Chapter 3

Learning by Examples

This chapter is for those, like us, who don't like to read manuals. A number of simple examples cover a good deal of the capacity of `FreeFem++` and are self-explanatory. For the modeling part this chapter continues at Chapter 9 where some PDEs of physics, engineering and finance are studied in greater depth.

3.1 Membranes

Summary *Here we shall learn how to solve a Dirichlet and/or mixed Dirichlet Neumann problem for the Laplace operator with application to the equilibrium of a membrane under load. We shall also check the accuracy of the method and interface with other graphics packages.*

An elastic membrane Ω is attached to a planar rigid support Γ, and a force $f(x)dx$ is exerted on each surface element $\mathrm{d}x = \mathrm{d}x_1\mathrm{d}x_2$. The vertical membrane displacement, $\varphi(x)$, is obtained by solving Laplace's equation:

$$-\Delta\varphi = f \ \text{ in } \ \Omega.$$

As the membrane is fixed to its planar support, one has:

$$\varphi|_\Gamma = 0.$$

If the support wasn't planar but at an elevation $z(x_1, x_2)$ then the boundary conditions would be of non-homogeneous Dirichlet type.

$$\varphi|_\Gamma = z.$$

If a part Γ_2 of the membrane border Γ is not fixed to the support but is left hanging, then due to the membrane's rigidity the angle with the normal vector n is zero; thus the boundary conditions are

$$\varphi|_{\Gamma_1} = z, \qquad \frac{\partial\varphi}{\partial n}|_{\Gamma_2} = 0$$

where $\Gamma_1 = \Gamma - \Gamma_2$; recall that $\frac{\partial\varphi}{\partial n} = \nabla\varphi \cdot n$. Let us recall also that the Laplace operator Δ is defined by:

$$\Delta\varphi = \frac{\partial^2\varphi}{\partial x_1^2} + \frac{\partial^2\varphi}{\partial x_2^2}.$$

With such "mixed boundary conditions" the problem has a unique solution (see (1987), Dautray-Lions (1988), Strang (1986) and Raviart-Thomas (1983)); the easiest proof is to notice that φ is the state of least energy, i.e.

$$E(\phi) = \min_{\varphi - z \in V} E(v), \quad \text{with} \quad E(v) = \int_\Omega (\frac{1}{2}|\nabla v|^2 - fv)$$

and where V is the subspace of the Sobolev space $H^1(\Omega)$ of functions which have zero trace on Γ_1. Recall that ($x \in \mathbb{R}^d$, $d = 2$ here)

$$H^1(\Omega) = \{u \in L^2(\Omega) \ : \ \nabla u \in (L^2(\Omega))^d\}$$

Calculus of variation shows that the minimum must satisfy, what is known as the weak form of the PDE or its variational formulation (also known here as the theorem of virtual work)

$$\int_\Omega \nabla\varphi \cdot \nabla w = \int_\Omega fw \quad \forall w \in V$$

Next an integration by parts (Green's formula) will show that this is equivalent to the PDE when second derivatives exist.

WARNING Unlike `freefem+` which had both weak and strong forms, `FreeFem++` implements only weak formulations. It is not possible to go further in using this software if you don't know the weak form (i.e. variational formulation) of your problem: either you read a book, or ask help form a colleague or drop the matter. Now if you want to solve a system of PDE like $A(u, v) = 0$, $B(u, v) = 0$ don't close this manual, because in weak form it is

$$\int_\Omega (A(u, v)w_1 + B(u, v)w_2) = 0 \quad \forall w_1, w_2...$$

Example Let an ellipse have the length of the semimajor axis $a = 2$, and unitary the semiminor axis Let the surface force be $f = 1$. Programming this case with `FreeFem++` gives:

Example 3.1 (membrane.edp)

```
//    file membrane.edp
real theta=4.*pi/3.;
real a=2.,b=1.;      //    the length of the semimajor axis and semiminor axis
func z=x;

border Gamma1(t=0,theta)     { x = a * cos(t); y = b*sin(t); }
border Gamma2(t=theta,2*pi)  { x = a * cos(t); y = b*sin(t); }
mesh Th=buildmesh(Gamma1(100)+Gamma2(50));

fespace Vh(Th,P2);                          //    P2 conforming triangular FEM
Vh phi,w, f=1;

solve Laplace(phi,w)=int2d(Th)(dx(phi)*dx(w) + dy(phi)*dy(w))
                - int2d(Th)(f*w) + on(Gamma1,phi=z);
plot(phi,wait=true, ps="membrane.eps");                          //   Plot phi
plot(Th,wait=true, ps="membraneTh.eps");                          //   Plot Th

savemesh(Th,"Th.msh");
```

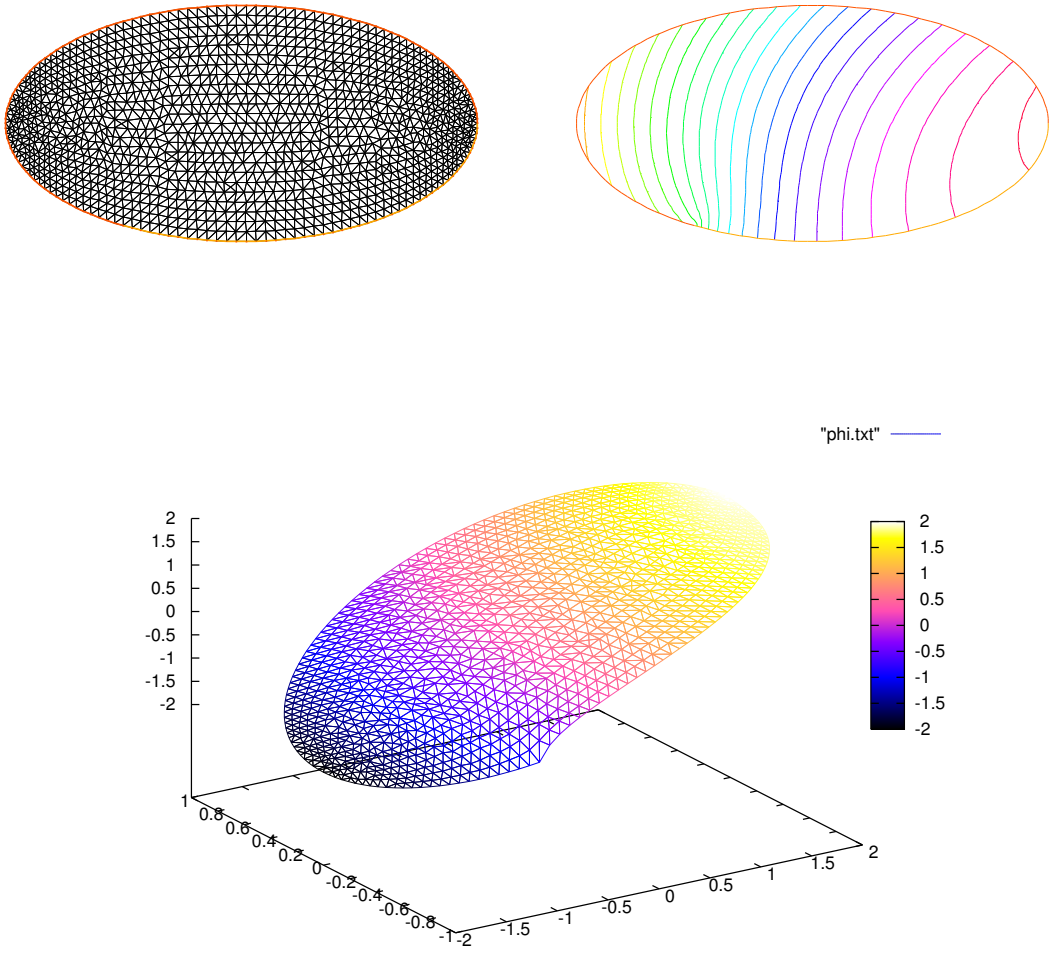

Figure 3.1: Mesh and level lines of the membrane deformation. Below: the same in 3D drawn by `gnuplot` from a file generated by `FreeFem++` .

A triangulation is built by the keyword `buildmesh`. This keyword calls a triangulation subroutine based on the Delaunay test, which first triangulates with only the boundary points, then adds internal points by subdividing the edges. How fine the triangulation becomes is controlled by the size of the closest boundary edges.

The PDE is then discretized using the triangular second order finite element method on the triangulation; as was briefly indicated in the previous chapter, a linear system is derived from the discrete formulation whose size is the number of vertices plus the number of mid-edges in the triangulation. The system is solved by a multi-frontal Gauss LU factorization implemented in the package `UMFPACK`. The keyword plot will display both $\mathbb{T}_h$ and φ (remove `Th` if φ only is desired) and the qualifier `fill=true` replaces the default option (colored level lines) by a full color display. Results are on figure 3.1.

```
plot(phi,wait=true,fill=true);              //   Plot phi with full color display
```

Next we would like to check the results!
One simple way is to adjust the parameters so as to know the solutions. For instance on the unit circle `a=1` , $\varphi_e = \sin(x^2+y^2-1)$ solves the problem when

$$z = 0,\ f = -4(\cos(x^2+y^2-1) - (x^2+y^2)\sin(x^2+y^2-1))$$

except that on Γ_2 $\partial_n\varphi = 2$ instead of zero. So we will consider a non-homogeneous Neumann condition and solve

$$\int_\Omega (\nabla\varphi\cdot\nabla w = \int_\Omega fw + \int_{\Gamma_2} 2w \quad \forall w \in V$$

We will do that with two triangulations, compute the L^2 error:

$$\epsilon = \int_\Omega |\varphi - \varphi_e|^2$$

and print the error in both cases as well as the log of their ratio an indication of the rate of convergence.

Example 3.2 (membranerror.edp)

```
                                                //    file membranerror.edp
verbosity =0;                                   //    to remove all default output
real theta=4.*pi/3.;
real a=1.,b=1.;       //    the length of the semimajor axis and semiminor axis
border Gamma1(t=0,theta)     { x = a * cos(t); y = b*sin(t); }
border Gamma2(t=theta,2*pi) { x = a * cos(t); y = b*sin(t); }

func f=-4*(cos(x^2+y^2-1) -(x^2+y^2)*sin(x^2+y^2-1));
func phiexact=sin(x^2+y^2-1);

real[int] L2error(2);                                  //    an array two values
for(int n=0;n<2;n++)
{
  mesh Th=buildmesh(Gamma1(20*(n+1))+Gamma2(10*(n+1)));
  fespace Vh(Th,P2);
   Vh phi,w;

  solve laplace(phi,w)=int2d(Th)(dx(phi)*dx(w) + dy(phi)*dy(w))
    - int2d(Th)(f*w) - int1d(Th,Gamma2)(2*w)+ on(Gamma1,phi=0);
```

```
  plot(Th,phi,wait=true,ps="membrane.eps");                    //   Plot Th and phi

  L2error[n]= sqrt(int2d(Th)((phi-phiexact)^2));
}

for(int n=0;n<2;n++)
 cout << " L2error " << n << " = "<<  L2error[n] <<endl;

cout <<" convergence rate = "<< log(L2error[0]/L2error[1])/log(2.)  <<endl;
```

the output is

```
 L2error 0 = 0.00462991
 L2error 1 = 0.00117128
 convergence rate = 1.9829
times: compile 0.02s, execution 6.94s
```

We find a rate of 1.93591, which is not close enough to the 3 predicted by the theory. The Geometry is always a polygon so we lose one order due to the geometry approximation in $O(h^2)$

Now if you are not satisfied with the `.eps` plot generated by `FreeFem++` and you want to use other graphic facilities, then you must store the solution in a file very much like in `C++`. It will be useless if you don't save the triangulation as well, consequently you must do

```
    {
      ofstream ff("phi.txt");
      ff << phi[];
     }
   savemesh(Th,"Th.msh");
```

For the triangulation the name is important: it is the extension that determines the format.

Still that may not take you where you want. Here is an interface with gnuplot to produce the right part of figure 3.2.

```
                                                    //    to build a gnuplot data file
{ ofstream ff("graph.txt");
   for (int i=0;i<Th.nt;i++)
   { for (int j=0; j <3; j++)
       ff<<Th[i][j].x  << "    "<< Th[i][j].y<< "  "<<phi[][Vh(i,j)]<<endl;
    ff<<Th[i][0].x  << "    "<< Th[i][0].y<< "  "<<phi[][Vh(i,0)]<<"\n\n\n"
   }
}
```

We use the finite element numbering, where `Wh(i,j)` is the global index of j^{Th} degrees of freedom of triangle number i.
Then open `gnuplot` and do

```
        set palette rgbformulae 30,31,32
        splot "graph.txt" w l pal
```

This works with `P2` and `P1`, but not with `P1nc` because the 3 first degrees of freedom of `P2` or `P2` are on vertices and not with `P1nc`.