AARMS-CRM Workshop on NA of SPDEs, July 2016

Lab 2: Singularly Perturbed PDEs

Niall Madden (Niall.Madden@NUIGalway.ie)

(If you are not familiar with MATLAB, you should read through the MATLAB Primer first)

This lab involves solving a singularly perturbed reaction-diffusion equation in two dimensions using a finite difference technique.

In §6 of the short course, we required certain compatibility conditions to hold in order for the analysis to be valid. We will experiment with these conditions to see if, in practice, they are necessary. We will also experiment with using a graded mesh to solve the PDE robustly.

The choice of finite differences for this lab is due to convenience. Implementing a finite element method would involve additional issues regarding quadrature and error estimation, which we cannot cover in the time allowed. If you are interested in FEMs, ask Niall for some FE code that corresponds to the FD code presented here.

1 The finite difference method

We will solve the following PDE numerically:

$$-\varepsilon^2(u_{xx}+u_{yy})+b(x,y)u=f(x,y) \quad \text{on } \Omega:=(0,1)^2, \tag{1a}$$

and with the boundary conditions

$$u(x,y)=g(x,y) \qquad \text{for } (x,y)\in\partial\Omega. \tag{1b}$$

Denote the mesh points of an arbitrary rectangular mesh as (x_i,y_j) for $i,j\in\{0,1,\ldots,N\}$, write the local mesh widths as $h_i=x_i-x_{i-1}$ and $k_j=y_j-y_{j-1}$, and let $\bar{h}_i=(x_{i+1}-x_{i-1})/2$, and $\bar{k}_j=(y_{j+1}-y_{j-1})/2$.

The scaled (symmetrised) 5-point, second-order, central difference operator (discrete Laplacian) is

$$\Delta^N:=\begin{bmatrix} \dfrac{\bar{k}_j}{h_i} & -\left(\bar{k}_j\left(\dfrac{1}{h_i}+\dfrac{1}{h_{i+1}}\right)+\bar{h}_i\left(\dfrac{1}{k_j}+\dfrac{1}{k_{j+1}}\right)\right) & \dfrac{\bar{k}_j}{h_{i+1}} \\ & \dfrac{\bar{h}_i}{k_{j+1}} & \\ & \dfrac{\bar{h}_i}{k_j} & \end{bmatrix}. \tag{2}$$

Then the finite difference scheme for the non-boundary points is

$$L^N:=-\varepsilon^2\Delta^N+\bar{h}_i\bar{k}_j b(x_i,y_j)=\bar{h}_i\bar{k}_j f(x_i,y_j). \tag{3}$$

2 Ordering the unknowns

To realise the method above, we have to write it as a system of $(N+1)^2$ linear equations: $AU=B$.

One of the required steps is to choose a numbering of the grid variables, so that they can be expressed as entries in a vector. We will take the usual choice – lexicographical ordering.

It's useful to be adept at switching between

- "*grid ordering*": the FD solution is expressed as a matrix. In MATLAB, if U is a $(N+1)\times(N+1)$ matrix, then the approximate solution at (x_i,y_j) is represented as `U(i+1,j+1)`. (Recall that MATLAB indexes all arrays from 1).
- *lexicographical ordering*: the FD solution is expressed as a vector. In MATLAB, if U is a vector with $(N+1)^2$ entries, the approximate solution at (x_i,y_j) is represented by `U(k)` where $k=i+1+j(N+1)$.
- We need both these representations: the vector one is the solution to the linear system, and the matrix one is used, for example, when plotting the solution. To switch between them, try

```
U = reshape(U, [], N+1)'; % vector-to-matrix
U = reshape(U', [], 1);   % matrix-to-vector
```

3 The FD solver

The method described above is implemented in the MATLAB function `Solve_2DRD.m`, which can be downloaded from `http://www.maths.nuigalway.ie/~niall/NASPDEs2016` This is a MATLAB function file. If N is a positive integer, then...

```
x = linspace(0,1,N+1)'; % x-mesh
y = linspace(0,1,N+1)'; % y-mesh
b = @(x,y)(x*0+y*0+1);
f = @(x,y,epsilon)(x*0+y*0+1);
g = @(x,y)(x.*y*0);
Solve_2DRD(1/10, x, y, b, f, b);
```

will return the numerical solution to

$$-\frac{1}{100}\Delta u + u = 1 \qquad \text{on } (0,1)^2,$$

with homogeneous Dirichlet boundary conditions, and solved on a uniform mesh with N intervals in each coordinate direction.

Read this code to understand how it works. Some aspects are non-obvious, mainly due to the banded structure of the system matrix.

The test harness `Test_2DRD.m` runs this programme for various values of N and ε. To run it, you'll also need

- `Problem_Data.m`, which stores the true solution to an artificially created problem that features layers near $x = 0$ and $y = 0$ (only), as well as the corresponding right-hand side.
- From Lab 1: `Make_1D_Fitted_Mesh.m`, which can generate a uniform or Shishkin mesh, as required.

The error is computed using this true solution. Verify that the code gives the expected results.

4 The double mesh principle

The test case provided for in `Problem_Data.m` is limited in its uses. If we decide to change the problem data, we have to construct a new solution. Often this is inconvenient or impossible. So it can be good practice to use the *double mesh principle*:

- Given a mesh $\bar{\Omega}^N$ with N intervals in each coordinate direction, construct a new one with $2N$ intervals in each direction, by bisecting each interval of the original one.
- Solve the problem on this mesh, and denote it $\tilde{U}$.
- Estimate the error as $\|U - \tilde{U}\|_{\bar{\Omega}^N}$.

This is known to give a reliable estimate for the rate of convergence. (*Aside: However, it underestimates the error by a factor of 4/3. Question: why? Hint: Think extrapolation).*

Modify the code to estimate the error using the double mesh principle.

5 Compatibility conditions

In Section 6 we reviewed the compatibility conditions derived in [Han and Kellogg, 1990]. As well as requiring that $f, b \in \mathcal{C}^{2,\alpha}(\bar{\Omega})$, the $g_i \in \mathcal{C}^{4,\alpha}([0,1])$, at $c_1 = (0,0)$ we needed these are

$$g_1 = g_2, \tag{4a}$$

$$-\varepsilon^2\big(\frac{\partial^2}{\partial x^2}g_1 + \frac{\partial^2}{\partial y^2}g_2\big) + b g_1 = f,^1 \tag{4b}$$

$$\frac{\partial^2}{\partial x^2}\big(-\varepsilon^2\frac{\partial^2}{\partial x^2}g_1 + b g_1 - f\big) = \frac{\partial^2}{\partial y^2}\big(-\varepsilon^2\frac{\partial^2}{\partial y^2}g_2 + b g_2 - f\big). \tag{4c}$$

Conditions analogous to (4) are satisfied at the other corners. Then $u \in \mathcal{C}^{4,\alpha}$.

Construct examples that violate these conditions in various ways, and determine the impact they have on the order of convergence.

[1] Actually, g_1 and g_2 are functions of a single variable, x and y respctively, but it is notationally convienent to express these ordinary derivatives as partial derivaties, particularly in (??).

6 A graded mesh

So far we have studied only the piecewise uniform mesh of Shishkin. There are many alternatives, but perhaps the best known is the graded mesh of Bakhvalov [Bakhvalov, 1969]. It is described in [Linß, 2010, §2.1.1] as being the mesh with the generating function: ψ, defined as

$$\psi(t) = \begin{cases} \chi(t) := -\dfrac{\sigma\varepsilon}{\beta}\ln\left(1 - t/q\right), & \text{for } t \in [0, \tau_B], \\ \phi(t) := \chi(\tau_B) + \chi'(\tau_B)(t - \tau_B), & \text{for } t \in [\tau_B, 1/2], \\ 1 - \psi(1 - t), & \text{for } t \in (1/2, 1], \end{cases}$$

where the Bakhvalov mesh transition point τ_B is chosen so that $\psi \in \mathcal{C}^1[0,1]$. The mesh parameter q controls the proportion of the mesh points in the layer regions (so $q = 1/3$ is a typical value). The parameter σ needs to be at least as large as the formal order of the scheme. Taking, e.g, $\sigma = 2.5$ for this test problem should give reasonable results.

There is a minor difficulty in implementing this mesh: one must solve a nonlinear problem to obtain τ_B so that $\psi \in \mathcal{C}^1$.

This is implemented as `Make_Bakhvalov_Mesh.m`. Incorporate this into your code, and compare with the accuracy obtained using the Shishkin mesh.

7 References

References

[Bakhvalov, 1969] Bakhvalov, N. (1969). Towards optimization of methods for solving boundary value problems in the presence of boundary layers. *Zh. Vychisl. Mat. i Mat. Fiz.*, 9:841–859.

[Han and Kellogg, 1990] Han, H. and Kellogg, R. B. (1990). Differentiability properties of solutions of the equation $-\epsilon^2\Delta u + ru = f(x, y)$ in a square. *SIAM J. Math. Anal.*, 21(2):394–408.

[Linß, 2010] Linß, T. (2010). *Layer-adapted meshes for reaction-convection-diffusion problems*, volume 1985 of *Lecture Notes in Mathematics*. Springer-Verlag, Berlin.