1 Exercise: 1-D heat conduction with finite elements

Reading

This finite element example is based on *Hughes* (2000), sec. 1.1-1.15.

1.1 Implementation of the 1-D heat equation example

In the previous two sections, we considered the example PDE

$$\frac{\partial^2 u}{\partial x^2} + s = 0 \tag{1}$$

on the domain $x \in [0;1]$, $u(x)$, $s(x)$, and subject to essential (Dirichlet) boundary condition $u(0) = g$ on the left, and natural (Neumann) BCs, $\frac{\partial u(1)}{\partial x} = h$ on the right. Equation (1) may be considered a general version of the steady-state heat equation

$$\frac{\partial^2 T}{\partial x^2} + H = 0 \tag{2}$$

with sources $s = H$, for example.

See sec. **??**, but in brief: If we have n elements between $n+1$ global nodes, the weak form of eq. (1) can be written for each global node A as

$$\sum_B a(N_A, N_B) d_B = (N_A, s) + N_A(1)h - a(N_A, \hat{N}_1)g. \tag{3}$$

Here, N_A are the shape functions in the interior, B is another global node, and $\hat{N}_1$ the boundary shape function for the essential boundary condition g. This can be further abbreviated by

$$K_{AB} = a(N_A, N_B) = \int_0^1 \frac{\partial N_A}{\partial x}\frac{\partial N_B}{\partial x} dx \tag{4}$$

$$F_A = (N_A, s) + N_A(1)h - a(N_A, \hat{N}_1)g \tag{5}$$

$$= \int_0^1 N_A s dx + \delta_{A,n+1} h - \left(\int_0^1 \frac{\partial N_A}{\partial x}\frac{\partial \hat{N}_1}{\partial x} dx \right) g, \tag{6}$$

where we have used the definitions of the bi-linear forms $a(\cdot,\cdot)$ and $(\cdot,\cdot)$ from before, and the Kronecker delta

$$\delta_{i,j} = \begin{cases} 1 & \text{if} \quad i = j \\ 0 & \text{else} \end{cases} \tag{7}$$

is used for the flux boundary condition (also see *Hughes*, 2000, chap 1). Note that it is sometimes helpful to not think about nodes but about elements. The weak form of the equations are satisfied on average per element, and by constructing an appropriate mapping/numbering we can easily go from a single element to a 1D or 2D domain.

The approximate solution of u after discretization of the weak form is given by

$$\tilde{u}(x) = \sum_{A=2}^{n+1} d_A N_A(x) + \hat{N}_1 g = \sum d_A N_A(x), \tag{8}$$

where the latter summation implies choosing the boundary shape function and BC if needed. The vector $\underline{d} = \{d_A\}$ values have to be obtained by solution of the matrix equation

$$\underline{\underline{K}}\,\underline{d} = \underline{F}, \tag{9}$$

with $\underline{\underline{K}} = \{K_{AB}\}$ and $\underline{F} = \{F_A\}$.

We discussed previously how the integration over the domain can be broken down into summation over integrals over each element (see sec. **??**). This integration is most easily performed in a local coordinate system $-1 \leq \xi \leq 1$ between the two nodes of each element, which has a mapping to the corresponding, global coordinate interval $[x_A; x_{A+1}]$. We can also express the shape functions as $x(\xi)$,

$$x(\xi) = \sum_A N_A(\xi) x_A \quad \text{and} \quad u(\xi) = \sum N_A(\xi) d_A. \tag{10}$$

The global $\underline{\underline{K}}$ matrix and the $\underline{F}$ vector are then assembled by looping over all elements $1 \leq e \leq n$ and adding each element's contribution for shared nodes. By change of integration variables and the chain rule, those elemental contributions follow as

$$\underline{\underline{k}}^e = \frac{1}{\Delta x} \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}, \tag{11}$$

where Δx is the element size, $x_{A+1} - x_A$, and for the force term

$$\underline{f}^e = \frac{\Delta x}{6} \begin{pmatrix} 2s_1 + s_2 \\ s_1 + 2s_2 \end{pmatrix} + \begin{cases} k_{a1} g & \text{for} \quad e = 1 \\ \delta_{a,n+1} & \text{for} \quad e = n \\ 0 & \text{else} \end{cases} \tag{12}$$

where we have assumed that the source function s varies linearly over each element, and s_1 and s_2 are the contributions from each local node a within the element from $s(x)$. After assembly, one needs to ensure that each row of the global $\underline{\underline{K}}$ matrix that corresponds to a fixed value (Dirichlet) boundary condition, will only have a diagonal entry and the other columns for this row are zero.

1.2 Exercises

a) Download `heat1dfe.m`, and all helper routines for this section. Read through the implementation of what is summarized above, `heat1dfe.m`, and understand this code.

b) Fill in the blanks in `heat1dfe.m` and experiment with a solution of eq. (1) for $n = 3$ elements.

   (i) Print out the stiffness matrix (`full(stiffness)`) to appreciate its banded structure. Does this look familiar to you?

   (ii) Choose the MATLAB solver (`solver=0`) and plot the finite element solution at the nodes, as interpolated within the elements, and compare with the analytical solution.

   For this, first plot the error as a function of x. Then, define an overall error metric, change the resolution succesively, and evaluate the error as a function of grid size, for example.

   (iii) Reads the following section on solving linear systems of equations, if you have time, and work on the examples (optional).

2 Solution of large, sparse linear systems of equations

Parts of this exercise are based on *Zhong* (2008).

The finite element method and implicit finite difference methods quickly leads to very large, linear systems of equations

$$\underline{\underline{K}}\,\underline{d} = \underline{F} \tag{13}$$

whose solution can be quite involved. Ideally, we would hand off the solution of eq. (13) to a computational scientist and use a "black box" solver. However, practice shows that the nature of the physical problem and the best solution method are often intertwined.

Choosing a different solver might also allow addressing larger, *e.g.* 3-D, problems because of improved efficiency. Moreover, it is very hard to make solvers bullet-proof and one often encounters problematic (*e.g.* unstable, or no convergence) performance in practice.

Linear systems of equations also arise in other fields of geophysics, such as inverse theory, and some exposure to computational linear algebra is needed to fully understand the `MILAMIN` (*Dabrowski et al.*, 2008) finite element implementation which we will use later. We therefore digress a bit here. If your research has you deal with matrices a lot, *Golub and Van Loan* (1996) is a classic numerical linear algebra text that might come in handy.

2.1 Direct solvers

For the finite element method, we can always write our problem in the form of eq. (13), where $\underline{\underline{K}}$ is a square, $n \times n$ matrix. A general strategy to solve eq. (13) is then LU decomposition

$$\underline{\underline{K}} = \underline{\underline{L}}\,\underline{\underline{U}}, \tag{14}$$

where $\underline{\underline{L}}$ and $\underline{\underline{U}}$ are lower and upper triangular matrices, respectively, which only have zeros in the other part of the matrix. The solution of eq. (13) can then be obtained efficiently from

$$\underline{\underline{K}}\,\underline{d} = \underline{\underline{L}}\,\underline{\underline{U}}\,\underline{d} = \underline{F} \tag{15}$$

by solving for $\underline{y} = \underline{\underline{L}}^{-1}\underline{F}$ and then $\underline{d} = \underline{\underline{U}}^{-1}\underline{y}$, because the inverse of $\underline{\underline{U}}$ and $\underline{\underline{L}}$ are computationally fast to obtain. LU is often how general matrix inversion is implemented on a computer.

For most FE problems, the $\underline{\underline{K}}$ matrix will also be sparse and banded. Special algorithms exist to exploit this feature such that the run time is ideally dominated by the number of non-zero entries of $\underline{\underline{K}}$, rather than the full size. Moreover, if $\underline{\underline{K}}$ is symmetric and positive definite, as in our example above, we can use the Cholesky decomposition for which $\underline{\underline{U}} = \underline{\underline{L}}^T$ and computations are twice as fast as for the general LU case. However, for complex, 3-D FE problems, current computational limitations often prohibit the use of direct solvers which is why iterative methods which do not require matrix decomposition or inversion, are used.

Notes:

- Symmetry means $\underline{\underline{K}} = \underline{\underline{K}}^T$, where $\underline{\underline{K}}^T$ is the transpose, $K^T_{ij} = K_{ji}$.
- Positive definite means that $\underline{c}^T \underline{\underline{K}}\, \underline{c} > 0$ for any non-zero $\underline{c}$. Graphically, this corresponds for a 2×2 matrix to a well defined minimum (lowest) point in a curved landscape, which is important for iterative methods (*e.g.* *Shewchuk*, 1994).
- Positive definite, symmetric matrices also arise in least-squares problems in geophysical inversions (*e.g.* seismic tomography, see for example *Boschi and Dziewoński*, 1999, for a nice introduction to linear algebra in this framework).
- Least-squares means that we wish to solve
$$\underline{\underline{A}}\, \underline{x} = \underline{b} \tag{16}$$
in the sense that $|\underline{\underline{A}}\, \underline{x} - \underline{b}| = \min$, *i.e.* deviations from the true solution are minimized, for a matrix $\underline{\underline{A}}$ that may be under-determined, *i.e.* not simply invertible. It can be shown that the general least squares solution $\underline{x}_{LS}$ is given by
$$\underline{x}_{LS} = \left(\underline{\underline{A}}^T \cdot \underline{\underline{A}}\right)^{-1} \cdot \underline{\underline{A}}^T \cdot \underline{d}, \tag{17}$$
where $\left(\underline{\underline{A}}^T \cdot \underline{\underline{A}}\right)^{-1}$ is the generalized inverse (which exists even if the inverse of $\underline{\underline{A}}$, $\underline{\underline{A}}^{-1}$, does not exist because $\underline{\underline{A}}$ is singular). $\underline{\underline{A}}^T \cdot \underline{\underline{A}}$ is symmetric and positive definite, meaning that Cholesky is also the method of choice for direct approaches to find $\underline{x}_{LS}$.

2.2 Iterative solvers

2.2.1 Jacobi method

The simplest iterative solution of eq. (13) is given by the Jacobi method. If $\underline{\underline{K}}$ is LU decomposed and we write the diagonal matrix (only non-zero along diagonal) of $\underline{\underline{K}}$ as $\underline{\underline{D}}$, then an iterative solution for $\underline{d}$ starting from an initial guess $\underline{d}^1$ (*e.g.* $\underline{0}$) can be obtained from
$$\underline{\underline{D}}\, \underline{d}^{i+1} = \underline{F} - (\underline{\underline{U}} + \underline{\underline{L}})\underline{d}^i, \tag{18}$$
where the iteration is over i and is stopped once $\underline{d}^{i+1}$ is not changing more than some tolerance from the previous solution estimate $\underline{d}^i$. On an element by element basis, this can be written as
$$d_j^{i+1} = \frac{1}{K_{jj}} \left(F_j - \sum_{l=1,\, l \neq j}^{n} K_{jl} d_l^i \right) \tag{19}$$
where the summation is over all l but for $l = j$. The Jacobi method following eq. (19) is implemented in `jacobi.m`. It serves mainly illustrative purposes but is guaranteed to converge, albeit slowly (see below), if $\underline{\underline{K}}$ is "diagonally dominant" which is satisfied strictly when the absolute value of the diagonal elements is larger than the sum of the absolute values of each row.

2.2.2 Gauss-Seidel method

An improvement over the Jacobi method is the Gauss-Seidel (GS) approach, where the iterative rule is

$$(\underline{\underline{D}} + \underline{\underline{K}})\underline{d}^{i+1} = \underline{F} - \underline{\underline{U}}\,\underline{d}^{i}. \tag{20}$$

The main benefit is that $\underline{d}^{i+1}$ can be computed from $\underline{d}^{i}$ directly, without having to store a full previous solution, following

$$d_j^{i+1} = \frac{1}{K_{jj}}\left(F_j - \sum_{l<j} K_{jl} d_l^{i+1} - \sum_{l>j} K_{jl} d_l^{i}\right). \tag{21}$$

Note that this operation can be done "in place", and is implemented in `gauss_seidel.m`. The GS method will converge (somewhat faster than the Jacobi method) for diagonally dominant, or positive definite and symmetric $\underline{\underline{K}}$.

2.2.3 Successive Over Relaxation (SOR)

Successive Over Relaxation is a more general variant of the Gauss-Seidel method that can lead to faster convergence. This is obtained by adding a parameter w which determines the weight of the current solution in the weighted average used to compute the next solution.

$$d_j^{i+1} = (1-w)\,d_j^{i} + \frac{w}{K_{jj}}\left(F_j - \sum_{l<j} K_{jl} d_l^{i+1} - \sum_{l>j} K_{jl} d_l^{i}\right) \tag{22}$$

Setting $w = 1$ will reduce SOR to the GS method. The optimal value of w is dependent upon the matrix $\underline{\underline{K}}$, but setting $w = 0.5$ is a good starting point. The method has been rigorously shown to converge for symmetric, positive definite matrices $\underline{\underline{K}}$ for $0 < w < 2$.

Exercise 1

a) Plot the Jacobi, GS, and SOR solutions for 32 elements and a tolerance of 10^{-4}, 10^{-5}, and 10^{-6} on one plot each; comment on the accuracy and number of iterations required. Can you improve the definition of tolerance for the Jacobi method?

b) Choose a tolerance of 10^{-6}, and record the number of iterations required to solve the 1-D FE example problem using the Jacobi and GS methods for increasing number of elements, *e.g.* for 8, 16, 32, 64, and 128 elements. (You might want to automate these computations and not wait until convergence and record the results by hand.)

c) Plot the number of iterations against number of elements for both methods.

d) Comment on the "scaling" of iteration numbers with size.

2.2.4 Conjugate gradient

You have now seen that while the Gauss-Seidel (GS) method is an improvement on the Jacobi approach, it still requires a large number of iterations to converge. This makes both methods impractical in real applications and other approaches are commonly used. One of those is the conjugate gradient (CG) method which works for positive-definite, symmetric, square ($n \times n$) matrices. The CG method is explained in a nice, geometric fashion by *Shewchuk* (1994). We cannot explore the motivation behind CGs in detail, but `conjgrad.m` provides a pretty straightforward MATLAB implementation which you should check out.

The CG method provides an exact solution after n iterations, which is often a prohibitively large number for real systems, and approximate solutions may sometimes be reached for a significantly smaller number of iterations. There are numerous tweaks involving modifications to the conjugate gradient method that pertain to "pre-conditioners" where we solve

$$\underline{\underline{M}}^{-1}\underline{\underline{K}}\underline{d} = \underline{\underline{M}}^{-1}\underline{F}, \tag{23}$$

for some $\underline{\underline{M}}$ which approximates $\underline{\underline{K}}$ but is simpler to handle than $\underline{\underline{K}}$. The best choice of these is, for some applications, an active area of research (*e.g.* *May and Moresi*, 2008).

For sparse least-squares problems, such as for the typically-mixed determined seismic tomography problem, the LSQR approach of *Paige and Saunders* (1982) is a popular choice that is used by many researchers for linear inversions. The robustness of the iterative solution compared to direct solvers was explored by *Boschi and Dziewoński* (1999), for example (it works!).

Exercise 2

a) Switch the solver from the GS method to conjugate gradient and increase the maximum iteration number stepwise from a fraction of n to the full n (as determined by the number of elements which you should choose large, *e.g.* 200, for this exercise). Test different initial guesses for $\underline{d}^i$ (*e.g.* all zero, random numbers), record the convergence and comment on the solution.

2.2.5 Multigrid method

An interesting philosophy to solving PDEs of the type we are considering for the 1-D finite element example is by using several layers of variable resolution grids (*e.g.* *Press et al.*, 1993, sec. 19.6). The insight is based on the observation that the Gauss-Seidel method is very good at reducing short-wavelength residuals in the iterative solution for $\underline{d}$ ("smoothing"), but it takes a long time to reduce the largest wavelength components of the residual. (You should try to plot successive solutions of the GS method compared to the analytical solution for different starting $\underline{d}_0$ to visualize this behavior.)

For the multigrid (MG) method, the idea is to solve the equations to within the desired tolerance only at a very coarse spatial discretization, where only a few iterations are required. Then, the solution is interpolated up to finer and finer levels where only a few GS iterations are performed at each level to smooth the solution. One then cycles back and forth until convergence is achieved at the finest, true solution level. There are several different approaches that are all called "multigrid" methods and basically only share the same philosophy. Differences are, for example found in terms of the way the cycling between fine and coarse resolutions are conducted (*e.g.* *Briggs et al.*, 2000), and we will only discuss the "V cycle" method. Multigrid methods are now implemented in most 3-D finite element methods (*Zhong et al.*, 2007) because MG has, ideally, the perfect scaling of $\mathcal{O}(N)$ where N is the size of the problem. MG methods areas of active research (*e.g.* algebraic multigrid, which is related to adaptive mesh refinement).

The multigrid method is based on expressing the PDE on L MG levels of resolution where the number of nodes in each level, n_i, is given by

$$n_i = b \times 2^{i-1} + 1 \quad \text{for} \quad i = 1, 2, \ldots, L, \tag{24}$$

where b is the base, typically a small number such as 2 or 4. At each i^{th} level, we need to construct separate stiffness matrices, $\underline{\underline{K}}_i$, and the corresponding force vector where the resolution for the $i = L$ solution is the best approximation to $\underline{\underline{K}}\,\underline{d} = \underline{F}$, and the forcing is only needed to be specified at $\underline{F}_L$ (see below).

An example implementation may proceed like so (see, *e.g.* *Press et al.*, 1993, sec. 19.6 for some alternatives): We start at the highest level, L, and perform only a few, fixed number of GS iterations for an rough approximate $\underline{d}_L$ from

$$\underline{\underline{K}}_L \underline{d}_L = \underline{F}_L \tag{25}$$

to remove the short wavelength misfit starting from an initial guess $\underline{d}_L = \underline{0}$. The residual is then given by

$$\underline{R}_L = \underline{F}_L - \underline{\underline{K}}\,\underline{d}_L. \tag{26}$$

We then *project*, or restrict, the residual to a coarser grid at $L-1$ by a projection operator P

$$\underline{R}_{L-1} = P_{L \to L-1} \underline{R}_L. \tag{27}$$

P will be some stencil giving more weight to the fine resolution nodes that are closer to the coarse resolution node to which we project. We next GS iterate

$$\underline{\underline{K}}_i \delta \underline{d}_i = \underline{R}_i \tag{28}$$

for $i = L-1$ for another small number of iterations (initializing $\underline{d}_i$ again with $\underline{0}$), performing another "smoothing" step, reducing short wavelength fluctuations. Note that eq. (28) now operates on the residual and not the load vector $\underline{F}$ such that we are computing corrections of $\underline{d}$, $\delta \underline{d}$. We then repeat the smoothing and projection steps down to $i = 1$ where eq. (28) can be solved quickly and exactly. This completes the downward leg of the V cycle where the longest wavelength residual has been addressed.

Next, we have to propagate the correction $\delta\underline{d}_1$ from $i = 1$ to $i = 2$ and higher resolutions by means of a "prolongation", *i.e.* an interpolation to higher resolution by an interpolation operator I

$$\delta\underline{d}_{i+1} = I_{i\to i+1}\delta\underline{d}_i. \tag{29}$$

I may be a linear interpolation, for example, which is easy to compute for the mesh structure eq. (24). This upward interpolated $\delta\underline{d}_{i+1}$ can then be smoothed by using it as a starting guess for a fixed number of GS iterations for

$$\underline{\underline{K}}_{i+1}\delta\underline{d}_{i+1} = \underline{R}_{i+1} \tag{30}$$

with $\delta\underline{d}_{i+1}$. We can now correct

$$\delta\underline{d}_{i+1} = \delta\underline{d}_{i+1} - \alpha\delta\underline{d}_{i+1} \tag{31}$$

$$\underline{R}_{i+1} = \underline{R}_{i+1} - \alpha\delta\underline{R}_{i+}1, \tag{32}$$

$$\tag{33}$$

with $\delta\underline{R}_{i+1} = -\underline{\underline{K}}_{i+1}\delta\underline{d}_{i+1}$ and weighting $\alpha = (\delta\underline{R}_{i+1} \cdot \underline{R}_{i+1})/|\delta\underline{R}_{i+1}|^2$. We continue by projecting $\delta\underline{d}_i$ in this fashion up to $i = L$, where we update $\underline{d}_L = \underline{d}_L + \delta\underline{d}_L$, which completes the upward leg of the V. The whole V cycle is then repeated until the desired tolerance for $\underline{d}_L$ is reached at which point $\underline{d}_L = \underline{d}$. Details of the implementations of the MG method, such as the smoothing, restriction, and prolongation operations, depend on the problem and the boundary conditions (*e.g. Press et al.*, 1993; *Briggs et al.*, 2000).

Exercise 3

a) Download the MG implementation of the 1-D FE example (based on `C` code by *Zhong*, 2008), `multigrid.m`. Read through the implementation, compare with the above recipe, and understand the approach.

b) Compare the number of iterations needed for the MG solver with that of the GS method for 32, 64, 128, 256 numbers of elements.

c) Plot the scaling of the number of iterations, or time spent in the multigrid subroutine, with the number of elements.

Bibliography

Boschi, L., and A. M. Dziewoński (1999), 'High' and 'low' resolution images of the Earth's mantle – Implications of different approaches to tomographic modeling, *J. Geophys. Res.*, *104*, 25,567–25,594.

Briggs, W. L., V. E. Henson, and S. F. McCormick (2000), *A multigrid tutorial*, 2 ed., The Society for Industrial and Applied Mathematics.

Dabrowski, M., M. Krotkiewski, and D. W. Schmid (2008), MILAMIN: MATLAB-based finite element method solver for large problems, *Geochem., Geophys., Geosys.*, *9*(Q04030), doi:10.1029/2007GC001719.

Golub, G. H., and C. F. Van Loan (1996), *Matrix computations*, 3 ed., Johns Hopkins University Press.

Hughes, T. J. R. (2000), *The finite element method*, Dover Publications.

May, D. A., and L. Moresi (2008), Preconditioned iterative methods for Stokes flow problems arising in computational geodynamics, *Phys. Earth Planet. Inter.*, *171*, 33–47.

Paige, C. C., and M. A. Saunders (1982), LSQR: an algorithm for sparse linear equations and sparse least-squares, *Trans. Math. Software*, *8*, 43–71.

Press, W. H., S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery (1993), *Numerical Recipes in C: The Art of Scientific Computing*, 2 ed., Cambridge University Press, Cambridge.

Shewchuk, J. R. (1994), An introduction to the conjugate gradient method without the agonizing pain, available online at `http://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.pdf`, accessed 10/2008.

Zhong, S. (2008), Iterative solutions of PDE, available online at `http://anquetil.colorado.edu/szhong/TEMP/tutorial_mg.tar.gz`, accessed 10/2008.

Zhong, S. J., D. A. Yuen, and L. N. Moresi (2007), Numerical methods in mantle convection, in *Treatise in Geophysics*, vol. 7, edited by G. Schubert and D. Bercovici, pp. 227–252, Elsevier.