Sundance Examples: Linear PDE

Kevin Long

June 21, 2012

1 Example: a parameter sweep

A common task is to solve a problem with several values of a parameter. For example, we might be interested in the problem

$$-\nabla \cdot \left[\left(1 + \frac{\xi}{10} e^x \right) \nabla u \right] - f = 0$$

on the unit square with boundary conditions

$$u = 0 \quad \text{on east and west edges}$$

$$\frac{\partial u}{\partial n} = 0 \quad \text{on north and south edges.}$$

To investigate how the solution change as the parameter ξ is changed we sweep over an interval $[0, \xi_{\text{max}}]$ using some number N_{step} of samples.

The obvious approach to programming this is to encapsulate the entire problem in a function with ξ as an input argument, and then call that function for each ξ in a loop. The trouble is that the linear problem and all its internal data structures must be rebuilt on each call.

A better solution is to build the equation set and problem once and for all, writing the adjustable parameter ξ not as a double-precision number but as a special expression type, `Parameter`. The value of the parameter can be changed between solves, allowing the same problem object to be reused for multiple solves at multiple parameter values. How does it work? Simple: the parameter value is shallow copied when given to the weak form and other expressions, so any change made later by the user is immediately and transparently reflected in all copies.

The full source code for this example is in **ParameterSweep.cpp**. In the following, we omit parts of the code not relevant to the parameter sweep such as construction of the mesh and cell filters, construction of unknown functions, reading of the solver file, and so on. All those details are of course shown in the full source code.

The `Parameter` expression is constructed as follows.

```
/* Define the parameter */
Expr xi = new Sundance::Parameter(0.0);
```

The namespace qualification is used to distinguish `Sundance::Parameter` from `Teuchos::Parameter`. In this problem, the value 0.0 given to the parameter constructor is unimportant, as it will be changed during the sweep.

With the parameter expression ready, we proceed to setting up a problem that uses it. For validation purposes, our example problem will have an exact solution which, unsurprisingly, depends on the paremeter value. We define the exact solution here, and then provide a parameter-depended forcing function f that produces that solution.

```
/* Construct a forcing term to provide an exact solution */
Expr uEx = x*(1.0-x)*(1.0+xi*exp(x));
Expr f = -(-20 - exp(x)*xi*(1 + 32*x + 10*x*x + exp(x)*(-1 + 2*x*(2 + x))*xi))
    /10.0;
```

Now we write the weak form, again using the parameter.

```
/* Define the weak form */
Expr eqn = Integral(interior, (1.0 + 0.1*xi*exp(x))*(grad*v)*(grad*u) - f*v,
    quad);
```

Next we set up the linear problem. Also, to visualize the exact solution we'll create a projector object that projects it onto a discrete space. The projector is given the expression `uEx`, which in turn contains a shallow copy of the parameter `xi`. Therefore, just as with the linear problem the internal state of the projector is always consistent with the current parameter value.

```
/* We can now set up the linear problem. This can be reused
 * for different parameter values. */
LinearProblem prob(mesh, eqn, bc, v, u, vecType);

/* make a projector for the exact solution. Just like the
 * problem, this can be reused for different parameter values. */
DiscreteSpace ds(mesh, new Lagrange(1), vecType);
L2Projector proj(ds, uEx);
```

With the problem and projector constructed, we're ready to loop over parameter values. The `setParameterVal` member function is used to update the parameter's numerical value.

```
/* Set up the sweep from xi=0 to xi=xiMax in nSteps steps. */
int nSteps = 10;
double xiMax = 2.0;


/* Do the sweep */
for (int n=0; n<nSteps; n++) {
  /* Update the parameter value */
  double xiVal = xiMax*n/(nSteps - 1.0);
  xi.setParameterValue(xiVal);
  Out::root() << "step n=" << n << " of " << nSteps << " xi=" << xiVal;
```

```
  /* Solve the problem. The updated parameter value is automatically used. */
  state = prob.solve(solver, soln);

   TEUCHOS_TEST_FOR_EXCEPTION(state.finalState() != SolveConverged, std::
       runtime_error,
     "solve failed!");

  /* Project the exact solution onto a discrete space for viz. The updated
     * parameter value is automatically used. */
  Expr uEx0 = proj.project();

  /* === output and validation code omitted === */
}
```

Notice that the same problem and projector objects were reused at each iteration.

Other applications of `Parameter` include:

- Continuation methods for nonlinear equations (see the document on nonlinear problems). Use a `Paremeter` for the continuation variable.
- In transient problems, the time variable can be defined as a `Parameter` allowing the same problems to be reused at every timestep.
- Multidimensional sampling methods such as Monte Carlo, latin hypercube, and sparse quadrature. In these problems, more than one `Parameter` object might be used.

2 Example: Coupled problems in 2D

2.1 Problem formulation

We solve the coupled equations

$$-\nabla^2 u_1 + u_2 = 0$$
$$-\nabla^2 u_2 + x = 0$$

on the unit square with boundary conditions

$$u_1(0,y) = u_1(1,y) = 0$$
$$u_2(0,y) = u_2(1,y) = 0$$
$$\frac{\partial u_1}{\partial n} = \frac{\partial u_2}{\partial n} = 0 \text{ on north and south boundaries.}$$

The exact solution is

$$u_1(x) = \frac{1}{360}\left(3x^5 - 10x^3 + 7x\right)$$
$$u_2(x) = \frac{1}{6}\left(x^3 - x\right)$$

2.1.1 Weak equation

The Galerkin weak form is

$$\int_\Omega \nabla v_1 \cdot \nabla u_1 + v_1 u_2 + \nabla v_2 \cdot \nabla u_2 + v_2 x \, d\Omega = 0$$

augmented by essential boundary conditions. We'll discretize u_1 with P^3 and u_2 with P^2.

2.2 Programming the problem

2.2.1 Meshing a rectangle

2.2.2 Coordinate-based cell filters

2.2.3 Writing the equations

2.2.4 Forming the coupled linear problem

2.2.5 Postprocessing and validation

3 Example: Convective cooling with ideal flow

4 Example sequence: Several formulations of Stokes flow

4.1 Driven cavity, vorticity-streamfunction formulation

4.2 Couette flow, pressure-stabilized equal-order discretization

4.3 Couette flow, mixed discretization and block preconditioning