  </gsolver>

  <field
    index="1"
    name="P"
    init="P_init.h5">
  </field>

  <field
    index="2"
    name="n"
    init="n_init.h5">
  </field>

  <field
    index="3"
    name="L"
    init="L_init.h5">
  </field>

  <exout
    index="1"
    name="vphi">
  </exout>

  <mesh
    x0="0.0"
    dx="0.005"
    nx="201">
  </mesh>

  <params
    D0="1.0"
    D1="4.0"
    chi0="1.0"
    chi1="4.0"
    nu0="1.0"
    nu1="4.0"
    alpha="2.0"
    rhostar="0.1"
    sigma="-0.1"
    gama_a="3.3"
    q_a="3.3"
    eta="10.0"
    eps0="1.0"
    lambda="2.0"
    Vrn="0.0"
    VrL="0.0"
    R="3.0">
  </params>

</solver>
```

- The "init_fields.m" file, which is an octave file that defines the initial condition.

```
dx=0.005;
xx=0.0:dx:1.0;
P=1.5*(tanh(100*(-xx+0.9))+1.0);
save('-hdf5',"P_init.h5","P");
n=0.5*(tanh(100*(-xx+0.9))+1.0)+0.05;
save('-hdf5',"n_init.h5","n");
L=0.0*(tanh(100*(-xx+0.9))+1.0);
save('-hdf5',"L_init.h5","L");
```

- The "angular.c" file where the equation itself is defined.

```
#include "mesh.h"
#include "solver.h"
#include <stdio.h>
#include <mxml.h>
#include <math.h>
#include <string.h>
#include "pdutils.h"
#include <gsl/gsl_errno.h>
#include <gsl/gsl_odeiv.h>

struct pars {
  double D0,D1,chi0,chi1,nu0,nu1,alpha,rhostar,sigma,gama_a,q_a,eta,eps0,lambda,V
      rn,VrL,R;
};

int fcal_angular(double t, const double y[], double dydt[], void *params){
  pdsol_rsolver* sv = (pdsol_rsolver *)params;
  pdsol_mesh *rm = sv->rm;
  const double *P, *n,*L;
  double *dPdt,*dndt,*dLdt;
  double dxp,dxm,dxc;
  double *Sn,*SP,*SL;
  double Pl,nl,Ll,dPldx,dnldx,dLldx,dvEydx,r,n_a,g_a,vareps;
  double alpar,lambda2;
  struct pars *ps=(struct pars *)sv->user;
  int l;

  /*setting up the fields*/
  P=&y[0];
  dPdt=&dydt[0];

  n=&y[rm->nx];
  dndt=&dydt[rm->nx];
```

```
L=&y[2*rm->nx];
dLdt=&dydt[2*rm->nx];

SP=&sv->aux[0];
Sn=&sv->aux[rm->nx];
SL=&sv->aux[2*rm->nx];

n_a=(n[rm->nx-1]+n[rm->nx-2])/2; // note: maybe we should just use n[rm->nx-1]
    ?
g_a=(n[rm->nx-2]-n[rm->nx-1])/(rm->xx[rm->nx-1]-rm->xx[rm->nx-2]); // -dn/dx
/*computing the fluxes */

for(l=0;l<rm->nx-1;l++){

  r=(rm->xx[l+1]+rm->xx[l])/2;
  alpar=ps->alpha*(1+r*r/ps->R/ps->R);
  lambda2=(1-r*r/ps->R/ps->R);
  /*Spatial stepsizes:*/
  dxp=rm->xx[l+1]-rm->xx[l];

  sv->exout[0]->dat[l]=L[l]*lambda2/n[l];

  Pl=(P[l+1]+P[l])/2;
  nl=(n[l+1]+n[l])/2;
  Ll=(L[l+1]+L[l])/2;

  dPldx=(P[l+1]-P[l])/dxp;
  dnldx=(n[l+1]-n[l])/dxp;
  dLldx=(L[l+1]-L[l])/dxp;

  /*ExB shear:*/
  dvEydx=ps->rhostar*dPldx*dnldx/(Pl*nl);

  vareps=ps->eps0/(1+ps->lambda*pow(dvEydx,2));

  /* Fluxes:*/
  /* Note: S[l] is the flux at l+1/2 */
  SP[l]=-ps->chi0*dPldx-ps->chi1*vareps*dPldx-2*ps->q_a*r*(1-r*r/2);
  Sn[l]=-ps->D0*dnldx-ps->D1*vareps*dnldx+ps->D1*vareps*ps->Vrn*nl
    -ps->gama_a*exp(-ps->eta*(n_a*(1.0-r)+g_a*(1.0-r)*(1.0-r)/2.0));
  SL[l]=-ps->nu0*dLldx-ps->nu1*vareps*dLldx+ps->nu1*vareps*ps->VrL*Ll
    -alpar*vareps*nl*(1-ps->sigma*dPldx/Pl)*dvEydx;
}
SP[rm->nx-1]=SP[rm->nx-2];
Sn[rm->nx-1]=Sn[rm->nx-2];
SL[rm->nx-1]=SL[rm->nx-2];
//SP[rm->nx-1]=0;
//Sn[rm->nx-1]=0;

/* the actual time step */

for(l=1;l<rm->nx-1;l++){
  dxp=rm->xx[l+1]-rm->xx[l];
  dxm=rm->xx[l]-rm->xx[l-1];
  dxc=(dxp+dxm)/2;

  /*Since S[l] is at l+1/2 and S[l-1] at l-1/2, we write the equations as:*/
  dPdt[l]=-(SP[l]*(rm->xx[l+1]+rm->xx[l])/2-SP[l-1]*(rm->xx[l]+rm->xx[l-1])/2)/
    dxc/rm->xx[l];
  dndt[l]=-(Sn[l]*(rm->xx[l+1]+rm->xx[l])/2-Sn[l-1]*(rm->xx[l]+rm->xx[l-1])/2)/
    dxc/rm->xx[l];
  dLdt[l]=-(SL[l]*(rm->xx[l+1]+rm->xx[l])/2-SL[l-1]*(rm->xx[l]+rm->xx[l-1])/2)/
    dxc/rm->xx[l];
  //    printf("dndt[%i]=%f\n",l,dndt[l]);
}
```

```
  /* the boundary conditions */

  dPdt[0]=dPdt[1];
  dPdt[rm->nx-1]=0;

  dndt[0]=dndt[1];
  dndt[rm->nx-1]=0;

  dLdt[0]=dLdt[1];
  dLdt[rm->nx-1]=0;

  if(L[2]!=L[2]){
    printf("Nan in L!");
    exit(-1);
  }
  return GSL_SUCCESS;
}

struct pars *get_params(pdsol_rsolver *sv){
  struct pars *ps;
  ps=malloc(sizeof(struct pars));
  ps->D0=pdget_par(sv,"D0");
  ps->D1=pdget_par(sv,"D1");
  ps->chi0=pdget_par(sv,"chi0");
  ps->chi1=pdget_par(sv,"chi1");
  ps->nu0=pdget_par(sv,"nu0");
  ps->nu1=pdget_par(sv,"nu1");
  ps->alpha=pdget_par(sv,"alpha");
  ps->rhostar=pdget_par(sv,"rhostar");
  ps->sigma=pdget_par(sv,"sigma");
  ps->gama_a=pdget_par(sv,"gama_a");
  ps->q_a=pdget_par(sv,"q_a");
  ps->eta=pdget_par(sv,"eta");
  ps->eps0=pdget_par(sv,"eps0");
  ps->lambda=pdget_par(sv,"lambda");
  ps->Vrn=pdget_par(sv,"Vrn");
  ps->VrL=pdget_par(sv,"VrL");
  ps->R=pdget_par(sv,"R");
  return ps;
}

int main (int argc, char *argv[]){
  double t=0;
  char *confname;
  pdsol_rsolver *sv;
  struct pars *ps;
  int l;
  if (argc==1){
    confname=strdup("init.xml");
  }
  else if (argc==2){
    if (!strcmp(argv[1],"-h")){
      printf("usage: ./angular <config.xml>\n");
      exit(-1);
    }
    confname=strdup(argv[1]);
  }
  else{
    printf("usage: ./angular <config.xml>\n");
    exit(-1);
  }
  sv=pdsol_init_rsolver(confname);
  //  sv->exout[0]->dat=&sv->aux[sv->rm->nx];
  sv->exout[0]->dat=malloc(sizeof(double)*sv->rm->nx);
  memset(sv->exout[0]->dat,0,sizeof(double)*sv->rm->nx);
  sv->gsv->sys->function=fcal_angular;
  ps=get_params(sv);
```

```
    sv->user=ps;
    pdsol_gslrun(sv);
    return 0;
  }
```

4 init.xml

4.1 init.xml file

The parameters of a partial differential equation system are defined in the init.xml file. Note that while we refer to this file, always as the init.xml file, its name is defined in the c file that defines the problem, so the name of this xml file can in fact be anything.

4.1.1 <solver> group

The init.xml file consists of the definition of an xml object called "solver". In other words an init.xml file has the form:

```
<?xml version="1.0"?>
<solver [parameters]>
  ...
</solver>
```

Here [parameters] are in fact related to the elements of pdsol_rsolver. The accepted parameters are:

Parameters:

nfields number of time evolving fields

naux number of auxiliary fields (fluxes are usually defined as auxiliary fields to increase speed).

nexout extra output fields.

dt the time step if using fixed time step, or the initial time step if using adaptive time step.

dtout the time step at which the output data will be written (note that when using adaptive time ste if dt>dtout, dt will be used)

tmax the end time of the simulation.

which has to be written in the in accordance with the xml notation. For example, the init.xml for the The Heat Equation starts as:

```
<?xml version="1.0"?>
<solver nfields="1" naux="1" dt="1e-8" dtout="0.01" tmax="0.1" name="heateqn">
  ...
</solver>
```

In addition to the parameters, one has to define objects (corresponding to instances of the created C structures). In particular one has to define the following objects as members of the <solver> object.

- <gsolver>
- <field>
- <mesh>
- <params>

4.1.2 <gsolver> object

<gsolver> object is a member of the <solver> object and describes the parameters related to the pdsol_-gslsolver structure. In particular:

Parameters:

method the time integration method: one of "rk2","rk4","rkf45","rkck","rk8pd","rk2imp","rk4imp","bsimp","gear1","gear
(see `http://www.gnu.org/software/gsl/` section on ODE solver).

step the time stepping method "adaptive" or "fixed".

eps_abs maximum acceptable absolute error in adaptive time stepping.

eps_rel maximum acceptable relative error in adaptive time stepping.

4.1.3 <field> object

There should be a number of fields that are equal to the nfields parameter of the solver object as defined above. Each of these fields are related to the pdsol_rfield structures and has to define, at least:

Parameters:

index the field index, starts from 1.

name the string which defines the name of the field (the output values are saved in an hdf file where the data are organized under field names)

init the filename of the hdf5 file with the initial value data for the field.

4.1.4 <mesh> object

A global mesh has to be defined as well, which is obviously related to the pdsol_mesh structure, and defines the parameters:

Parameters:

x0 the center point of the mesh

dx the grid spacing size.

nx the number of points.

4.1.5 <params> object

Finally, we also need a <params> object, which can be used to define the actual model parameters (as well as other numbers that can be changed at runtime). Usually, the parameters of this obejct are simply the variables used in the problem. The contents of this object, is read into the "params" variable of the pdsol_-solver structure in the form of an array of pdsol_param structures. Note that the order of the parameters of this array is not realiable. One should instead use their names. Ideally, one should write a simple utility function to read the parameters from this array into readable variables as in the example page_angular [e.g. see the get_params() function].

5 Data Structure Index

5.1 Data Structures

Here are the data structures with brief descriptions:

6 File Index

6.1 File List

Here is a list of all documented files with brief descriptions:

7 Data Structure Documentation

7.1 pdsol_cfield Struct Reference

complex field structure with complex numbers at mesh points.

```
#include <field.h>
```

Data Fields

- pdsol_mesh ∗ rm

  the mesh

- char ∗ name

  the name of the field

- complex ∗ dat

  the complex data organized according to the mesh.

- hid_t hdata_id

  the hid_t (id number) of the xml data associated with the field.

7.1.1 Detailed Description

Definition at line 23 of file field.h.

The documentation for this struct was generated from the following file:

- field.h

7.2 pdsol_gslsolver Struct Reference

pdsol_gslsolver gsl solver structure which stores gsl information.

```
#include <solver.h>
```

Data Fields

- const gsl_odeiv_step_type ∗ stype

  gsl ode solver time step type (i.e. rk2, rkf45, rk8 etc.)

- gsl_odeiv_step ∗ step

  pointer to gsl ode step structure

- gsl_odeiv_control ∗ ctrl

  pointer to gsl ode control structure

- gsl_odeiv_evolve ∗ ev

  pointer to gsl ode evolve structure for adaptive time step.

- gsl_odeiv_system ∗ sys

  pointer to gsl ode system structure.

- char adapt

  adaptive time step or not.

- double eps_abs

  maximum allowed, absolute error for adaptive time step.

- double eps_rel

  maximum allowed relative error for adaptive time step.

7.2.1 Detailed Description

Definition at line 214 of file solver.h.

The documentation for this struct was generated from the following file:

- solver.h

7.3 pdsol_mesh Struct Reference

basic mesh stucture

```
#include <mesh.h>
```

Data Fields

- int **nx**
- double ∗ **xx**

7.3.1 Detailed Description

Definition at line 6 of file mesh.h.

The documentation for this struct was generated from the following file:

- mesh.h

7.4 pdsol_param Struct Reference

Stores the name and the value of a parameter.

```
#include <solver.h>
```

Data Fields

- double val

  value of the parameter.

- char ∗ name

  name of the parameter.

7.4.1 Detailed Description

Definition at line 234 of file solver.h.

The documentation for this struct was generated from the following file:

- solver.h

7.5 pdsol_rfield Struct Reference

field structure with real numbers defined at each mesh point.

```
#include <field.h>
```

Data Fields

- pdsol_mesh ∗ rm

    the mesh

- char ∗ name

    the name of the field

- double ∗ dat

    the real data of the field organized according to the mesh.

- hid_t hdata_id

    the hid_t (id number) of the xml data associated with the field.

7.5.1 Detailed Description

Definition at line 11 of file field.h.

The documentation for this struct was generated from the following file:

- field.h

7.6 pdsol_rsolver Struct Reference

the high level solver structure, which defines the problem.

```
#include <solver.h>
```

Data Fields

- pdsol_mesh ∗ rm

    the mesh.

- pdsol_rfield ∗∗ fields

    the actual fields.

- pdsol_rfield ∗∗ exout

    extra output fields, or dummy fields for additional output.

- int nfields

    number of actual fields.

- int naux

number of auxiliary fields (e.g. fields that don't evolve in time)

- int nparams

  number of parameters.

- int nexout

  number of extra output fields

- double dtfixed

  the time step if using fixed time step.

- double dtout

  time step at which the result will be written.

- double tmax

  the ending time of the simlation.

- double t

  the actual time.

- double h

  the actual time step (set automatically if using adaptive time step)

- double t0

  the initial time of the simulation.

- pdsol_gslsolver ∗ gsv

  the pointer to the pdsol_gslsolver structure.

- double ∗ data

  the raw data of the actual fields.

- double ∗ aux

  the raw data of the auxiliary fields.

- double ∗ exdat

  the raw data of the extra output fields.

- pdsol_param ∗ params

  parameters

- char ∗ name

  a name for the problem.

- char adaptive

  adaptive time step or not.

- mxml_node_t ∗ tree

  the xml tree that defines the problem.

- hid_t hfile_id

  the hid_t objects that define the file and the group within the xml tree.

- hid_t **hgroup_id**
- int nt

  the time step number.

- void ∗ user

  user defined data.

7.6.1 Detailed Description

Definition at line 242 of file solver.h.

The documentation for this struct was generated from the following file:

- solver.h

8 File Documentation

8.1 field.h File Reference

```
#include "mesh.h"
#include <mxml.h>
#include <complex.h>
#include <hdf5.h>
```

Data Structures

- struct pdsol_rfield

  field structure with real numbers defined at each mesh point.

- struct pdsol_cfield

  complex field structure with complex numbers at mesh points.

Functions

- pdsol_rfield ∗ pdsol_init_rfield (pdsol_mesh ∗rm, mxml_node_t ∗node, double ∗dptr)

  initializes a real field.

- void pdsol_print_rfield (pdsol_rfield ∗phi)

  prints out a real field

- pdsol_cfield ∗ pdsol_init_cfield (pdsol_mesh ∗rm, mxml_node_t ∗node, double ∗dptr)

  initializes a complex field.

- void pdsol_print_cfield (pdsol_cfield ∗phi)

  prints out a complex field.

8.1.1 Detailed Description

Definition in file field.h.

8.1.2 Function Documentation

8.1.2.1 pdsol_cfield∗ pdsol_init_cfield (pdsol_mesh ∗ *rm*, mxml_node_t ∗ *node*, double ∗ *dptr*)

this function initializes a complex field structure (pdsol_cfield) using the given mesh, the field node from an xml tree and the pointer to its data.

Parameters:

rm mesh.

node the field node.

dptr the data for the real field.

Returns:

pointer to the newly initialized pdsol_cfield

Definition at line 44 of file field.c.

8.1.2.2 pdsol_rfield∗ pdsol_init_rfield (pdsol_mesh ∗ *rm*, mxml_node_t ∗ *node*, double ∗ *dptr*)

this function initializes a real field structure (pdsol_rfield) using the given mesh, the field node from an xml tree and the pointer to its data.

Parameters:

rm mesh.

node the field node.

dptr the data for the real field.

Returns:

pointer to the newly initialized pdsol_rfield

Definition at line 9 of file field.c.

8.1.2.3 void pdsol_print_cfield (pdsol_cfield ∗ *phi*)

simply prints out the data of a complex field.

Parameters:

phi the field to be printed

Definition at line 71 of file field.c.

8.1.2.4 void pdsol_print_rfield (pdsol_rfield ∗ *phi*)

simply prints out the data of a reald field.

Parameters:

phi the field to be printed

Definition at line 36 of file field.c.

8.2 mesh.h File Reference

```
#include <mxml.h>
```

Data Structures

- struct pdsol_mesh

  basic mesh stucture

Functions

- pdsol_mesh ∗ pdsol_init_mesh (mxml_node_t ∗node)

  initializes a mesh

- void pdsol_print_mesh (pdsol_mesh ∗rm)

  prints the mesh

8.2.1 Detailed Description

Definition in file mesh.h.

8.2.2 Function Documentation

8.2.2.1 pdsol_mesh∗ pdsol_init_mesh (mxml_node_t ∗ *node*)

this function initializes a simple uniform rectangular mesh using the xml node which describes the mesh.

Parameters:

node the xml node

Returns:

pointer to a pdsol_mesh structure.

Definition at line 8 of file mesh.c.

8.2.2.2 void pdsol_print_mesh (pdsol_mesh ∗ *rm*)

prints the mesh

Parameters:

rm the mesh

Definition at line 28 of file mesh.c.

8.3 pdutils.h File Reference

```
#include <hdf5.h>
#include <solver.h>
```

Functions

- char ∗ pdsol_sysinfo ()
  returns information about the current platform.
- char ∗ pdsol_date ()
  returns system time. (using ctime).
- void pdsol_hdfwrite (hid_t dest_id, hid_t space_id, hid_t type_id, char ∗name, const void ∗val)
  writes data as hdf file.
- double pdget_par (pdsol_rsolver ∗sv, char ∗name)
  reads a parameter from the xml file.

8.3.1 Detailed Description

Definition in file pdutils.h.

8.3.2 Function Documentation

8.3.2.1 double pdget_par (pdsol_rsolver ∗ *sv*, char ∗ *name*)

reads a parameter from the xml file

Parameters:

sv pointer to the solver structure.

name name of the parameter.

Returns:

the value of the parameter.

Definition at line 48 of file pdutils.c.

8.3.2.2 void pdsol_hdfwrite (hid_t *dest_id*, hid_t *space_id*, hid_t *type_id*, char ∗ *name*, const void ∗ *val*)

writes the given data in the form of an hdf file.

Parameters:

dest_id destination id.
space_id space id.
type_id type id.
name filename.
val the data.

Definition at line 40 of file pdutils.c.

8.4 pdxml.h File Reference

```
#include <mxml.h>
```

Functions

- int pdsol_getxmlint (mxml_node_t ∗node, const char ∗name)
  get integer value from xml node.
- char ∗ pdsol_getxmlstr (mxml_node_t ∗node, const char ∗name)
  get string from xml node.
- double pdsol_getxmldouble (mxml_node_t ∗node, const char ∗name)
  get double value from xml node.

8.4.1 Detailed Description

Definition in file pdxml.h.

8.4.2 Function Documentation

8.4.2.1 double pdsol_getxmldouble (mxml_node_t ∗ *node*, const char ∗ *name*)

searches the given xml node, branching if necessary, for a parameter with the given name. Returns the double value of this parameter

Parameters:

node xml node to be searched.
name the name of the parameter to be searched.

Returns:

double value of the parameter if found. it prints out an error message and returns null in the case of an error.

Definition at line 15 of file pdxml.c.

8.4.2.2 int pdsol_getxmlint (mxml_node_t ∗ *node*, const char ∗ *name*)

searches the given xml node, branching if necessary, for a parameter with the given name. Returns the integer value assigned to this parameter in the xml node.

Parameters:

node xml node to be searched.

name the name of the parameter to be searched.

Returns:

the integer value of the parameter if found. Prints out an error message and returns 0 if the parameter can't be found.

Definition at line 4 of file pdxml.c.

8.4.2.3 char∗ pdsol_getxmlstr (mxml_node_t ∗ *node*, const char ∗ *name*)

searches the given xml node, branching if necessary, for a string parameter with the given name. Returns the pointer to a newly allocated string with the value of this parameter

Parameters:

node xml node to be searched.

name the name of the parameter to be searched.

Returns:

the pointer to the string or NULL on error.

Definition at line 26 of file pdxml.c.

8.5 solver.h File Reference

```
#include "mesh.h"
#include "field.h"
#include <mxml.h>
#include <hdf5.h>
#include <gsl/gsl_odeiv.h>
```

Data Structures

- struct pdsol_gslsolver

  pdsol_gslsolver gsl solver structure which stores gsl information.

- struct pdsol_param

  Stores the name and the value of a parameter.

- struct pdsol_rsolver

  the high level solver structure, which defines the problem.

Functions

- pdsol_rsolver ∗ pdsol_init_rsolver (const char ∗filename)

  initializes the pdsol_rsolver structure.

- pdsol_gslsolver ∗ pdsol_init_gslsolver (mxml_node_t ∗node, int sz)

  initializes the gsl solver.

- pdsol_param ∗ pdsol_getparams (mxml_node_t ∗node, int ∗n)

  reads the parameters.

- void pdsol_hwrite_rfields (pdsol_rsolver ∗sv)

  writes the fields into the hdf file.

- int pdsol_gslrun (pdsol_rsolver ∗sv)

  runs the problem defined by the solver.

- void pdsol_print_rsolver (pdsol_rsolver ∗rm)

  prints some information about the solver.

8.5.1 Detailed Description

Definition in file solver.h.

8.5.2 Function Documentation

8.5.2.1 pdsol_gslsolver∗ pdsol_init_gslsolver (mxml_node_t ∗ *node*, int *sz*)

initializes the gsl solver using the corresponding xml node. Note that this is called automatically by pdsol_init_rsolver.

Parameters:

node the xml node with the gsl solver information.

sz the total size of the data to be passed, usually nfields∗npoints.

Returns:

pointer to the created pdsol_gslsolver strucutre.

Definition at line 147 of file solver.c.

8.5.2.2 pdsol_rsolver∗ pdsol_init_rsolver (const char ∗ *filename*)

initializes the pdsol_rsolver structure using an xml file.

Parameters:

filename the name of the xml file

Returns:

pointer to the created pdsol_rsolver structure.

Definition at line 203 of file solver.c.

Index