```
// ================================
// this script computes the eigenvalues of the three problems
// for the elasticity operator
// L:= -DeltaVectorial - alpha grad div = - (alpha+1) grad div + curl curl,
// where DeltaVectorial = grad div - curl curl,
// obtained from the standard Lame operator
// by taking Lame constants to be mu = 1 and alpha = lambda+1
//
// The three problems are
//
// Lu = LambdaD u, u=0 on the boundary (Dirichlet)
//
// Lu = LambdaN u, Tu=0 on the boundary (Neumann)
// (Tu is the stress vector)
//
//
// ================================
//
// input parameters used below:
// (REMEMBER that array enumeration starts from ZERO, NOT ONE)
//
// Nalphas - number of values of alpha for which calculations are done
// alphas - array (0:Nalphas-1) of values of alpha
//
// NeigD - number of Dirichlet eigenvalues to compute
//
//
// Nmesh - number of mesh points on each piece of the boundary
//
// tol - tolerance
//
// Filename - for saving the results
// ================================
//
// output parameters used below:
//
// For each alpha:
//
// EigsD(0:NeigD-1) - Dirichlet eigenvalues
//
// EigsN(0:NeigN-1) - Neumann eigenvalues
//
//
//
// ================================
// set up input variables, other services variables, and arrays
//
int Nalphas = 3;

real[int] alphas(Nalphas);
alphas(0) = 0.5;
alphas(1) = 4.;
alphas(2) = 101.;
```

```
//
int NeigD = 200;
real[int] EigsD(NeigD); // holding array for Dirichlet eigenvalues
//
int NeigN = 250;
real[int] EigsN(NeigN); // holding array for Neumann eigenvalues
//
//
int Nmesh = 50;
//
real tol=1e-7;
//
string Filename = "elastic-square-eigenvalues-online.txt";
// ================================
// set geometry and preliminaries for quadratic forms

border a(t=0,1) { x = t; y = 0; label=1;} // set up geometry for the square
border b(t=0,1) { x = 1; y = t; label=1;}
border c(t=0,1) { x = 1-t; y = 1; label=1;}
border d(t=0,1) { x = 0; y = 1-t; label=1;}
mesh Th=buildmesh(a(Nmesh)+b(Nmesh)+c(Nmesh)+d(Nmesh));
fespace Vh(Th,[P2,P2]);
Vh [u,v],[uu,vv];
real sqrt2=sqrt(2.);
macro epsilon(u1,u2) [dx(u1),dy(u2),(dy(u1)+dx(u2))/sqrt2]
// EOM
// the sqrt2 is because we want: epsilon(u1,u2)'* epsilon(v1,v2) == ε(u) :
   ε(v)<CR>macro div(u,v) ( dx(u)+dy(v) )
// EOM
// two macros are from Example in Femlab++ manual
//
{ofstream Eva(Filename);
Eva << "Nalphas = " << Nalphas << " NeigD = " << NeigD << " NeigN = " << NeigN
<<"\n";
Eva << "alphas = " << alphas << "\n";}
ofstream Eva(Filename, append);
// ================================
// now start looping thorough alphas
for (int ii=0; ii<Nalphas; ii++) {
   real alpha=alphas(ii);
   cout << "alpha = " << alpha << "\n";
   varf
LameD([u,v],[uu,vv])=int2d(Th)((alpha-1)*div(u,v)*div(uu,vv)+2.*(epsilon(u,v)'*epsi
lon(uu,vv)))+on(1,u=0,v=0);
   varf
LameN([u,v],[uu,vv])=int2d(Th)((alpha-1)*div(u,v)*div(uu,vv)+2.*(epsilon(u,v)'*epsi
lon(uu,vv)));
   varf VolumeIn([u,v],[uu,vv])=int2d(Th)(u*uu+v*vv);
   matrix AD=LameD(Vh,Vh,solver=sparsesolver);
   matrix AN=LameN(Vh,Vh,solver=sparsesolver);
   matrix BV=VolumeIn(Vh,Vh);
//
   int kD=EigenValue(AD,BV,sym=true,sigma=0,value=EigsD,tol=tol,maxit=0,ncv=0);
   int kN=EigenValue(AN,BV,sym=true,sigma=0,value=EigsN,tol=tol,maxit=0,ncv=0);
```

```
   for (int mm=0; mm<EigsN.n; mm++) {
      if (abs(EigsN(mm))<tol) {EigsN(mm)=0.;}
   } // fixes small values

   Eva << "alpha = " << alpha << "\n";
   Eva << "EigsD = " << EigsD << "\n";
   Eva << "EigsN = " << EigsN << "\n";
//
// now finish the output
//
   Eva << "================================\n";

}
//assert(nerr==0);
```