# Computer lab exercises and homework for Sept. 23

Due date: Friday, Oct. 7 at midnight.

## Background and reading assignment

Iterative methods are a common way to solve $n \times n$ linear systems of the form

$$\mathbf{Au} = \mathbf{f}$$

when $n$ is large and $\mathbf{A}$ is sparse (i.e., most of the entries of $\mathbf{A}$ are zero). Such systems arise, for example, from finite-difference schemes to solve the Poisson and Helmholtz equations on rectangular domains.

The reading assignment for this exercise is Chapters 1 and 2 of William Brigg's *A Multigrid Tutorial* (SIAM, 2000). The text is available online through the ASU library, and you may download PDFs of the first two chapters. These chapters describe a couple of model problems derived from Poisson's equation with Dirichlet boundary conditions, given by

$$\begin{aligned} -\nabla^2 u &= f \quad \text{on } \Omega \\ u &= 0 \quad \text{on } \partial\Omega \end{aligned} \tag{1}$$

on the domain $\Omega$.

Finite difference schemes applied to this problem lead to sparse systems of linear equations. In one dimension, $\Omega$ is the interval $[a, b]$, which we divide into $n$ subintervals (with $n + 1$ grid points). The finite-difference scheme applied to Eq. (1) leads to a system of $n - 1$ linear equations of the form

$$\frac{-v_{j-1} + 2v_j - v_{j+1}}{h^2} = f_j, \quad 1 \le j \le n - 1 \tag{2}$$

where $f_j = f(x_j)$ and $v_j$ is the approximation to the solution $u(x_j)$. The grid points are subscripted from $x_0$ to $x_n$; the points $x_0$ and $x_n$ are the boundary points. For the Dirichlet problem, $x_0 = x_n = 0$, and we must solve for the solution on the interior points $x_1, x_2, \ldots, x_{n-1}$.

Equation (2) corresponds to the matrix-vector equation

$$\mathbf{Av} = \mathbf{f},$$

where $\mathbf{A}$ is a tridiagonal $(n - 1) \times (n - 1)$ matrix with 2's on the diagonal and $-1$'s on the adjoining sub- and superdiagonals. While such a matrix may be factored efficiently with LAPACK, higher-dimensional problems—particularly in 3 dimensions—yield block-diagonal matrices that are very large and sparse, even for grids of relatively modest size.

Iterative methods, as the name implies, try to converge upon a solution in steps, rather than performing a full *LU* decomposition. One difficulty with Gaussian elimination on a sparse matrix is "fill-in"—the factored matrices $\mathbf{L}$ and $\mathbf{U}$ are not necessarily sparse. One classical scheme for solving sparse systems is Jacobi iteration. It is well suited to vector architectures. One difficulty, however, is that Jacobi iteration can be slow. An alternative scheme, Gauss-Seidel iteration, can be faster, but in its classical form involves a recurrence relation that is not vectorizable. A slight modification, red-black Gauss-Seidel iteration, can be parallelized. Multigrid refers to a family of iterative methods that can be rapidly convergent for certain types of discretization problems.

Chapter 1 of Brigg's monograph outlines the basics of finite-difference methods and the associated linear equations. Chapter 2 describes the Jacobi and Gauss-Seidel methods, along with an analysis of the methods' ability to damp various kinds of solution error.

## Homework problems

1. Complete Exercises 2 and 5 in Chapter 1 of Briggs' monograph.

2. Write a vectorized Fortran subroutine to implement weighted Jacobi iteration for the linear system $\mathbf{Au} = \mathbf{f}$, where $\mathbf{A}$ is the matrix arising from the 1-d model problem, Eq. (1.3) in the text with $\sigma = 0$. Let your subroutine be of the form

   ```
   subroutine jacobi1d(n,niter,omega,f,v,resid,error,info)
   ```

   The arguments are as follows:

   **`n`** is an input integer giving the number of subintervals in the domain. It must be at least 2.

   **`niter`** is an input integer specifying the number of times to iterate the method. It must be at least 1.

   **`omega`** is the input weighting (damping) factor. It must satisfy $0 < \omega \leq 1$.

   **`f`** is the input $(n-1)$-vector giving the discretized right-hand side.

   **`v`** on input is a $(n-1)$-vector giving an initial guess of the solution. On return, `v` contains the final iterated solution vector on return.

   **`resid`** is the output $(n-1)$-vector giving the residual from the last iteration.

   **`error`** is an output vector of length `niter` whose its $j$th entry is the norm of the error vector on the $j$th iteration.

   **`info`** is an output error indicator. Set it to a negative value (and document your choice) if `n`, `niter`, or `omega` has an illegal value and to zero otherwise.

   Supply each dummy argument with an appropriate `intent` modifier. Document your subroutine in MATLAB style. If you need other inputs or outputs to your routine, then include them with appropriate documentation.

3. Write a vectorized Fortran subroutine to implement red-black Gauss-Seidel iteration for the model problem. Let your subroutine have the same form as above:

   ```
   subroutine gs1d(n,niter,f,resid,error,info)
   ```

   (`omega` is not needed) with otherwise similar arguments. Wrap both subroutines in a module called `relaxation`. Use double-precision arithmetic.

4. Write a main program as follows:

   (a) Call your Jacobi iterator with $\omega = 2/3$ for a grid of size $n = 64$ and initial guess vectors $\mathbf{v}_1$, $\mathbf{v}_3$, and $\mathbf{v}_6$. (One call for each $\mathbf{v}$.) Use 100 iterations and $\mathbf{f} = \mathbf{0}$ as explained in the text (see p. 12). Output a 4-column table listing the iteration number and the norm of the error at each iteration for each of the $\mathbf{v}$'s. More specifically,

   - The first column is the iteration number $j$. Your table will have 100 rows, corresponding to $j = 1, 2, \ldots, 100$.
   - The second column is the error at iteration $j$ in the solution starting with $\mathbf{v}_1$ as the initial guess.
   - The third column is the error at iteration $j$ in the solution starting with $\mathbf{v}_3$ as the initial guess.
   - Likewise for $\mathbf{v}_6$ in the fourth column.

   You can get nicely formatted output as follows. Suppose that `error1`, `error3`, and `error6` are the iteration errors from the respective initial guesses. Then you may print each row of the table with a loop like this:

   ```
   do j=1, 100
     write(6, '(i5, 3es12.3)') j, error1(j), error3(j), error6(j)
   enddo
   ```

   Don't forget to enclose the format strings in parentheses as indicated. The format item `i5` prints `j` as a decimal integer, right-justified in a field that is 5 characters wide. `es12.3` formats a floating-point number as $\pm 1.234\mathrm{e} \pm 05$, right-justified in a field that is 12 characters wide. The leading 3 is a repeat count. Adjust the field widths to your liking. Your results, when plotted, should reproduce Fig. 2.3(a) in Briggs.

   (b) Proceed as in (a) with your red-black Gauss-Seidel iterator. Your results, when plotted, should reproduce Fig. 2.3(c) in Briggs.

5. Include a makefile for your program that compiles and links on the lab computers in ECA. For debugging, you may wish to include the flags `-g -fcheck=bounds` in your Gfortran compilation. (See last week's makefile for an example `DEBUG` make variable.) For optimization, you can try the flags `-O2 -ftree-vectorize` or the flag `-O3`. Optimizers can be buggy, so you might try `-O2` first and verify that your program's results don't change. Package your code with a command like

   ```
   tar -c -f hw1007.tar relaxation.f90 main.f90 Makefile
   ```

   and any other files that you need to complete the assignment.