AFEPack Notes

Liu Chengyu

December 23, 2020

Parabolic Equations

Finishing the study the example of poisson-equation, I tried to add the $\frac{\partial u}{\partial t}$ into the problem. Thus the problem changed as follow:

$$\frac{\partial u}{\partial t} - \triangle u = f \ in \ \ \Omega, \tag{1}$$

$$u_{t=0}(x, y) = sin(\pi(x))sin(2\pi y), \tag{2}$$

$$u = u \ on \ \partial\Omega, \tag{3}$$

$$\Omega = [0, 1]x[0, 1]. \tag{4}$$

In fact, we can set the $u = sin(\pi(x + t))sin(2\pi y)$, then we can get the exact f as follow:

$$f = \pi cos(\pi(x + t))sin(2\pi y) + 5\pi^2 sin(\pi(x + t))sin(2\pi y)$$

Due to the special form of the function u in (1), we could image that the graph of new equation would show as the graph of the original poisson equation moving through the x-axis.

Then I use the AFEPack to solve the problem to verify our assumption. At first, I discretized the equation in time as follow:

$$\frac{u^n - u^{n-1}}{dt} - \triangle u^n = f. \tag{5}$$

then I can transform the (5) as :

$$\frac{u^n}{dt} - \triangle u^n = f + \frac{u^{n-1}}{dt}. \tag{6}$$

Multiplying trial functions v in both side of (6), then we can define the new bilinear operater $a(u, v)$ as :

$$\int_\Omega (\frac{u^n}{dt} v + \bigtriangledown u \cdot \bigtriangledown v) = \int_\Omega (f + \frac{u^{n-1}}{dt}). \tag{7}$$

In fact, we have the result of u^{n-1} at n time step and we just need to calculate the value of the u^n. So the (7) can be converted as:

$$\int_\Omega \frac{uv}{dt} + \bigtriangledown u \cdot \bigtriangledown v = \int_\Omega f + \frac{u_h}{dt}. \tag{8}$$

in which u_h represents the result at n-1 time step. Obviously, (8) is different as the Galerkin form of poisson equation. So I change the function getElementmatrix in the .cpp file to build a new stiff matrix. What's more, I also add the new item $\frac{u_h}{dt}$ in to the right hand side as follow codes:

```
class Matrix : public L2InnerProduct<DIM,double>
{
    private:
    double _dt;
    public:
    Matrix(FEMSpace<double,DIM>& sp, double dt) :
    L2InnerProduct<DIM,double>(sp, sp), _dt(dt) {}
    virtual void getElementMatrix(const Element<double,DIM>& e0,
    const Element<double,DIM>& e1,
    const ActiveElementPairIterator< DIM >::State s)
    {
        double vol = e0.templateElement().volume();
        u_int acc = algebricAccuracy();
        ...
        ...
        for (u_int l = 0;l < n_q_pnt;++ l) {
            double Jxw = vol*qi.weight(l)*jac[l];
            for (u_int i = 0;i < n_ele_dof;++ i) {
                for (u_int j = 0;j < n_ele_dof;++ j) {
                    elementMatrix(i,j) += Jxw*(bas_val[i][l]*bas_val[j
                        ][l]/_dt +
                    innerProduct(bas_grad[i][l], bas_grad[j][l]));
                }
            }
        }
    }
};
```

In above code, I add the new items into the element stiff matrix by adding them in the command:

```
elementMatrix(i,j) += Jxw*(bas_val[i][l]*bas_val[j][l]/_dt +
innerProduct(bas_grad[i][l], bas_grad[j][l]));
```

Then I add the new item in the right hand side by this:

```
Vector<double> rhs(fem_space.n_dof());
FEMSpace<double,DIM>::ElementIterator the_ele = fem_space.beginElement
    ();
FEMSpace<double,DIM>::ElementIterator end_ele = fem_space.endElement();
for (;the_ele != end_ele;++ the_ele) {
    double vol = the_ele->templateElement().volume();
    const QuadratureInfo<DIM>& qi = the_ele->findQuadratureInfo(4);
    u_int n_q_pnt = qi.n_quadraturePoint();
    ...
    ...
    const std::vector<int>& ele_dof = the_ele->dof();
    u_int n_ele_dof = ele_dof.size();
    for (u_int l = 0;l < n_q_pnt;++ l) {
        double Jxw = vol*qi.weight(l)*jac[l];
        double f_val = _f_(q_pnt[l]);
        for (u_int i = 0;i < n_ele_dof;++ i) {
            // Build the rhs vector;
            rhs(ele_dof[i]) += Jxw*bas_val[i][l]*(u_h_val[l]/dt + f_val
                );
        }
    }
```

After these changes, I get the numerical result of u. But it shows like there exist a shake every 50 time steps iterations(the maximum value changes besides 1).

Here are the graph:

References