AM225: Assignment 3 (due 5 PM, March 25)

*Complete **at least two** questions. If you submit answers for more, your grade will be calculated using the best scores. The questions are roughly ordered by difficulty, from easiest to hardest.*

1. **Strassen's algorithm.** The standard approach for multiplying two $n \times n$ matrices together, $AB = C$, requires $O(n^3)$ operations. In 1969, Strassen discovered an algorithm that requires asymptotically fewer operations. Suppose n is even, and write the matrix product as

$$\begin{pmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{pmatrix} \begin{pmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{pmatrix} = \begin{pmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{pmatrix} \tag{1}$$

   where each term is a submatrix of size $\frac{n}{2} \times \frac{n}{2}$. If the seven quantities

$$Q_0 = (A_{00} + A_{11})(B_{00} + B_{11}), \tag{2}$$
$$Q_1 = (A_{10} + A_{11})B_{00}, \tag{3}$$
$$Q_2 = A_{00}(B_{01} - B_{11}), \tag{4}$$
$$Q_3 = A_{11}(B_{10} - B_{00}), \tag{5}$$
$$Q_4 = (A_{00} + A_{01})B_{11}, \tag{6}$$
$$Q_5 = (A_{10} - A_{00})(B_{00} + B_{01}), \tag{7}$$
$$Q_6 = (A_{01} - A_{11})(B_{10} + B_{11}) \tag{8}$$

   are computed, then the entries in C are given by

$$C_{00} = Q_0 + Q_3 - Q_4 + Q_6, \tag{9}$$
$$C_{10} = Q_1 + Q_3, \tag{10}$$
$$C_{01} = Q_2 + Q_4, \tag{11}$$
$$C_{11} = Q_0 + Q_2 - Q_1 + Q_5. \tag{12}$$

   This procedure can be applied recursively. Since the original problem of dimension n is replaced by seven problems of dimension $\frac{n}{2}$, the total number of operations required is $O(n^{\log_2 7}) = O(n^{2.807})$. While Strassen's method requires more operations than the standard implementation, the improvement in exponent will eventually make it worthwhile for large matrices.

   (a) Write a program to implement the standard matrix multiplication algorithm. Time your code on random matrices of dimensions that are powers of two from $n = 16$ upward.[1]

   (b) Implement Strassen's algorithm, and time your code on random matrices of dimensions that are powers of two from $n = 16$ upward. Compare your results to part (a). Fit your timing results to a power law αn^β and report the values of α and β.[2]

   (c) Time the BLAS matrix–matrix multiply routine, `dgemm`, and compare your results to the previous sections. You should find that the BLAS routine is faster, but has $O(n^3)$ scaling.

[1] Try to make n as large as your system will allow. For example, for $n = 8192$, you will need at least $3 \times 8192^2 \times (8 \text{ bytes}) = 403$ MB of memory to represent the three matrices.

[2] You should find $\beta < 3$ as expected.

(d) **Optional.** Extend your codes from parts (a) and (b) to use multithreading and repeat the timing comparison.

(e) **Optional.** Extend your code from part (b) to work on matrices that are not powers of two.

(f) **Optional.** It is difficult to beat BLAS for matrices of practical size, due to its fine tuning of caching performance. Could a combination of Strassen and BLAS be faster than BLAS itself?

2. **Fractal-based preconditioning** In lecture 9, the preconditioned conjugate gradient (PCG) algorithm was introduced, and tested on a linear system to find radial basis function weights. The program files contain a program called `rbf_time.cc`, which compares the speed of solving the system with LAPACK and PCG for k points varying from 10 to 10000. This code uses the compact Wendland radial function with cutoff radius set to $r(k) = 5/\sqrt{k}$, which results in approximately 15 entries in each line of the linear system. The PCG uses a block Jacobi preconditioner with blocks of size $b(k) = \sqrt{k}$.[3] Each block is inverted using dense linear algebra via LAPACK.

   Ideally, we want the blocks to be as dense as possible, and thus it is beneficial to order the points in the linear system so that they are physically nearby. The current code orders the points by the y coordinate (see `rbf::init_random`), which helps, but points can still be delocalized in the x direction, resulting in sparse Jacobi blocks.

   (a) Write a program to calculate the total non-zero matrix entries $T(k)$ and the non-zero matrix entries in the Jacobi blocks $P(k)$.[4] Make a plot of the proportion of non-zero matrix entries in the Jacobi blocks $P(k)/T(k)$ as a function of k, for a variety of values from $k = 10$ to $k = 10000$.

   (b) The Hilbert curve is a space-filling fractal that bijectively maps a one-dimensional coordinate q continuously into a square with coordinates $\mathbf{x} = (x, y)$. Detailed information about the construction of Hilbert curves is available online (*e.g.* Wikipedia). For the given example, the sample points $\mathbf{x}_i$ lie in the square $[-1, 1]^2$. Write a program that computes the 1D coordinate q_i for each point, and then re-sorts the points based on these q values, thereby ensuring that points are physically close together in the indexing. Make a corresponding plot of $P(k)/T(k)$ like part (a) using this point initialization method.

   (c) Use the program `rbf_time.cc` to compare the time of PCG for the original initialization method and the fractal-based method, over the range from $k = 10$ to $k = 10000$.

   (d) **Optional.** Try varying the functions $r(k)$ and $b(k)$ to see how they affect your results in the previous parts.

   (e) **Optional.** Are there better ways to organize the points than the Hilbert curve?

3. **Alternative cubic elements.** In the lectures, we considered solving the elliptic problem

$$-\frac{d}{dx}\left(x\frac{du}{dx}\right) = f(x) \tag{13}$$

[3] In many cases $b(k)$ will not exactly divide k. In this case a final block is used with size less than $b(k)$.

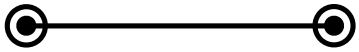
[4] You could do this by adding your own function to the `rbf` class.

for a function $u(x)$ on $\Omega = (1, 2)$ with the boundary conditions

$$u(1) = 0, \qquad \left.\frac{du}{dx}\right|_{x=2} = g \tag{14}$$

where g is a real constant. The interval was divided into N subintervals of equal length, and the finite elements were constructed from cubics with a nodal basis. Following the notation introduced the lectures, this corresponds to a standard element shown below.

Modify this example code to solve the problem using the alternative cubic basis whereby the function values and first derivatives are specified, which corresponds to the standard element shown below.



With this basis, your finite element functions will be in $C^1(\Omega)$.

(a) For $N = 3$, make a plot of the finite element basis functions that is similar to Fig. 1 in the example notes. In the notes, the finite element problem on N intervals had a basis with $3N$ functions.[5] For this alternative basis, what is the dimension of the finite element function space when N intervals are used?

(b) Write a program to solve the elliptic problem using the alternative cubic basis.[6] Using the manufactured solution considered in the notes, make a log–log plot of the L_2 error of the numerical solution as a function of N for a range of values from $N = 10$ to $N = 1000$. On your plot, include also the L_2 errors for the original code with the nodal basis.

(c) Make another log–log plot comparing the L_2 error for the two methods, but for this case, put the wall clock time for the computation on the horizontal axis. Which basis is more efficient for achieving a desired level of accuracy?

4. **Simulating an MRI.**[7] Magnetic resonance imaging (MRI) is a noninvasive medical imaging technique in which a body is subject to a magnetic field operating at a frequency f. A simple model of an MRI is given by the Helmholtz equation

$$\nabla^2 u(\mathbf{x}) + k^2 u(\mathbf{x}) = v(\mathbf{x}), \tag{15}$$

where $u(\mathbf{x})$ is the electric field strength, k is the wave number, and $v(\mathbf{x})$ is the electric excitation. In general, both $u(\mathbf{x})$ and k can be complex-valued. Note that $k^2 = \omega^2 \mu \epsilon$, where $\omega = 2\pi f$, μ is the magnetic permeability, and ϵ is the electric permittivity. In free space, $k^2 = k_0^2 = \omega^2 \mu_0 \epsilon_0$, where $\mu_0 = 4\pi \times 10^{-7}\,\mathrm{H\,m^{-1}}$ and $\epsilon_0 = 8.8542 \times 10^{-12}\,\mathrm{F\,m^{-1}}$.

(a) Implement a second-order accurate finite difference scheme for Eq. 15 in free space on the domain $[0, 1]^2$ meters with homogeneous Dirichlet boundary conditions. Set $v(\mathbf{x})$ to be the impulse $v(\mathbf{x}) = \delta(\mathbf{x} - (0.5, 0.5))$ and use a frequency of $f = 21.3\,\mathrm{MHz}$. Assume $u(\mathbf{x})$ is complex-valued. Solve the resulting system using (i) a direct banded solver

[5] In the notes, there was one additional function ψ_0 which was included for completeness, but was not considered the finite-element linear system since it was constrained to zero by the Dirichlet condition at $x = 1$.

[6] You can either write your own code, or modify the one from the example.

[7] This question was written by Dan Fortunato (TF, Spring 2018).

in LAPACK[8] and (ii) a fast solver based on the discrete sine transform[9] (DST). Make a single log–log plot of the timings of both methods using $N+1$ gridpoints in each dimension for $N = 16, 32, \ldots, 1024$.[10]

(b) Human tissue can be modeled as a material with $\mu = \mu_0$ and $\epsilon = \epsilon(\mathbf{x})$, causing $k = k(\mathbf{x})$ to vary in space:

$$\nabla^2 u(\mathbf{x}) + k(\mathbf{x})^2 u(\mathbf{x}) = v(\mathbf{x}). \tag{16}$$

Use method (i) to simulate an MRI of a human head with $N = 256$ and $v(\mathbf{x}) = \delta(\mathbf{x} - (0.6, 0.7))$, and show the resulting MRIs by plotting $|u|$ for $f = 21.3\,\text{MHz}$ and $f = 298.3\,\text{MHz}$. To model the head, use the provided binary data `mri_data_256.dat`, which gives values for $\epsilon(\mathbf{x})/\epsilon_0$ as an $(N+1) \times (N+1)$ matrix of complex numbers stored in row-major order.

(c) Does the DST-based method work for Eq. 16? Explain why or why not.

(d) As N gets large, method (i) becomes computationally infeasible, and iterative methods are used instead. A simple iterative method for solving Eq. 16 that can utilize the DST-based solver is

$$\nabla^2 u^{(i+1)} + k_0^2 u^{(i+1)} = v - \left(k(\mathbf{x})^2 - k_0^2\right) u^{(i)}, \tag{17}$$

where $u^{(i)}$ denotes the solution at iteration i and $u^{(0)} = 0$. Implement this method and produce a log–log plot of the L^∞-norm relative error at each iteration, taking the solutions from (b) as the true solutions for $f = 21.3\,\text{MHz}$ and $f = 298.3\,\text{MHz}$. Does convergence depend on f?

(e) **Optional.** Define the error at iteration i to be $\delta u^{(i)} = u^{(i)} - u$. Show that

$$\delta u^{(i+1)} = \left(\nabla^2 + k_0^2\right)^{-1} \left(k_0^2 - k(\mathbf{x})^2\right) \delta u^{(i)} \tag{18}$$

and derive a condition on $k(\mathbf{x})$ that guarantees convergence.

(f) **Optional.** Solve Eq. 16 using the iterative method with $N = 1024$ and $f = 21.3\,\text{MHz}$ and record the time to convergence. Estimate the time it would take for method (i) to solve this problem. Use the provided data `mri_data_1024.dat`.

5. **Testing the Schur complement for the Poisson equation.** Figure 1 shows an example of a *perfect squared square*, whereby a square of side length 112 is tessellated by twenty one smaller squares with unique integer side lengths. A text file `pss.txt` is provided that lists the square side lengths and (x, y) offsets from the bottom left corner.

   In the lectures, we considered the model problem of solving the Poisson equation $-\nabla^2 u = f$ on the domain $[0,1]^2$ with zero Dirichlet boundary conditions. A C++ class to solve this problem using the FFT is provided in the course examples (`poisson_fft.cc`).

[8] If you index the grid in the conventional way, *i.e.* (i, j) maps to $i + (N+1)j$, then the matrix you need to solve is banded with $N+1$ lower and $N+1$ upper diagonals. In this case, there is a LAPACK routine `zgbsv` that provides a huge saving in memory and execution time over a full dense solver. See `banded_test.cc` for an example.

[9] Use FFTW (`www.fftw.org`) or another library to perform the FFTs.

[10] You should have no problem in using $N = 1024$ with the DST method. However, the direct solver may become too expensive, so you may skip some of the larger grid sizes for this case.

Suppose now that the domain is discretized into 112×112 grid. Specifically, the domain is divided into squares with side length $h = 1/112$, with gridpoints located on the corners of the squares. Consider the following two equivalent methods to solve this problem:

- Applying the FFT solver to the entire grid. Following the the notation of lecture 9, this grid corresponds to using $N = 111$, since there is a matrix of 111×111 unknown entries in the grid interior.
- Using the Schur complement method to solve the twenty-one non-overlapping square domains numbered $k = 1, \ldots 21$ with grid sizes shown. In this case, the final domain $k = 22$ corresponds to all gridpoints that lie on the internal square edges in Fig. 1. In the AM225 code examples Git repository, the directory `2c_fft+ddecomp/pss` contains a code `pss_test.cc` that reads in the perfect squared square information, and prints ASCII art showing how the grid is subdivided. It also prints out a table of adjacencies for all gridpoints in the final domain with $k = 22$.

Write codes to solve this problem using the two methods above, for the source term of $f(x, y) = e^{x-y}$. Show that your two codes give identical results up to numerical precision.

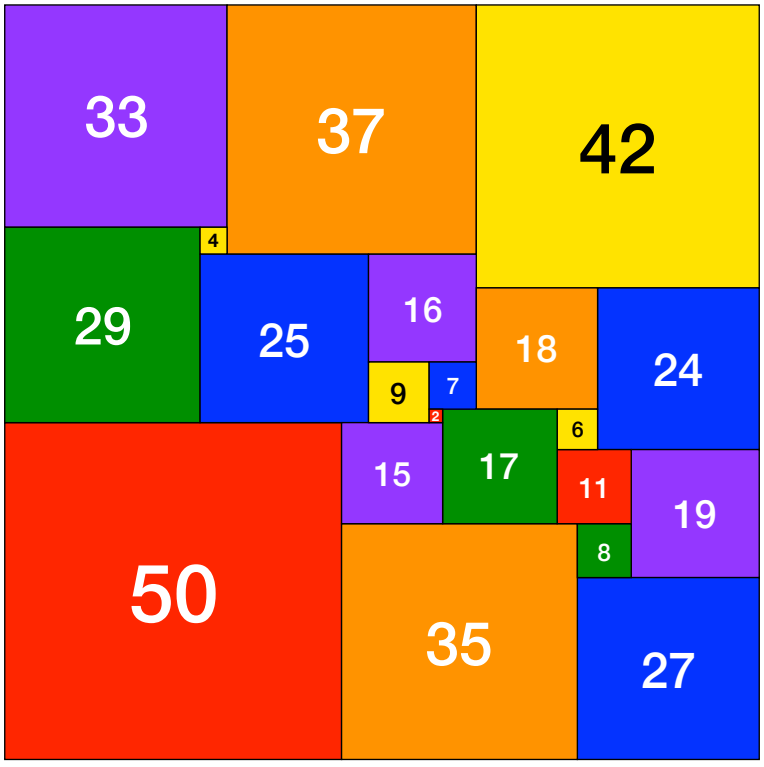


Figure 1: A perfect squared square.