

A very brief introduction to eigenvalue computations with FREEFEM

Michael Levitin

Department of Mathematics and Statistics, University of Reading
Reading RG6 6AX, UK
m.levitin@reading.ac.uk michaellevitin.net

Updated and expanded September 2019; updated September 2022

The aim of these brief notes is to show how easily one can learn to compute the eigenvalues of simple boundary value problems numerically without knowing (almost) any programming or even any details of the Finite Element Method. They supplement my mini-course at [Inverse and Spectral Problems for \(Non\)-Local Operators](#) Summer School in Leipzig in 2018.

I Getting and installing FREEFEM

FREEFEM is a free software package for solving initial and boundary value problems for PDEs using the [Finite Element Method \(FEM\)](#). Previous versions of the package were known as FREEFEM++, which explains the name of this file and the web address of the page where it resides. The home page of the package is at

www.freefem.org.

The full HTML documentation is available at

doc.freefem.org

or downloadable as PDF from

doc.freefem.org/pdf/FreeFEM-documentation.pdf

(it is automatically downloaded to your computer during the installation) — warning, it is a long (and sometimes very confusing) document.

In the first instance, install the package for your operation system, following the instructions and links at

doc.freefem.org/introduction/installation.html.

Do not bother with downloading the source files from Github — unless you really know what you are doing.

FREEFEM works a little bit like \LaTeX : you first create an `.edp` script with instructions for FREEFEM, say `example.edp`, then you run FREEFEM on this file by executing a command

```
> FreeFem++ example.edp
```

in your terminal window (there may be shortcuts on your desktop created for this during the installation depending on your operation system; sometimes you need to give the full path to `example.edp`), then you correct the errors, and run FREEFEM again. You repeat the cycle until everything is working!

I am grateful to [Matteo Capoferri](#), a PhD student at University College London, for test-running my instructions and examples. All the remaining errors are of course my own.

As with \LaTeX , there is a number of useful editing programmes which allow you to run the full cycle straight from the editor by pressing a button. They include Emacs or Atom (all platforms), Notepad++ for Windows, or TextMate for Macs (which I use; to save and run a script directly from TextMate press `Option-Shift-R`); all are free. Some details on fine-tuning these editors for use with FREEFEM are at the bottom of FREEFEM installation guide page.

My explanations of FREEFEM syntax and use will be based on the script `example-ML.edp` which you should download from

michaellevitin.net/FreeFem++.html.

2 FREEFEM principles — weak form

FREEFEM works with a *weak form* of an eigenvalue problem, therefore every spectral problem in FREEFEM has the form

$$a(u, v) = \lambda b(u, v). \quad (1)$$

where a and b are some quadratic forms, λ is a spectral parameter, and some boundary conditions may have to be additionally imposed. Here are several examples, where $\Omega \in \mathbb{R}^2$ is a bounded domain with boundary $\partial\Omega$.

2.1 Neumann Laplacian

Consider

$$-\Delta u = \lambda u \quad \text{in } \Omega, \quad \left. \frac{\partial u}{\partial n} \right|_{\partial\Omega} = 0.$$

Multiplying the equation by any $v \in H^1(\Omega)$ and integrating by parts, we arrive at

$$\langle \text{grad } u, \text{grad } v \rangle = \lambda \langle u, v \rangle, \quad (2)$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product in $L_2(\Omega)$. Thus, in this case

$$\begin{aligned} a(u, v) &= \langle \text{grad } u, \text{grad } v \rangle = \iint_{\Omega} \text{grad } u \cdot \text{grad } v \, dx dy, \\ b(u, v) &= \iint_{\Omega} uv \, dx dy \end{aligned} \quad (3)$$

(I omit complex conjugation whenever possible for simplicity).

2.2 Dirichlet Laplacian

Consider

$$-\Delta u = \lambda u \quad \text{in } \Omega, \quad u|_{\partial\Omega} = 0.$$

Multiplying the equation by any $v \in H_0^1(\Omega)$ and integrating by parts, we again arrive at (2), (3), but now only for v which satisfy additionally $v|_{\partial\Omega} = 0$.

2.3 Robin Laplacian

Consider, for a given $\gamma \in \mathbb{R}$,

$$-\Delta u = \lambda u \quad \text{in } \Omega, \quad \left(\frac{\partial u}{\partial n} - \gamma u \right) \Big|_{\partial\Omega} = 0.$$

Acting as above and multiplying by $v \in H^1(\Omega)$, we arrive at (2) with b as in (3) and

$$a(u, v) = \langle \text{grad } u, \text{grad } v \rangle - \gamma \langle u, v \rangle_{\partial\Omega} = \iint_{\Omega} \text{grad } u \cdot \text{grad } v \, dx dy - \gamma \int_{\partial\Omega} uv \, ds, \quad (4)$$

for any $v \in H^1(\Omega)$, where $\langle \cdot, \cdot \rangle_{\partial\Omega}$ denotes the inner product in $L_2(\partial\Omega)$.

2.4 Steklov problem

The Steklov eigenvalue problem has the form

$$-\Delta u = 0 \quad \text{in } \Omega, \quad \left(\frac{\partial u}{\partial n} - \lambda u \right) \Big|_{\partial\Omega} = 0, \quad (5)$$

where the spectral parameter λ now appears in the boundary condition. The weak form of this problem is given by (2) with a as in (3), and

$$b(u, v) = \langle u, v \rangle_{\partial\Omega} = \int_{\partial\Omega} uv \, ds.$$

3 The structure of an .edp script

In this section, I will describe some commands of the FREEFEM programming language, using my example file `example-ML.edp` as a guide. You can create your own files using the constructions from this and next section by mimicking the example.

Every .edp script for solving an eigenvalue problem consists, roughly speaking, of the following parts:

- declaration and initialisation of variables;
- description of the boundary;
- meshing;
- description of the quadratic form;
- converting the weak form (1) into a matrix eigenvalue problem

$$AU = \lambda BU, \quad (6)$$

where A, B are matrices corresponding to the forms a and b ;

- solving (6) and post-processing,

with various plotting/output commands inserted where required.

Let us look at the file `example-ML.edp` line by line. The script will compute the eigenvalues of the Neumann Laplacian for a rectangle $[0, L\pi] \times [0, \pi]$. The first 11 lines of the script are just the comments, in fact every line starting with the double slash (or any text at the end of the line after a double slash) is ignored by FREEFEM, and is there just for the ease of reading the script. By the way, empty lines are also ignored.

The actual script starts with the declarations in lines 12–23. Let us look at them line by line, ignoring the comments.

The line

```
12 | real L=0.5;
```

declares a variable `L` to be real, and assigns value `0.5` to it. Variable names can be of arbitrary length and consist of upper- and lower-case letters, numbers, and underscore, and start with a letter. One can declare several variables at once, not necessarily assigning any values to them, for example one can have

```
real L1, L2=0.5, L3;
```

to define three real variables `L1` (unassigned), `L2` (with the value `0.5`), and `L3` (unassigned).

IMPORTANT - every individual command should end with the semicolon!

The line

```
14 int npoints= 20;
```

declares a variable `npoints` to be integer, and assign value 20 to it; this variable will be used later.
The line

```
19 int N=50;
```

declares a variable `N` to be integer, and assigns value 50 to it. This denotes the number of eigenvalues we want to compute.
The line

```
20 real [int] Evalues (N);
```

declares `Evalues` to be an array of real numbers of length `N` indexed by integers from 0 to $N-1$, which will eventually hold the eigenvalues. Note that interchanging lines 19 and 20 would give an error — we cannot declare an array until we know its size.

The last declaration in line

```
23 real t;
```

declares `t` as another real variable.

I now proceed to define the boundary of our domain. The boundary should be defined as a collection of smooth parametrised curves swept in counterclockwise direction. We first define each individual curve in

```
29 border lower(t=0,L) { x = pi*t; y = 0; label=1; };
30 border right(t=0,1) { x = pi*L; y = pi*t; label=1; };
31 border upper(t=L,0) { x = pi*t; y = pi; label=1; };
32 border left(t=1,0) { x = 0; y = pi*t; label=1; };
```

A definition of a boundary piece usually takes the form

```
border border_name(t=t0,t1) {x=a_function(t); y = another_function(t);
    label=name_or_number;}
```

to define a parametric curve. Note that we can have $t_1 < t_0$ as in lines 31 and 32. The part `label=...` is optional — but labels are important if we want to integrate over the boundary, or impose different boundary conditions on different boundary pieces, allowing us to group them together.

Once all the boundary pieces are defined, I create the mesh by executing `buildmesh` command in

```
38 mesh Th=buildmesh(lower(npoints*L*pi)+right(npoints*pi)+upper(npoints*L
    *pi)+left(npoints*pi));
```

and assigning the output to variable `Th` declared to be a `mesh`. The general format of `buildmesh` command is

```
buildmesh(boundary1(points1)+...+boundaryX(pointsX));
```

where each `boundary1, ..., boundaryX` has been previously defined as a `border`, and `points1, ..., pointsX` indicate how many mesh points to place on each border.

The command

```
42 plot(Th, wait=1);
```

plots the mesh (and the geometry) and waits one second before continuing.

I then declare the finite element space V_h on the mesh \mathcal{T}_h by

```
45 fespace Vh (Th , P2) ;
```

There is a large library of finite elements to choose from; for our purposes either P2 or P1 would suffice. I then declare variables u and v to be the elements of this space by

```
46 Vh u , v ;
```

The next commands

```
49 varf a(u,v)=int2d(Th)( dx(u)*dx(v) + dy(u)*dy(v) );
50 varf b(u,v)=int2d(Th)(u*v) ;
```

define the quadratic forms (`varf`) a and b according to (3) using built-in differentiation operators dx and dy , and the 2d-integration function `int2d`.

I then create the matrices A and B using the quadratic forms a and b (do not worry about `solver=sparsesolver` option at this stage):

```
53 matrix A=a(Vh , Vh , solver=sparsesolver) ;
```

```
55 matrix B=b(Vh , Vh) ;
```

I then create an array to hold the eigenfunctions (if we want to plot them later, for example; if you are only interested in eigenvalues, you may skip this step).

```
58 Vh [int] Efunctions (N) ;
```

I've now assembled all the necessary ingredients to actually compute the eigenvalues using the command

```
61 int k=EigenValue (A , B , sym=true , sigma=0 , value=Evalues , vector=Efunctions) ;
```

The input parameters here are matrices A and B assembled earlier, the option `sym=true` indicating that our problem is symmetric, the option `sigma=0` telling the programme to look for eigenvalues close to 0, and the instructions where to store the eigenvalues and the eigenvectors. The output parameter k is the number of eigenvalues actually computed — in most cases it will be just the length of array `Evalues`. If you do not need the eigenfunctions, omit `vector=Efunctions` option.

Finally, I print the eigenvalues on the screen using

```
64 cout << Evalues ;
```

and plot the contour plot of the 6th eigenfunction using

```
67 plot (Efunctions [5]) ;
```

(remember — arrays are numbered starting from zero!)

If everything worked correctly, you will see two plots and an output similar to

```
-- mesh: Nb of Triangles = 4294, Nb of Vertices 2241
Real symmetric eigenvalue problem: A*x - B*x*lambda
50
```

```

-3.108917267e-14 1.000000008 4.000000489 4.000000498 5.000001121
8.000004624 9.000005517 13.00001963 16.00003071 16.00003138
17.000038 20.0000669 20.00006906 25.00011858 25.0001404
29.00019679 32.00029074 36.00035258 36.00035443 37.00036793
40.00050424 40.00051836 41.00062122 45.00078621 49.00090374
52.00122384 52.00127188 53.00114965 61.0020541 64.00197654
64.00200986 65.00198599 65.00242175 68.00240693 68.00246894
72.00342198 73.00321228 80.0043361 80.00439092 81.00407524
85.00456212 85.00562486 89.00617836 97.00770116 100.0075649
100.007774 100.0086516 100.0092472 101.0077133 104.0083766
times: compile 0.009288s, execution 1.81942s, mpirank:0

```

You can see that the precision is pretty good! If you want a better accuracy, choose a larger number of mesh points by increasing `npoints`.

4 Extensions and modifications

One can of course do much more with FREEFEM. I'll give some examples below, listing in each case modifications to my original file `example-ML.edp`.

4.1 Curvilinear boundaries

Replace line 29, for example, by

```
border lower(t=0,L) { x = pi*t; y = 0.5*sin(2*pi*t/L); label=1; };
```

As this piece of the boundary is now longer than in the original example, you may wish to increase the number of mesh points on this piece by using, for example,

```
mesh Th=buildmesh(lower(2*npoints*L*pi)+...);
```

in line 39.

4.2 Holes

To cut out a circular hole centred at $(L, 0.5)$ with the radius $r = \min(L/3, 1/3)$ from our rectangle, add the lines

```
real r; if (L>1) r=1./3.; else r=L/3.;
border hole(t=0,2*pi) {x=L+r*cos(t); y=0.5+r*sin(t); label=1;}
```

and change the original line 39 to

```
mesh Th=buildmesh(lower(npoints*L*pi)+right(npoints*pi)+upper(npoints*
L*pi)+left(npoints*pi)+hole(-npoints*2*pi*r));
```

Note that we change the orientation of the border piece `hole` using a *negative* number of points as its parameter in the last line.

4.3 Dirichlet problem

To impose the Dirichlet boundary condition on the whole boundary, simply change the original line 49 to

```
varf a(u,v)=int2d(Th)( dx(u)*dx(v) + dy(u)*dy(v) )+on(1,u=0);
```

To impose the Dirichlet condition only on some boundary pieces, give them different labels (for example, 1, 2, 3, 4), and then use, say, `on(1,2,u=0)` to impose the Dirichlet conditions on pieces labeled 1 and 2. Alternatively, you can group boundary pieces together by giving several of them the same label.

Note that the boundary condition is applied only to u and only in the form a .

4.4 Steklov problem

To solve the Steklov eigenvalue problem (5), change the original line 50 to

```
varf b(u,v)=int1d(Th,1)(u*v);
```

(note the format for one-dimensional integrals). Mixed Neumann-Dirichlet-Steklov problems can be treated in the same manner.

4.5 Spectrum of the Laplace–Beltrami operator on the unit sphere \mathbb{S}^2

It is not immediately clear from the previous examples how to find the eigenvalues of the Laplace–Beltrami operator on the sphere — even if we write the quadratic forms in polar coordinates, implementing the periodicity conditions is non-trivial. Instead we will use the following trick which reduces the problem to two problems on the unit disk.

First, we note that as the sphere is symmetric with respect to the equator, each eigenfunction is either symmetric or anti-symmetric with respect to the equator, that is, satisfies on the equator either the Neumann or the Dirichlet boundary condition. Thus the spectrum of the Laplace–Beltrami operator on \mathbb{S}^2 is the union of the spectra of the Neumann and Dirichlet Laplacians on the hemisphere. To compute those, we use the stereographic projection onto the unit disk $\mathbb{D} := \{(x, y) : x^2 + y^2 < 1\}$ reducing the problems to the problem

$$-\Delta u = \frac{4}{(1+x^2+y^2)^2} \lambda u, \tag{7}$$

with either Neumann or Dirichlet condition imposed on $\partial\mathbb{D}$; $-\Delta$ in the left-hand side of (7) is the usual (Cartesian) Laplacian.

To solve this in FREEFEM, first replace the original lines 29–33 by

```
border d(t=0,2*pi) { x = cos(t); y = sin(t); label=1; };
```

Then change the original line 39 to

```
mesh Th=buildmesh(d(npoints*L*2*pi));
```

and original line 50 to

```
varf b(u,v)=int2d(Th)(4/(x^2+y^2)^2*u*v);
```

Now run the script to obtain the Neumann eigenvalues on the hemisphere; then change according to §4.3 to get the Dirichlet eigenvalues. Combining these together, you should approximately get the eigenvalues $\lambda = k(k+1)$, $k = 0, 1, 2, \dots$ of the Laplace-Beltrami operator on the sphere, each with multiplicity $2k+1$, that is, the sequence 0, 2, 2, 2, 6, 6, 6, 6, 12, \dots , 12, 20, \dots