NGSolve - 4.4

Joachim Schöberl

February 6, 2013

Contents

Chapter 1

Getting Started

NGSolve is a finite element library. It contains equation assembling and equation solving. It does not contain mesh operations, so it has to be linked to a mesh handler such as Netgen.

1.1 Installing NGSolve as Netgen Add-On

Install Netgen and NGsolve as expained on `http://sourceforge.net/apps/mediawiki/ngsolve`

1.2 A first demo

Start the program 'netgen'. There should pop up a window with menu items, and a large white visualization area.

Select the menu item *Solve → Load PDE* and open the PDE file *ngsolve/pde_tutorial/d1_square.pde*. You get some text in the shell window, and you will find a blue square in the drawing area. You have loaded the problem description, including geometry and mesh. In the button line (below the menu items) you see the selection-button showing 'Geometry'. Here, you can switch to 'Mesh', and later, you switch to 'Solution'. This selects the visualization mode in the drawing area.

Now, you are ready to start the first solve. Press the *Solve*-button.. The visualization switches to Solution, and you see two gray triangles.

Click *Solve→Visualization* to open the visualization dialog box. Here, you can select what you want to see. Look for the selection button *Scalar Function*, and change from 'None' to 'u'. The triangles will be colored.

Clearly, this mesh is too coarse. Go on, and press the *Solve*-button five more times. The mesh gets refined, and new solutions will be computed.

Play around with the items in the visualization box. You may try Autoscale, Isolines, and Textures. Go on and do some more refinement steps (by pressing the short-cuts `<s>`-`<p>`.

The next chapter will describe the structure of the PDE input file.

Chapter 2

The PDE Description file

The way to describe the equations in NGSolve is very close to the variational formulation (the virtual displacement formulation).

2.1 The weak formulation

The partial differential equation describing a heat flow problem with thermic conductivity λ, a volumetric source density f, an inflow density g at the part Γ_N of the boundary, and a heat exchange to the environment of temperature u_0, and conductivity α on Γ_R is

$$
\begin{array}{rcll}
-\mathrm{div}\lambda\nabla u & = & f & \text{in } \Omega, \\
n^T\lambda\nabla u & = & g & \text{on } \Gamma_N, \\
n^T\lambda\nabla u & = & \alpha(u_0 - u) & \text{on } \Gamma_R.
\end{array}
$$

The weak form of the equation is to find the temperature distribution u in the proper Sobolev space $V := H^1(\Omega)$ such that

$$
\int_\Omega \lambda\nabla u \cdot \nabla v dx + \int_{\Gamma_R} \alpha uv \, ds = \int_\Omega fv \, dx + \int_{\Gamma_N} gv \, ds + \int_{\Gamma_R} \alpha u_0 v \, ds
$$

holds for all test functions $v \in V$. The finite element method replaces the Sobolev spaces V by a finite dimensional sub-space.

The PDE input file defines the variational problem. You specify the finite element space, and then, the bilinear-form (left hand side), and the linear form (right hand side) are build up from elementary blocks called integrators. The table below lists the names of the integrators needed in the equation above:

laplace lam	$\int_\Omega \lambda\nabla u \cdot \nabla v \, dx$
robin alpha	$\int_{\Gamma_R} \alpha uv \, ds$
source f	$\int_\Omega fv \, dx$
neumann g	$\int_{\Gamma_N} gv \, ds$

Note that the last term is handled by the neumann - integrator with coefficient αu_0. You might miss Dirichlet boundary conditions. Indeed, NGSolve always approximates them by Robin b.c. with large conductivity α.

2.2   The input file

The file below is a valid input file to NGSolve. The first two lines load the prepared geometry file and the mesh file. It is a square. The boundary splits into the 4 sides Γ_1 to Γ_4. We specify Robin boundary conditions (with conductivity $\alpha = 10^5$) on Γ_1 to Γ_3, and Neumann boundary condition (with $g = 1$) on Γ_4. This is defined in terms of coefficients. The list of numbers correspond to the sub-domains, if the integral is taken over the domain, or, to parts of the boundary, if the integral is taken over the boundary, respectively. The next lines define the mathematical objects finite-element space, grid-function, bilinear-form, linear-form, and a preconditioner. The last line (numproc) calls the solver for boundary value problems (bvp). Here, a preconditioned conjugate gradients iteration is called.

```
geometry = ngsolve/pde_tutorial/square.in2d
mesh = ngsolve/pde_tutorial/square.vol

define coefficient coef_lam
1,

define coefficient coef_alpha
1e5, 1e5, 1e5, 0,

define coefficient coef_f
1,

define coefficient coef_g
0, 0, 0, 1,

define fespace v -order=1
define gridfunction u -fespace=v -nested

define bilinearform a -fespace=v
laplace coef_lam
robin coef_penalty

define linearform f -fespace=v
source coef_source
```

```
neumann coef_g

define preconditioner c -type=multigrid -bilinearform=a -smoothingsteps=1

numproc bvp np1 -bilinearform=a -linearform=f -gridfunction=u -preconditioner=c \
    -maxsteps=50
```

Chapter 3

Reference Manual

This is the most up to date description of the PDE input file. General rules for writing input files are:

- The input file is case sensitive. All keywords must be in lower-case.

- Line comments start with the pound sign #. The rest of the line is ignored. Block comments are not available.

- The input file must start with geometry and mesh definition. The geometry is optional, the mesh statement is required:

  ```
  geometry = geodir/geofilename.geo
  mesh = meshdir/meshfilename.vol
  ```

  The filename is relative to the starting directory of ng. Spaces are optional.

- After that, `define` commands and `numproc` commands follow. Each object must be given a unique name. The objects are called in order of definition. The calling action depends on the type of object. For example, it is memory allocation for grid-functions, matrix assembling for bilinear-forms, and linear equation solving for the bvp-numproc.

- Many commands require option flags. There are define flags, numerical flags, string flags, number-list flags, and string-last flags. Examples for every type are

  ```
  -printstatus
  -ref_fac=0.1 -solution=u
  -parameters=[1.0,2,-5] -spaces=[v1,v2,v3]
  ```

  Flags need an initial '-' character. Spaces within one flag are not allowed.

3.1 Constants and Variables

Constants and variables are defined as follows:

```
define constant pi = 3.1415
define constant omega = (2 * pi * 10E3)
define variable t = 0
```

A constant is given its value at definition. A variable is initialized to a value, but later, its value may be changed from a numproc. An example is the time variable for dynamic problems.

The right hand side can be a constant, or a constant expression. Expressions must be put into brackets. Expressions may contain the following operations:

- $+, -, /, *, ()$

- sin, cos, tan, exp, log

3.2 Coefficient functions

Coefficient functions are defined as list of constants or functions. Each element corresponds to a sub-domain, or a part of the boundary (specified by the boundary condition number).

```
define coefficient coef_lam  10,15, -5, (5*x), (sin(y)),
```

- Functions must be put into brackets. Function may be built as described in Section **??**, but may now additionally contain variables.

- Predefined variables are x, y, and z for the Cartesian coordinates.

- Specifying too many coefficients is ok, too less will throw an error exception.

Coefficients can be specified by materials. Use the '-material' flag. Currently this version is only supported for CSG geometries.

```
define coefficient coef_nu material
        iron 10000
        air  1
        default 1
```

All subdomains consisting of material 'iron' get the constant coefficient 10000, the 'air' domain as well as all others get 1.

3.3 Finite Element Spaces

The definition

```
define fespace <name> <flaglist>
```

defines the finite element space `<name>`. Example:

```
define fespace v -order=2 -dim=3
```

There are various classes of finite element spaces. Default are continuous, nodal-valued finite element spaces. The following define flags select the type of spaces

non of the flags below	continuous nodal finite element space
`-hcurl`	H(curl) finite elements (Nedelec-type, edge elements)
`-hdiv`	H(div) finite elements (Raviart-Thomas, face elements)
`-l2`	non-continuous elements, element by element
`-l2surf`	element by element on surface
`-h1ho`	Arbitrary order continuous elements
`-hcurlho`	Arbitrary order H(curl) elements
`-hdivho`	Arbitrary order H(div) elements
`-l2ho`	Arbitrary order non-continuous elements

The following flags specify the finite element spaces

`-order=<num>`	Order of finite elements
`-dim=<num>`	Number of fields (number of copies of fe), 2 for 2D elasticity
`-vec`	set -dim=spacedim
`-tensor`	set -dim=spacedim*spacedim
`-symtensor`	set -dim=spacedim * (spacedim+1) / 2, (symmetric stress tensor)
`-complex`	complex valued fe-space

A compound fe-space combines several fe-spaces to a new one. Useful, e.g., for Reissner-Mindlin plate models containing the deflection w and two rotations beta:

```
fespace vw -order=2
fespace vbeta -order=1
fespace v -compound -spaces=[vw,vbeta,vbeta]
```

The fespace maintains the degrees of freedom. On mesh refinement, the space provides the grid transfer operator (prolongation). High order fe spaces maintain a lowest-order fespace of the same type for preconditioning.

3.3.1   H1-Finite Element Space

3.4   Grid-functions

A grid-function is a function living in a finite element space. Definition:

```
define gridfunction <name> <flaglist>
```

Example:

```
define gridfunction u -fespace=v
```

The string-flag `fespace` defines the fespace the grid function lives in. The flag must refer to a valid fespace.

If the define flag `nested` is specified, the grid-function will be prolongated from the coarse space to the fine space when the mesh is refined.

3.5   Bilinear-forms

A bilinear form is defined by

```
define bilinearform <name> <flaglist>
integrator1
integrator2
integrator3
...
```

Example

```
define bilinearform a -fespace=v
laplace lam
robin alpha
```

A bilinear-form is always defined as sum over integrators. A bilinear-form maintains the stiffness matrix. For multi-level algorithms, a bilinear-form stores all matrices. Bilinear-forms for high order spaces have a bilinear-form for the corresponding lowest order space.

The following flags are defined

-fespace=¡name¿	bilinear form is defined on fe space ¡name¿
-symmetric	bilinear form is symmetric (store just lower left triangular matrix)
-nonassemble	do not allocate matrix (bilinear-form is used, e.g., for post-processing)
-project	use Galerkin projection to generate coarse grid matrices

An integrator is defined as

```
token <coef1> <coef2> ... <flaglist>
```

Example:

```
elasticity coef_e coef_nu -order=4
```

The ¡coefi¿ refers to a coefficient function defined above. It provides the coefficients defined sub-domain by dub-domain for integrators defined on the domain (e.g., laplace), or, the coefficient boundary-patch by boundary-patch for integrators defined on the surface (e.g., robin).

Allowed flags are

-order=num	use integration rule of order num. Default order is computed form element order.
-comp=num	use scalar integrator as component num for system (e.g., penalty term for y-displace
-normal	add integrator in normal direction (penalty for normal-displacement)

The integrator tokens are

laplace lam	$\int_\Omega \lambda \nabla u \cdot \nabla v \, dx$
mass rho	$\int_\Omega \rho u v \, dx$
robin alpha	$\int_\Gamma \alpha u v \, ds$
elasticity e nu	$\int_\Omega D\varepsilon(u) : \varepsilon(v) \, dx$ (with D..3D elasticity tensor, or plane stress)
curlcurledge nu	$\int_\Omega \nu(\nabla \times u)(\nabla \times v) \, dx$ for $H(curl)$ spaces
massedge sigma	$\int_\Omega \sigma u \cdot v dx$ for $H(curl)$ spaces
robinedge sigma	$\int_\Gamma \sigma(n \times u)(n \times v) ds$ for $H(curl)$ spaces

3.6   Linear-forms

A linear form is defined by

```
define linearform <name> <flaglist>
integrator1
integrator2
integrator3
...
```

Example

```
define linearform f -fespace=v
source coef_f
neumann coef_g
```

A linear-form is always defined as sum over integrators. A linear-form maintains the right hand side vector.

The following flags are defined

| -fespace=¡name¿ | bilinear form is defined on fe space ¡name¿ |

An integrator is defined as

```
token <coef1> <coef2> ... <flaglist>
```

Example:

```
source coef_fy -comp=2
```

The ¡coefi¿ refers to a coefficient function defined above. It provides the coefficients defined sub-domain by dub-domain for integrators defined on the domain (e.g., source), or, the coefficient boundary-patch by boundary-patch for integrators defined on the surface (e.g., neumann).

Allowed flags are

-order=num	use integration rule of order num. Default order is computed form element order
-comp=num	use scalar integrator as component num for system (e.g., penalty term for y-displ
-normal	add integrator in normal direction (surface load in normal direction)

The integrator tokens are

source f	$\int_\Omega f v \, dx$
neumann g	$\int_\Gamma g v \, ds$
sourceedge jx jy jz	$\int_\Omega j \cdot v \, dx$ for 3D $H(curl)$ spaces
neumannedge jx jy jz	$\int_\Gamma (n \times j) \cdot (n \times v) \, ds$ for 3D $H(curl)$ spaces
curledge f	$\int_\Omega f(\nabla \times v_z) \, dx$ for 2D $H(curl)$ spaces
curlboundaryedge f	$\int_\Gamma f n \cdot (\nabla \times v) \, ds$ for 3D $H(curl)$ spaces

3.7   Preconditioners

A preconditioner is defined by

```
define preconditioner <name> -type=<type> <flaglist>
```

Example:

```
define preconditioner c -type=multigrid -bilinearform=a -smoothingsteps=2
```

A preconditioner must have a type flags specifying the type of preconditioner. The remaining flags depend on the preconditioner.

Preconditioners are:

- -type=local: Local preconditioner, symmetric Gauss-Seidel Jacobi, block Gauss-Seidel

  Flags are
-bilinearform=¡name¿	name of bilinear-form containing matrix
-block	use block Gauss-Seidel (block defined by fe-space)

- -type=multigrid: Multigrid preconditioner

Flags are	-bilinearform=¡name¿	name of bilinear-form containing matrix
	-smoother=¡smoother¿	type of smoother: 'point'..GS, 'block'..block GS, 'li
	-coarsetype=¡coarse¿	type of coarse grid solver: 'exact'..factorization, 'sm
	-smoothingsteps=nsm	number of pre- and post-smoothing steps
	-increasesmoothingsteps=inc	smoothing steps on level l are $nsm * inc^{L-l}$ with L.
	-coarsesmoothingsteps=nsmc	smoothing steps for coarse grid solver (if smoother)
	-finesmoothingsteps=nsmf	smoothing steps for high order equation
	-test	compute eigenvalues of preconditioned system
	-timing	measure time per preconditioning step
	-updateall	update coarse grid matrices (e.g. in combination wi

- -type=direct: Cholesky factorization

3.8 Numerical Procedures

Numerical procedures are functions where the actual computations happen. Numprocs call iterative solvers, perform time integration loops, control postprocessing, and error estimators.

3.8.1 Boundary Value Problem

Keyword: `bvp`

The numproc bvp takes a matrix (from a bilinear form), a right hand side vector (from a gridfunction), call an iterative solver to compute the solution vector stored in a gridfunction.

Example:

```
define bvp np1 -bilinearform=a -linearform=f -solution=u -preconditioner=c
```

Flags are:	-bilinearform=¡name¿	bilinear form to provide matrix
	-linearform=¡name¿	linear form to provide right hand side vector
	-gridfunction=¡name¿	gridfunction to store solution
	-preconditioner=¡name¿	preconditioner for iterative solver
	-maxsteps=num	maximal number of iterations
	-prec=num	relative error reduction
	-solver=cg—qmr	choice of iterative solver, default is cg.
	-print	print matrix, rhs, and solution to test.out file

3.8.2 Post processing

Keyword: `calcflux`

Compute derivatives of solution. Depending on the problem, this function computes the gradient, the flux, strain or stress, magnetic induction, etc...

Example:

```
numproc calcflux pp1 -bilinearform=a -solution=u -flux=p -applyd
```

	bilinearform=¡name¿	The flux is defined by the first integrator of the bilinearfor
Flags are:	solution=¡name¿	The input gridfunction
	flux=¡name¿	The output gridfunction. Must be nodal-valued or element
	-applyd	apply coefficientmatrix. Switches between stress and strai

3.8.3   Evaluation of numerical values

Keyword: `evaluate`

Evaluate (bi)linear functionals on gridfunctions, compute point values, etc,..

Examples:

```
numproc evaluate ev1 -linearform=f -gridfunction=u -text=NormalFlux
numproc evaluate ev2 -bilinearform=a -gridfunction=u -point=[0.3,0.5,0.5] -applyd
```

	-bilinearform=¡name¿	bilinear form to evaluate a(u,v), or, take first integrator
	-linearform=¡name¿	linear form to evaluate f(v)
	-gridfunction=¡name¿	gridfunction to evaluate
	-gridfunction2=¡name¿	gridfunction v to evaluate a(u,v)
Flags are:	-point=[px,py,pz]	point where to evaluate Bu
	-point2=[qx,qy,qz]	evaluate elong line [P,Q] and store values in file
	-filename=¡name¿	file to store results
	-applyd	switch between stain/stress, i.e., Bu or DBu
	-text=¡text¿	output is "¡text¿ = " values

3.8.4   Error estimator

Keyword: `zzerrorestimator`

Performs a Zienkiewicz-Zhu type error estimator.

Example:

```
numproc zzerrorestimator ee1 -bilinearform=a -solution=u -error=err
```

Averages the fluxes, and computes difference between averaged flux and origi-
nal flux. Difference is added to element by element gridfunction error. Averaging
is done sub-domain by sub-domain to avoid too strong refinement along material
interfaces.

-bilinearform=¡name¿	first integrator of bilinear-form defines flux
-solution=¡name¿	gridfunction containing solution
-error=¡name¿	element-by-element, order 0 gridfunction containing element contrib

3.8.5 Refinement marker

Keyword: `markelements`

Marks elements for mesh refinement.

Example:

```
numproc markelements me -error=err -minlevel=2 -fac=0.1
```

All elements having element error greater than fac times maximal element error are marked for refinement.

-error=¡name¿	element by element, order 0, gridfuncti
-minlevel=¡l¿	Do l-1 levels of uniform refinement bef
-fac=val	Elements having contribution greater t
-error2=¡name¿	mark elements based on element by ele

error_T * error2_T, used for goal driven error estimator.