9.4 Eigenvalue Problems

This section depends on your installation of FreeFem++ ; you need to have compiled (see README_arpack), ARPACK. This tools is available in FreeFem++ if the word "eigenvalue" appear in line "Load:", like:

```
-- FreeFem++ v1.28 (date Thu Dec 26 10:56:34 CET 2002)
 file : LapEigenValue.edp
 Load: lg_fem lg_mesh eigenvalue
```

This tools is based on the arpack++ [1] the object-oriented version of ARPACK eigenvalue package [1].
The function EigenValue computes the generalized eigenvalue of $Au = \lambda Bu$ where sigma $=\sigma$ is the shift of the method. The matrix OP is defined with $A - \sigma B$. The return value is the number of converged eigenvalue (can be greater than the number of eigen value nev=)

```
int k=EigenValue(OP,B,nev= , sigma= );
```

where the matrix $OP = A - \sigma B$ with a solver and boundary condition, and the matrix B.
From version 3.31 you have also a functional interface like

```
int k=EigenValue(n,FOP1,FB,nev= , sigma= );
```

where n is the size of the matrix and the now the matrix defined throught function which defined the respectively the matrix product of OP^{-1} and B, like in

```
int n = OP1.n;
func real[int] FOP1(real[int] & u) { real[int] Au=OP^-1*u;return Au;}
func real[int] FB(real[int] & u) { real[int] Au=B*u;return Au;}
```

Note 9.1 *Boundary condition and Eigenvalue Problems*
The locking (Dirichlet) boundary condition is make with exact penalization so we put 1e30=tgv on the diagonal term of the locked degree of freedom (see equation (6.32)). So take Dirichlet boundary condition just on A and not on B. because we solve $w = OP^-1 * B * v$.
If you put locking (Dirichlet) boundary condition on B matrix (with key work **on***) you get small spurious modes* (10^{-30})*, due to boundary condition, but if you forget the locking boundary condition on B matrix (no key work "on") you get huge spurious* (10^{30}) *modes associated to these boundary conditons. We compute only small mode, so we get the good one in this case.*

sym= the problem is symmetric (all the eigen value are real)

nev= the number desired eigenvalues (nev) close to the shift.

value= the array to store the real part of the eigenvalues

ivalue= the array to store the imag. part of the eigenvalues

vector= the FE function array to store the eigenvectors

[1] http://www.caam.rice.edu/software/ARPACK/

rawvector= an array of type `real[int,int]` to store eigenvectors by column. (up to version 2-17).

For real non symmetric problems, complex eigenvectors are given as two consecutive vectors, so if eigenvalue k and $k+1$ are complex conjugate eigenvalues, the kth vector will contain the real part and the $k+1$th vector the imaginary part of the corresponding complex conjugate eigenvectors.

tol= the relative accuracy to which eigenvalues are to be determined;

sigma= the shift value;

maxit= the maximum number of iterations allowed;

ncv= the number of Arnoldi vectors generated at each iteration of ARPACK.

Example 9.17 (lapEignenValue.edp) *In the first example, we compute the eigenvalues and the eigenvectors of the Dirichlet problem on square* $\Omega =]0,\pi[^2$.
The problem is to find: λ, *and* ∇u_λ *in* $\mathbb{R}\times H_0^1(\Omega)$

$$\int_\Omega \nabla u_\lambda \nabla v = \lambda \int_\Omega uv \quad \forall v \in H_0^1(\Omega)$$

The exact eigenvalues are $\lambda_{n,m} = (n^2+m^2), (n,m) \in {\mathbb{N}_*}^2$ *with the associated eigenvectors are* $u_{m,n} = sin(nx) * sin(my)$.
We use the generalized inverse shift mode of the `arpack++` *library, to find 20 eigenvalues and eigenvectors close to the shift value* $\sigma = 20$.

```
//    Computation of the eigen value and eigen vector of the
//    Dirichlet problem on square ]0,π[^2
//    ---------------------------------------
//    we use the inverse shift mode
//    the shift is given with the real sigma
//    ---------------------------------------
//    find λ and u_λ ∈ H_0^1(Ω) such that:
//                          ∫_Ω ∇u_λ∇v = λ ∫_Ω u_λ v, ∀v ∈ H_0^1(Ω)
verbosity=10;
mesh Th=square(20,20,[pi*x,pi*y]);
fespace Vh(Th,P2);
Vh u1,u2;

real sigma = 20;                                        //    value of the shift

                                    //    OP = A - sigma B ; // the shifted matrix
varf  op(u1,u2)= int2d(Th)(  dx(u1)*dx(u2) + dy(u1)*dy(u2) - sigma* u1*u2 )
                    +  on(1,2,3,4,u1=0) ;               //    Boundary condition

varf b([u1],[u2]) = int2d(Th)(  u1*u2 );    //   no Boundary condition see note
9.1
matrix OP= op(Vh,Vh,solver=Crout,factorize=1);  //    crout solver because the
matrix in not positive
matrix B= b(Vh,Vh,solver=CG,eps=1e-20);

                                                        //    important remark:
```

```
                //    the boundary condition is make with exact penalization:
     //    we put 1e30=tgv on the diagonal term of the lock degree of freedom.
         //    So take Dirichlet boundary condition just on a variational form
                                          //    and not on b variational form.
                                         //    because we solve w = OP^-1 * B * v

int nev=20;                 //    number of computed eigen value close to sigma

real[int] ev(nev);                            //    to store the nev eigenvalue
Vh[int] eV(nev);                             //    to store the nev eigenvector

int k=EigenValue(OP,B,sym=true,sigma=sigma,value=ev,vector=eV,
                   tol=1e-10,maxit=0,ncv=0);

//    tol= the tolerance
//    maxit= the maximum iteration see arpack doc.
//    ncv see arpack doc.  http://www.caam.rice.edu/software/ARPACK/
//    the return value is number of converged eigen value.

for (int i=0;i<k;i++)
{
  u1=eV[i];
  real gg = int2d(Th)(dx(u1)*dx(u1) + dy(u1)*dy(u1));
  real mm= int2d(Th)(u1*u1) ;
  cout << " ---- " <<  i<< " " << ev[i]<< " err= "
       <<int2d(Th)(dx(u1)*dx(u1) + dy(u1)*dy(u1) - (ev[i])*u1*u1) << " --- "<<endl;
  plot(eV[i],cmm="Eigen  Vector "+i+" valeur =" + ev[i]  ,wait=1,value=1);
}
```

The output of this example is:

```
   Nb of edges on Mortars  = 0
   Nb of edges on Boundary = 80, neb = 80
 Nb Of Nodes = 1681
 Nb of DF = 1681
Real symmetric eigenvalue problem: A*x - B*x*lambda


Thanks to ARPACK++ class ARrcSymGenEig
Real symmetric eigenvalue problem: A*x - B*x*lambda
Shift and invert mode  sigma=20

Dimension of the system            : 1681
Number of 'requested' eigenvalues  : 20
Number of 'converged' eigenvalues  : 20
Number of Arnoldi vectors generated: 41
Number of iterations taken         : 2

Eigenvalues:
  lambda[1]: 5.0002
  lambda[2]: 8.00074
  lambda[3]: 10.0011
  lambda[4]: 10.0011
  lambda[5]: 13.002
```

```
 lambda[6]: 13.0039
 lambda[7]: 17.0046
 lambda[8]: 17.0048
 lambda[9]: 18.0083
 lambda[10]: 20.0096
 lambda[11]: 20.0096
 lambda[12]: 25.014
 lambda[13]: 25.0283
 lambda[14]: 26.0159
 lambda[15]: 26.0159
 lambda[16]: 29.0258
 lambda[17]: 29.0273
 lambda[18]: 32.0449
 lambda[19]: 34.049
 lambda[20]: 34.0492

---- 0 5.0002 err= -0.000225891 ---
---- 1 8.00074 err= -0.000787446 ---
---- 2 10.0011 err= -0.00134596 ---
---- 3 10.0011 err= -0.00134619 ---
---- 4 13.002 err= -0.00227747 ---
---- 5 13.0039 err= -0.004179 ---
---- 6 17.0046 err= -0.00623649 ---
---- 7 17.0048 err= -0.00639952 ---
---- 8 18.0083 err= -0.00862954 ---
---- 9 20.0096 err= -0.0110483 ---
---- 10 20.0096 err= -0.0110696 ---
---- 11 25.014 err= -0.0154412 ---
---- 12 25.0283 err= -0.0291014 ---
---- 13 26.0159 err= -0.0218532 ---
---- 14 26.0159 err= -0.0218544 ---
---- 15 29.0258 err= -0.0311961 ---
---- 16 29.0273 err= -0.0326472 ---
---- 17 32.0449 err= -0.0457328 ---
---- 18 34.049 err= -0.0530978 ---
---- 19 34.0492 err= -0.0536275 ---
```

9.5 Evolution Problems

`FreeFem++` also solves evolution problems such as the heat equation:

$$\begin{aligned} &\frac{\partial u}{\partial t} - \mu \Delta u = f \quad \text{in } \Omega \times]0,T[, \\ &u(\boldsymbol{x},0) = u_0(\boldsymbol{x}) \quad \text{in } \Omega; \qquad (\partial u/\partial n)\,(\boldsymbol{x},t) = 0 \quad \text{on } \partial\Omega \times]0,T[. \end{aligned} \tag{9.26}$$

with a positive viscosity coefficient μ and homogeneous Neumann boundary conditions. We solve (9.26) by FEM in space and finite differences in time. We use the definition of the partial derivative of the solution in the time derivative,

$$\frac{\partial u}{\partial t}(x,y,t) = \lim_{\tau \to 0} \frac{u(x,y,t) - u(x,y,t-\tau)}{\tau}$$

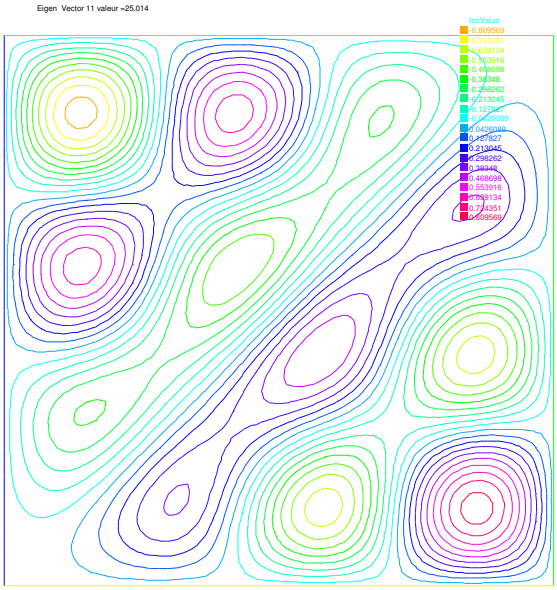

Figure 9.17: Isovalue of 11th eigenvector $u_{4,3} - u_{3,4}$

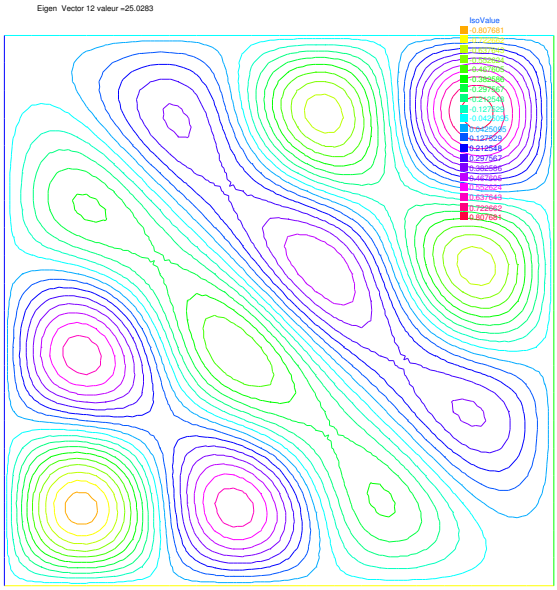

Figure 9.18: Isovalue of 12th eigenvector $u_{4,3} + u_{3,4}$

which indicates that $u^m(x,y) = u(x,y,m\tau)$ will satisfy approximatively

$$\frac{\partial u}{\partial t}(x,y,m\tau) \simeq \frac{u^m(x,y) - u^{m-1}(x,y)}{\tau}$$

The time discretization of heat equation (9.27) is as follows:

$$\begin{aligned} &\frac{u^{m+1} - u^m}{\tau} - \mu \Delta u^{m+1} = f^{m+1} \quad \text{in } \Omega \\ &u^0(\boldsymbol{x}) = u_0(\boldsymbol{x}) \quad \text{in } \Omega; \qquad \partial u^{m+1}/\partial n(\boldsymbol{x}) = 0 \quad \text{on } \partial\Omega, \quad \text{for all } m = 0, \cdots, [T/\tau], \end{aligned} \tag{9.27}$$

which is so-called *backward Euler method* for (9.27). To obtain the variational formulation, multiply with the test function v both sides of the equation:

$$\int_\Omega \{u^{m+1}v - \tau \Delta u^{m+1} v\} = \int_\Omega \{u^m + \tau f^{m+1}\} v\,.$$

By the divergence theorem, we have

$$\int_\Omega \{u^{m+1}v + \tau \nabla u^{m+1} \cdot \nabla v\} - \int_{\partial\Omega} \tau \left(\partial u^{m+1}/\partial n\right) v = \int_\Omega \{u^m v + \tau f^{m+1} v\}.$$

By the boundary condition $\partial u^{m+1}/\partial n = 0$, it follows that

$$\int_\Omega \{u^{m+1}v + \tau \nabla u^{m+1} \cdot \nabla v\} - \int_\Omega \{u^m v + \tau f^{m+1} v\} = 0. \tag{9.28}$$

Using the identity just above, we can calculate the finite element approximation u_h^m of u^m in a step-by-step manner with respect to t.