Math 566 **Homework 5** **Due April 2, 2021**
Grady Wright

1. (**Gaussian elimination for a structured matrix**) Assuming no pivoting is needed (to avoid breakdown or to ensure numerical stability), devise an efficient way to arrange the computations for solving an n-by-n linear system with non-zero entries in the coefficient matrix only in the first and last rows and columns and also in the two main diagonals. In the case of a 9×9 matrix, the non-zero entries would appear as follows:

$$\begin{bmatrix} \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & & & & & & \bullet & \bullet \\ \bullet & & \bullet & & & & \bullet & & \bullet \\ \bullet & & & \bullet & & \bullet & & & \bullet \\ \bullet & & & & \bullet & & & & \bullet \\ \bullet & & & \bullet & & \bullet & & & \bullet \\ \bullet & & \bullet & & & & \bullet & & \bullet \\ \bullet & \bullet & & & & & & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \end{bmatrix} \begin{bmatrix} \\ \\ \\ \\ x \\ \\ \\ \\ \\ \end{bmatrix} = \begin{bmatrix} \\ \\ \\ \\ b \\ \\ \\ \\ \\ \end{bmatrix},$$

where $\bullet$'s represent a non-zero entry and blanks represent zero entries. It is sufficient to illustrate the efficient method for this 9×9 case. However, for the general n-by-n case, determine the number of operations (additions/subtractions and multiplications/divisions) your algorithm requires (these should both be $O(n)$). Your method should not introduce a non-zero value in any step where there was previously (or initially) a zero.

   Note: You are not computing an LU factorization of the matrix, but solving the system (i.e. doing Gaussian elimination-type steps on the augmented system).

2. Pentadiagonal linear systems have the general form

$$\begin{bmatrix} a_1 & c_1 & e_1 & & & & & \\ b_1 & a_2 & c_2 & e_2 & & & & \\ d_1 & b_2 & a_3 & c_3 & e_3 & & & \\ & d_2 & b_3 & a_4 & c_4 & e_4 & & \\ & & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & & d_{n-4} & b_{n-3} & a_{n-2} & c_{n-2} & e_{n-2} \\ & & & & d_{n-3} & b_{n-2} & a_{n-1} & c_{n-1} \\ & & & & & d_{n-2} & b_{n-1} & a_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_{n-2} \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ \vdots \\ f_{n-2} \\ f_{n-1} \\ f_n \end{bmatrix}$$

   and arise in enough applications (although not as much as tridiagonal systems) that it makes sense to design special algorithms for solving them.

   (a) Derive a fast algorithm for solving these systems using a similar approach to what was done in class and in the book for solving tridiagonal systems (you can assume no pivoting is necessary). Fast here means it should take $\mathcal{O}(n)$ operations.

   (b) Determine the exact number of operations your algorithm requires for solving a general n-by-n pentadiagonal system.

   (c) Implement your algorithm as a function in some programming language. The function should take as input the vectors $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$, $\mathbf{d}$, $\mathbf{e}$, and $\mathbf{f}$ containing the respective entries in the system above.

(d) Test your code from part (c) using the values $a_i = i$, for $i = 1, \ldots, n$, $b_i = c_i = -(i+1)/3$, for $i = 1, \ldots, n-1$, $d_i = e_i = -(i+2)/6$, for $i = 1, \ldots, n-2$, and $f_1 = 1/2$, $f_2 = 1/6$, $f_i = 0$, for $i = 3, \ldots, n-2$, $f_{n-1} = 1/6$, and $f_n = 1/2$. Use $n = 100$ and $n = 1000$ and verify that algorithm gives the correct answer.

3. (**Cholesky for sparse matrices**) As we discussed in class, if A is a symmetric positive definite matrix then A can be decomposed as $A = R^T R$, where R is an upper triangular matrix. This decomposition is called the Cholesky decomposition.

   For this problem you will use the Cholesky function for sparse matrices built into the programming language of your choosing. For example, the function `chol` in MATLAB is a very sophisticated implementation that efficiently handles the decomposition of a *sparse* symmetric positive definite matrix by exploiting any structure formed by the non-zero entries in the matrix.

   (a) Download the file bcsstk38.mat from the course web page. This file contains a sparse matrix that arises from a structural analysis of an engine compartment of one of Boeing's airplanes. The file is in MATLAB format, but Python and Julia have function for reading these file types. Load the matrix into the software environment you are using. In MATLAB, this would be done using the following series of commands:

```
>>load bcsstk38;
>>A = Problem.A;
```

   Make a spy plot of the matrix A showing its sparsity pattern. Compute the sparsity ratio of A using

$$\text{sparsity ratio} = 1 - \frac{\text{number non-zeros in } A}{\text{number of total entries in } A}.$$

   In MATLAB, the function `nnz` and `numel` will be helpful here. Report the sparsity ratio along with some descriptive text in the title of the spy plot of A.

   (b) Compute the Cholesky decomposition of the matrix from part (a). Plot the sparsity pattern of the upper triangular matrix from the decomposition. Compute the amount of "fill-in" from the Cholesky decomposition. The "fill-in" can be defined as the ratio of the number of non-zero entries in the Cholesky decomposition to the number of non-zero entries in the original matrix. Report this number along with some descriptive text in the title of the spy plot of the upper triangular matrix.

   (c) What you would like to happen is for the "fill-in" from the Cholesky decomposition to be as small as possible since this translate directly into a more computationally efficient method for solving the underlying linear system involving the matrix A. For any given sparse symmetric positive definite matrix, the "fill-in" that occurs from the Cholesky decomposition depends on how the rows and columns of the matrix are ordered. Several algorithms exist for permuting the rows and columns to try and minimize the "fill-in". All these algorithms are based on results from Graph Theory. One of the most popular, and in some cases best, algorithms is called the Reverse Cuthill-Mckee method. MATLAB contains a function for this method called `symrcm`. In Python, the function is `scipy.sparse.csgraph.reverse_cuthill_mckee`. There are also various Julia implementations you can find by searching for symrcm.

   Your goal is to use this algorithm on the original matrix A from part (a) to compare the "fill-in" from the Cholesky decomposition of the permuted `A` matrix to the "fill-in" from the original one. Create plots for the sparsity pattern of the permuted A matrix and its Cholesky decomposition. In the latter include the "fill-in".

4. As discussed in the previous problem, Cholesky's method can be used to factor a symmetric positive definite (SPD) matrix A such that $A = R^T R$, where R is an upper triangular matrix. The disadvantage of this method is that it requires n square roots. It was mentioned in class that these square roots could be avoided by factoring A into the form $A = \tilde{R}^T D \tilde{R}$, where D is a diagonal matrix and $\tilde{R}$ is an upper triangular matrix with 1s on the diagonal. Following the same steps we used to derive Cholesky's method in class, derive a procedure for computing the entries in $\tilde{R}$ and D for this new $A = \tilde{R}^T D \tilde{R}$ decomposition.

5. (**Sherman-Morrison formula**) Consider the following linear system

$$\underbrace{(A - \mathbf{u}\mathbf{v}^T)}_{B}\mathbf{x} = \mathbf{b},$$

where A is n-by-n and $\mathbf{u}$ and $\mathbf{v}$ are vectors of length n. The matrix B is obtained through a *rank one* correction of the matrix A. Linear systems of this type appear in several applications [1]. If A is easy to "invert" then there is a nice formula for "inverting" B known as Sherman-Morrison formula [1]:

$$B^{-1} = A^{-1} + \frac{1}{\alpha}A^{-1}\mathbf{u}\mathbf{v}^T A^{-1},$$

where $\alpha = 1 - \mathbf{v}^T A^{-1}\mathbf{u}$. Here we need to assume that $\mathbf{v}^T A^{-1}\mathbf{u} \neq 1$. Note that the Sherman-Morrison formula says the the inverse of the rank one correction to A is a rank one correction of the inverse of A.

(a) Suppose you have a fast algorithm for solving $A\mathbf{y} = \mathbf{c}$. Explain how to use this algorithm to design a fast algorithm for solving $(A - \mathbf{u}\mathbf{v}^T)\mathbf{x} = \mathbf{b}$.

(b) Consider the n-by-n linear system

$$\left(\begin{bmatrix} -2 & 1 & 0 & 0 & 0 & \cdots & 0 \\ 1 & -2 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & -2 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 & 1 & -2 & 1 \\ 0 & \cdots & \cdots & \cdots & 0 & 1 & -2 \end{bmatrix} - \begin{bmatrix} 2 & 2 & 2 & 2 & 2 & \cdots & 2 \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \end{bmatrix}\right) \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ \vdots \\ \vdots \\ p_{n-1} \\ p_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ \vdots \\ b_{n-1} \\ b_n \end{bmatrix} \tag{1}$$

This type of system arises from a second order finite difference approximation to a second order boundary value problem with an integral constraint, e.g.

$$\begin{aligned} \text{Differential equation:} \quad & p''(x) = f(x), \quad 0 \le x \le 1 \\ \text{Boundary condition:} \quad & p(1) = \beta \\ \text{Integral condition:} \quad & \int_0^1 p(x)\,dx = \gamma \end{aligned}$$

For this problem, the right hand side would be

$$b_1 = h^2 f(x_1) - \frac{2\gamma}{h},\ b_j = h^2 f(x_j),\ j = 2, \ldots, n-1,\ b_n = h^2 f(x_n) - \beta,$$

where $h = 1/(n+1)$ and $x_j = jh$, $j = 1, \ldots, n$.

The matrix in (1) is of the type $A - \mathbf{u}\mathbf{v}^T$. Identify the vectors $\mathbf{u}$ and $\mathbf{v}$ in from this system.

(c) Write a code that solves the linear system from part (b) using the fast algorithm you described in part (a). Note that the A in this linear system is tridiagonal, so we can use the Thomas algorithm to factor it in $\mathcal{O}(n)$ operations. However, you can just use a sparse matrix library to solve the system, such as the one provided in MATLAB through the `spdiags` function. Use $f(x) = -2$, $\beta = 0$, $\gamma = 2/3$, and $n = 100$ and plot the solution p. This solution should converge to $p(x) = 1 - x^2$ as $n \to \infty$.

References

[1] W. Hager. Updating the inverse of a matrix. *SIAM Review*, 31(2):221–239, 1989.

[1]This is sometimes also called the Sherman-Morrison-Woodbury formula. However, none of these people were the first to discover it [1].