

The point of the programming project is to give you the opportunity to appreciate the practical aspects of approximating PDEs. The point is *not* to make production-level code. I encourage you to write your programs in Matlab/octave, as it provides support for many vector & matrix functions. You can use some of the higher-level functionality of Matlab but do not, for example, use the PDE solver toolbox. As part of your project, you will submit a printout of your sourcecode. Please put human-readable comments in your code.

Select one of the following projects:

1. (**Elliptic PDE**) Write code to solve the general elliptic PDE with Dirichlet boundaries:

$$\begin{cases} -\nabla \cdot (\phi \nabla u) = f & (x, y) \in [0, 1] \times [0, 1] \\ u(0, y) = a(y) & 0 < y \leq 1, \\ u(x, 0) = b(x) & 0 \leq x \leq 1, \\ u(1, y) = c(y) & 0 < y \leq 1, \\ u(x, 1) = d(x) & 0 < x < 1 \end{cases}$$

where f, ϕ, a, b, c, d are given functions of x, y , and u is to be found. Discretize this system with the usual $\mathcal{O}(h^2)$ centered difference formul. Implement the Jacobi and Symmetric Gauss-Seidel iterative system solvers.

Your code should keep track of the (approximate) number of floating point operations, for purposes of comparison.

Test your code on problems for which you know the solution u , as well as those for which you do not. For each problem, provide a graph of residual versus iteration. For those problems for which you know the solution u you should present a graph of the ℓ_2 relative error of your approximate solution versus iteration; you should also plot contours of your approximate solution and the actual solution. You should present the data for your experiments in a way that conveys the essentials of how your algorithm performs on each problem.

Test your code on the following problems:

- (a) Let $u(x, y) = (x - x^2)(y^3 - y^2)$, and let $\phi(x, y) = e^x$. (This is the exact solution, you should find f, a, b, c, d yourself).
- (b) Let $u(x, y) = (x - x^2)(y - y^2) \exp(x^{4.5})$, and let $\phi(x, y) = \cos x$. (This is the exact solution.)
- (c) Let $\phi(x, y) = f(x, y) = 0$, and let $a(y) = c(y) = \sin(y), b(x) = d(x) = \sin(x)$.

2. **(Conjugate Gradient for poorly conditioned Elliptic PDE)** Write code to solve the following elliptic PDE with (zero) Dirichlet boundaries:

$$\begin{cases} u_{xx} + u_{yy} + 4(1 + \epsilon) \left(1 - \cos \frac{\pi}{n}\right) u = f & (x, y) \in [0, 1] \times [0, 1] \\ u(0, y) = u(1, y) = 0 & 0 < y \leq 1, \\ u(x, 0) = u(x, 1) = 0 & 0 \leq x \leq 1 \end{cases}$$

where f is a given function of x, y , ϵ and n are given constants, and u is to be found. Discretize this system with the usual $\mathcal{O}(h^2)$ centered difference scheme, using n cells in each direction, *i.e.*, let $\Delta x = \Delta y = 1/n$. You should find that for ϵ small, the discretized system you wish to solve, *i.e.*, $\mathbf{L}_h \mathbf{u}_h = \mathbf{f}_h$, is poorly conditioned.

Note that it should seem odd to you that the n from the PDE is used in the numerical solver. This example is artificial, but allows easy control over the conditioning of the discretized matrix via parameter ϵ . This example illustrates what might occur, seemingly at random, in solving the *Helmholtz Equation*:

$$\nabla^2 u + \lambda u = f,$$

for some scalar λ .

Implement the Conjugate Gradient method with diagonal (a.k.a. “Jacobi”) preconditioner to solve this problem [3, 6, 4].

Your code should keep track of the (approximate) number of floating point operations, for purposes of comparison.

Test your code on problems for which you know the solution u , as well as those for which you do not. For each problem, provide a graph of residual versus iteration. For those problems for which you know the solution u you should present a graph of the ℓ_2 relative error of your approximate solution versus iteration; you should also plot contours of your approximate solution and the actual solution. You should present the data for your experiments in a way that conveys the essentials of how your algorithm performs on each problem.

Test your code on the following problems:

- First set $\epsilon = -1$, and test the code with $f(x, y) = -2[x - x^2 + y - y^2]$. This problem has exact solution $u(x, y) = (x - x^2)(y - y^2)$.
- Now test the code with $f(x, y) = 1$. For various values of n , test your code with various values of ϵ going to zero; test your code with and without the preconditioner. Test the effects of small ϵ on your code, and how well the preconditioner deals with the ill-conditioning. For these tests, you want to look at how the residual is reduced with increasing iterations.
- Try to come up with a problem (*i.e.*, data f, ϵ) for which preconditioning makes a real difference in convergence of the method.

3. (Hyperbolic PDE) Write code to solve the initial value *transport equation*:

$$\begin{cases} -u_t = au_x & x \in \mathbb{R}, t > 0 \\ u(x, 0) = u_0(x) & x \in \mathbb{R} \end{cases}$$

Implement the following schemes to solve this problem: (a) The Upwind scheme. (b) The Lax Wendroff scheme. (c) The (unstable) “average scheme”:

$$U_i^{j+1} = U_i^j - \frac{a_i^j \Delta t}{2\Delta x} (U_{i+1}^j - U_{i-1}^j).$$

Apply your code to the following problems:

(a) Let $x_0 = -1, x_f = 1, t_f = 1$, and

$$\begin{aligned} a(x, t) &= |x| \\ u_0(x) &= \begin{cases} 1 & -0.25 \leq x \leq 0.25, \\ 0 & \text{otherwise.} \end{cases} \end{aligned}$$

(b) Let

$$\begin{aligned} a(x, t) &= \frac{1+x}{t+(1+x)^2} \\ u_0(x) &= \begin{cases} 1 & 0.25 \leq x \leq 0.5, \\ 0 & \text{otherwise.} \end{cases} \end{aligned}$$

Use $x_0 = 0, x_f = 1, t_f = 1$. Note the exact solution appears to be

$$u(x, t) = u_0\left(x - \frac{t}{1+x}\right).$$

(c) Let

$$\begin{aligned} a(x, t) &= \frac{1+x}{t+(1+x)^2} \\ u_0(x) &= e^{-10x^2} \end{aligned}$$

Use $x_0 = -1, x_f = 1, t_f = 1$.

Experiment to find satisfactory mesh spacings $\Delta t, \Delta x$, then produce graphs of the approximate and actual solutions at different times. (*cf.* Fig. 4.6–4.8 of [5]) If the unstable method performs acceptably for these problems, try to find a problem for which it is unsatisfactory.

Test the effects of mesh spacing on the ℓ_2 norm of the error (approximate solution minus actual solution), as well as the effects of mesh spacing on flops. Write a report of your findings.

4. (**Parabolic PDE**) Write code to solve the *heat equation*:

$$\begin{cases} u_{xx} = u_t & x \in [0, 1], t \in [0, T], \\ u(x, 0) = u_0(x) & x \in [0, 1], \\ u(0, t) = a(t) & t \in (0, T], \\ u(1, t) = b(t) & t \in (0, T] \end{cases}$$

Implement the following schemes to solve this problem: (a) Explicit scheme. (b) Implicit scheme. (c) Crank Nicolson scheme. Note that you should have already implemented the explicit scheme for homework problem 3.2. You may reuse that code.

Apply your code to the following problems:

(a) Let $T = 1$, and

$$u_0(x) = x^2, \quad a(t) = 0, \quad b(t) = 1.$$

(b) Let $T = 1$, and

$$\begin{aligned} u_0(x) &= \sin(\pi x), \\ a(t) = b(t) &= 0. \end{aligned}$$

Note this has exact solution

$$u(x, t) = \exp(-\pi^2 t) \sin(\pi x)$$

(c) Let $T = 1$, and

$$\begin{aligned} u_0(x) &= (x - x^2) e^x, \\ a(t) = b(t) &= 0. \end{aligned}$$

Investigate the behaviour of these methods for different mesh sizes, and different values of

$$r = \frac{\Delta t}{(\Delta x)^2}.$$

In particular, try the values

$$r = 0.25, 0.49, 0.51, 0.75, 1.0, \quad \text{and} \quad \Delta x = (1/10), (1/20), (1/40), (1/80).$$

Present data and make some conjectures about the convergence of the ℓ_2 norm of the error for each of these methods with respect to Δx , for the problem(s) for which you know the exact solution u . How do these methods respond to different values of r ? Present some graphs of your computed (and when known, superimpose the actual) solutions at time T . Write a report on your findings.

5. **(Method of Lines for mixed PDE)** Write code to solve *Burgers' Equation*:

$$\begin{cases} -uu_x + \nu u_{xx} = u_t & x \in [0, 1], t \in [0, T], \\ u(x, 0) = u_0(x) & x \in [0, 1], \\ u(0, t) = a(t) & t \in (0, T], \\ u(1, t) = b(t) & t \in (0, T] \end{cases}$$

Implement the method of lines [2, 3] to solve this equation. Your code should include the following schemes for the spatial discretization: (a) one sided upwind scheme, (b) centered difference. Implement a 4th order Runge Kutta scheme [4, pp.613] to solve the resultant system of ODEs.

Apply your code to the following problems:

(a) Let $T = 2$, and let the exact solution be

$$u(t, x) = \frac{1}{1 + \exp((x/2\nu) - (t/4\nu))}.$$

(b) Let $T = 2$, and

$$\begin{aligned} u_0(x) &= 0.5 \sin(\pi x) + \sin(2\pi x), \\ a(t) = b(t) &= 0. \end{aligned}$$

Investigate the affect on your code of varying the viscosity, ν . In particular, try $\nu = 1, 0.5, 0.1, 0.01, 0.001$. Make three dimensional plots of your approximate solutions or overlay plots of $u(x, t)$ for $t = 0, 0.25, 0.5, \dots$. For the first problem, compare your computed solution to the actual solution. How does viscosity affect accuracy and smoothness of the computed solution? What is the effect of mesh size? Write a report on your findings.

6. (**Your Suggestion**) The abovementioned projects are by no means exhaustive of techniques for solving PDEs. I am amenable to allowing you to suggest your own project, however, it should be comparable in difficulty and work to those listed above. Some possible projects:
- (a) Finite Element method for Elliptic PDE. (Note you should *not* reuse code from a previous course for this project.)
 - (b) Spectral method [7, 8].
 - (c) Finite Volume Scheme.
 - (d) Simulation of a specific physical system, *e.g.*, fluid flow, planetary motion, fabric in the wind, combustion, etc.

References

- [1] Guillaume Bal. Lecture notes, course on numerical analysis, 2002. URL <http://www.columbia.edu/~gb2030/COURSES/E6302/NumAnal.ps>.
- [2] Randolph E. Bank, Peter Rentrop and Donald R. Smith. Numerical treatment of partial differential equations, 1995. Course Notes for UCSD Math M172.
- [3] Michael T. Heath. *Scientific Computing: An Introductory Survey*. McGraw-Hill, Boston, 1997. ISBN 0-07-027684-6.
- [4] David Kincaid and Ward Cheney. *Numerical analysis*. Brooks/Cole Publishing Co., Pacific Grove, CA, second edition, 1996. ISBN 0-534-33892-5. Mathematics of scientific computing.
- [5] K. W. Morton and D. F. Mayers. *Numerical solution of partial differential equations*. Cambridge University Press, Cambridge, second edition, 2005. ISBN 978-0-521-60793-0; 0-521-60793-0. An introduction.
- [6] Jonathan Richard Shewchuk. An introduction to the conjugate gradient method without the agonizing pain, 1994. URL <http://www-2.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.ps>.
- [7] Lloyd N. Trefethen. Finite difference and spectral methods for ordinary and partial differential equations, 1996. URL <http://web.comlab.ox.ac.uk/oucl/work/nick.trefethen/pdetext.html>.
- [8] Lloyd N. Trefethen. *Spectral methods in MATLAB*. Software, Environments, and Tools. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2000. ISBN 0-89871-465-6.