Math 365 **Homework 3** **Due Oct. 5, 2016**
Grady Wright

Some comments before you begin: Problems 1–4 are intended to sharpen your skills with manipulating matrices and vectors in MATLAB . You should be able to do every problem without using one `for` or `while` loop. In fact, you will not receive full credit if you use a loop in any of these problems! In case you want to brush up on your skills, I recommend reading chapter 2 of "Learning MATLAB" by Toby Driscoll (see the course webpage for a link). Some of the problems below are adapted (or straight copied) from the exercises at the end of chapter 2 of this book.

1. (**Diagonal matrices and their inverse**, 5pts) A N-by-N diagonal matrix is one of the form

$$D = \begin{bmatrix} d_1 & 0 & 0 & \cdots & 0 \\ 0 & d_2 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & d_N \end{bmatrix},$$

where d_1, d_2, ..., d_N are some numbers. The inverse of D is simply

$$D^{-1} = \begin{bmatrix} 1/d_1 & 0 & 0 & \cdots & 0 \\ 0 & 1/d_2 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1/d_N \end{bmatrix},$$

provided no d_1, d_2, ..., d_N is zero.

Diagonal matrices can be constructed quite easily using the `diag` function in MATLAB . For example the code

```
D = diag(1:5)
```

produces the matrix

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 5 \end{bmatrix}$$

Your goal is to write a one-line MATLAB statement for computing the inverse of a D diagonal matrix. You may not use any loops and you may not use the built in function `inv` or any function that does Gaussian elimination, such as 'backslash' `\` or `linsolve`. Also, `D^(-1)` is not acceptable as this uses Gaussian elimination. Illustrate that your code works on the matrix created by `D=diag(2.^(1:5))`.

Hint: Read the documentation on `diag` to learn about the other things it can do.

2. (**Manipulating matrices**, 20pts) There are many useful functions in MATLAB for manipulating matrices and vectors, like the `diag` function from the previous problem. Some other examples include the `tril`, `triu`, `reshape`, and `repmat` functions. The `tril` and `triu` can be used to return any portion of the *lower* and *upper* part of a matrix A, respectively. The `reshape` command allows you to change the dimensions of a matrix from n-by-m to p-by-q, provided $mn = pq$. For example,

```
A = reshape(1:25,[5 5])
```

produces the matrix

$$A = \begin{bmatrix} 1 & 6 & 11 & 16 & 21 \\ 2 & 7 & 12 & 17 & 22 \\ 3 & 8 & 13 & 18 & 23 \\ 4 & 9 & 14 & 19 & 24 \\ 5 & 10 & 15 & 20 & 25 \end{bmatrix}$$

The `repmat` command allows you to create a matrix with a repeating pattern. For example,

```
a = 1:2:9;
A = repmat(a,[5 1])
```

produces the matrix

$$A = \begin{bmatrix} 1 & 3 & 5 & 7 & 9 \\ 1 & 3 & 5 & 7 & 9 \\ 1 & 3 & 5 & 7 & 9 \\ 1 & 3 & 5 & 7 & 9 \\ 1 & 3 & 5 & 7 & 9 \end{bmatrix}$$

Note that this is just the row vector `a` repeated 5 times.

Use the above functions, or any combination of them, in the problems below. Your code may not use *any* loops and you may not just input these matrices directly in MATLAB . However, you may use any operations such as addition, subtraction, and multiplication. You may also use the additional MATLAB functions `diag`, `eye`, and `transpose`. The code you produce should require no more than two lines and should generalize to any size matrix with the same pattern.

(a) Produce the matrices

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 \\ 3 & 8 & 0 & 0 & 0 \\ 4 & 9 & 14 & 0 & 0 \\ 5 & 10 & 15 & 20 & 0 \end{bmatrix}, \quad \begin{bmatrix} 0 & 0 & 3 & 4 & 5 \\ 0 & 0 & 0 & 4 & 5 \\ 0 & 0 & 0 & 0 & 5 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad \begin{bmatrix} 0 & 0 & 3 & 4 & 5 \\ 2 & 0 & 0 & 4 & 5 \\ 3 & 8 & 0 & 0 & 5 \\ 4 & 9 & 14 & 0 & 0 \\ 5 & 10 & 15 & 20 & 0 \end{bmatrix}$$

(b) Produce the matrices

$$\begin{bmatrix} 2 & 4 & 6 & 8 & 10 \\ 2 & 4 & 6 & 8 & 10 \\ 2 & 4 & 6 & 8 & 10 \\ 2 & 4 & 6 & 8 & 10 \\ 2 & 4 & 6 & 8 & 10 \end{bmatrix}, \quad \begin{bmatrix} 25 & 20 & 15 & 10 & 5 \\ 24 & 19 & 14 & 9 & 4 \\ 23 & 18 & 13 & 8 & 3 \\ 22 & 17 & 12 & 7 & 2 \\ 21 & 16 & 11 & 6 & 1 \end{bmatrix}, \quad \begin{bmatrix} 0 & 20 & 15 & 10 & 5 \\ 24 & 0 & 14 & 9 & 4 \\ 23 & 18 & 0 & 8 & 3 \\ 22 & 17 & 12 & 0 & 2 \\ 21 & 16 & 11 & 6 & 0 \end{bmatrix},$$

$$\begin{bmatrix} -1 & 20 & 15 & 10 & 5 \\ 24 & -1 & 14 & 9 & 4 \\ 23 & 18 & -1 & 8 & 3 \\ 22 & 17 & 12 & -1 & 2 \\ 21 & 16 & 11 & 6 & -1 \end{bmatrix}$$

(c) Produce the matrices

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 4 & 4 & 4 & 4 \\ 1 & 4 & 9 & 9 & 9 \\ 1 & 4 & 9 & 16 & 16 \\ 1 & 4 & 9 & 16 & 25 \end{bmatrix}, \quad \begin{bmatrix} 0 & 4 & 9 & 16 & 25 \\ -4 & 0 & 9 & 16 & 25 \\ -9 & -9 & 0 & 16 & 25 \\ -16 & -16 & -16 & 0 & 25 \\ -25 & -25 & -25 & -25 & 0 \end{bmatrix}, \quad \begin{bmatrix} -1 & 4 & 9 & 16 & 25 \\ 4 & -4 & 9 & 16 & 25 \\ 9 & 9 & -9 & 16 & 25 \\ 16 & 16 & 16 & -16 & 25 \\ 25 & 25 & 25 & 25 & -25 \end{bmatrix}$$

(d) Produce the matrix (note the block pattern)

$$\begin{bmatrix} 1 & 16 & 49 & 1 & 16 & 49 \\ 4 & 25 & 64 & 4 & 25 & 64 \\ 9 & 36 & 81 & 9 & 36 & 81 \\ 1 & 16 & 49 & 1 & 16 & 49 \\ 4 & 25 & 64 & 4 & 25 & 64 \\ 9 & 36 & 81 & 9 & 36 & 81 \end{bmatrix}$$

3. (**Toeplitz matrices**, 10pts)

(a) The two matrices A and B below are examples of what are called Toeplitz matrices, which are matrices which are constant along their diagonals. These matrices occur quite often in applications. Read the online help for the MATLAB function `toeplitz` and use this function to produce A and B below. Note: use vector concatenation and the function `zeros` instead of typing all those zeros. Your code should take one line to produce A and one to produce B.

$$A = \begin{bmatrix} -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -2 \end{bmatrix},$$

$$B = \begin{bmatrix} -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -2 \end{bmatrix}$$

This should again require no loops and exactly one line.

(b) Now use `toeplitz` to create the matrices

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 0 & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ 0 & 0 & 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 0 & 0 & 0 & 1 & 2 & 3 & 4 \\ 0 & 0 & 0 & 0 & 0 & 1 & 2 & 3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix},$$

$$B = \begin{bmatrix} 1 & -4 & 7 & -10 & 13 & -16 & 19 \\ -3 & 1 & -4 & 7 & -10 & 13 & -16 \\ 5 & -3 & 1 & -4 & 7 & -10 & 13 \\ -7 & 5 & -3 & 1 & -4 & 7 & -10 \\ 9 & -7 & 5 & -3 & 1 & -4 & 7 \\ -11 & 9 & -7 & 5 & -3 & 1 & -4 \\ 13 & -11 & 9 & -7 & 5 & -3 & 1 \end{bmatrix}$$

You may not use `triu` or `tril` to construct A and your code needs to use one line. Your code for B should not require explicitly typing a vector and should not use any loops. It should be written so that only one or two small changes are required to produce a similar B of any size.

4. (**Max, min, and logical indexing**, 10pts) Create a 9×9 matrix with "random" entries using `rand(9)`. Call this matrix A.

   (a) Write a one line MATLAB statement for finding the maximum value in each row of A

   (b) Write a one line MATLAB statement for finding the minimum value in each column of A

   (c) Write a one line MATLAB statement for finding the maximum value in all of A.

   (d) Use the "logical indexing" capabilities of MATLAB to set all values in A that are less than 0.5 to 0. Again, a one-liner.

   Include the results for (a)–(d) in your homework report.

5. (**Timing a linear solve.**, 25pts) In this problem, you will investigate the cost of solving linear systems and compare your results to the theoretical computational cost we have discussed in class.

   Before starting this problem, you will need the following information about the laptop or PC you are working on.

   - The amount of RAM in your computer. Typical values are betweem 4 and 12 GB (gigabytes).
   - The clock speed of your PC or laptop. A typical value is 2.5 GHz (gigahertz).

   (a) **Experimental setup.** Determine a matrix size that you can comfortably fit into your available RAM. For example, if you have a 4 GB machine, you should be able to comfortably store a matrix that occupies about 800MB. Store this value in a variable "`Mb`". Use the following information to compute a maximum matrix dimension N that you can store in `Mb` megabytes of memory.

      - A megabyte has 1024 kilobytes
      - A kilobyte is 1024 bytes
      - A floating point number is 8 bytes.
      - An $N \times N$ matrix contains N^2 floating point numbers.

      Call the N you compute '`nmax`' (N_{max}).

(b) **Calibrate the timing experiment.** We need to "calibrate" our experimental set up by timing an operation whose flop count is easy to compute. Create two random matrices A and B each of size $N_{max} \times N_{max}$. Using the MATLAB functions `tic` and `toc`, determine how much time (seconds) it takes to compute the product AB. Determine the number of floating point operations (additions and multiplications) it takes to compute the $N_{max} \times N_{max}$ matrix-matrix product (see the Linear Algebra lecture slides). Use this number to estimate the number of floating point operations per second ('flops') your computer can carry out. Call this flop rate '`flops`'.

Compare this number to the theoretical number obtained using your computer's clock speed. The following information might be useful.

- A gigahertz is 10^9 hertz.
- A hertz is equal to one clock cycle.
- A typical microprocessor (CPU) can compute 4 flops per clock cycle.

If you have a dual or quad core machine, you may also want to investigate whether Matlab is making use of multiple cores.

(c) **Time a dense linear solve.** Create an integer sequence of values `Nvec` between, say, $N = 100$ and the N_{max} you found above. To generate this sequence, it is a good idea to use the `logspace` command.

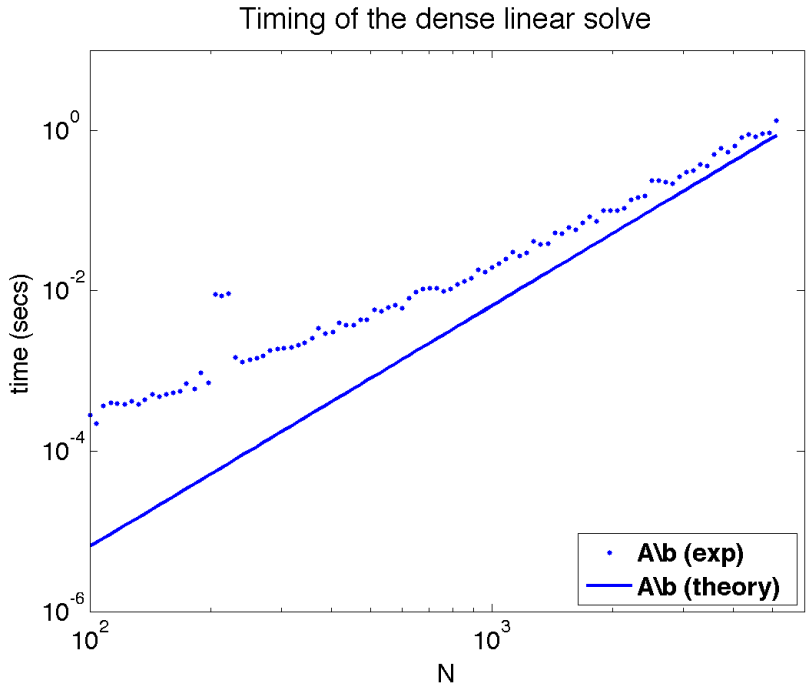

Figure 1: Timing plot for problem 3. The x-axis is the matrix size N ($N \times N$ matrix), and the y-axis is the time (in seconds) as measured by `tic` and `toc` for solving a linear system of size N.

Using a `for` loop, loop over the entries N_i of the sequence. In each pass through the loop, create a random matrix A of size $N_i \times N_i$, and a random right hand side vector $\mathbf{b}$ of size $N_i \times 1$. Then, using `tic` and `toc`, time how long it takes to solve the system $A\mathbf{x} = \mathbf{b}$ using the backslash operator. Store this time as the i^{th} entry in a vector '`lutimes`'.

(d) On a **log-log** plot, plot the time values you found above (stored in `lutimes`) verses the N_i in your sequence of N values. On the same set of axis, plot the curve of the theoretical time estimated by the operation count we discussed in class. Be sure to use the flop rate '`flops`' you computed above to get a time (in seconds) from an operation count. The two plots should be very close. Be sure to label your plots. Use the Matlab `legend` command to identify each curve that you get. Your plot should look something like the figure shown in Figure 1.

(e) (**Extra Credit - 5 points**) Visit the website http://www.top500.org and answer the following questions.

- Using the value from the 'Linpack Benchmark', what is the performance of the world's fastest supercomputer, measured in GFlops ($= 10^9$ flops)?
- How much faster is the world's fastest computer than your computer or laptop?
- How recently would your computer have made it on the Top 500 list of the world's fastest computers?
- When can we expect to see an exaflop machine?

The 'Linpack Benchmark' solves a dense linear system of equations, very much like what you have done for this problem.

6. (**Heat transfer**, 30pts) Imagine that we have a metal rod of length L (m) whose temperature at each end is held fixed at $20C^{\circ}$ (293.15 K) and which is being heated by a flame held at the midpoint. We can approximate the steady state temperature at equally spaced points along the rod using the following model. We divide the rod into N intervals of equal length h as shown in the figure below. This partitioning of the rod gives us $N+1$ equally spaced points

$$x_j = hj, \qquad h = \frac{L}{N}, \qquad j = 0, 1, 2, \ldots N$$

The steady state temperature T_j (in Kelvin) at interior points x_j in the rod can be modeled as

$$T_j = \frac{T_{j+1} + T_{j-1}}{2} + h^2 f(x_j), \qquad j = 1, 2, \ldots, N-1,$$

where $T_0 = T_N = 293.15K$. The flame $f(x)$ is given by

$$f(x) = \frac{S}{2\rho c_p \beta} \exp\left(-\left(\frac{x - L/2}{\varepsilon}\right)^2\right)$$

where

- S is the heating rate for the flame (W/m^3)
- β is the thermal diffusivity of the metal (m^2/s)
- c_p is the heat capacity of the metal $(J/(kg \cdot K))$
- ρ is the density of the metal (kg/m^3)
- ε is a scaling factor specifying the flame width (m)

See the figure below for a schematic of this problem.

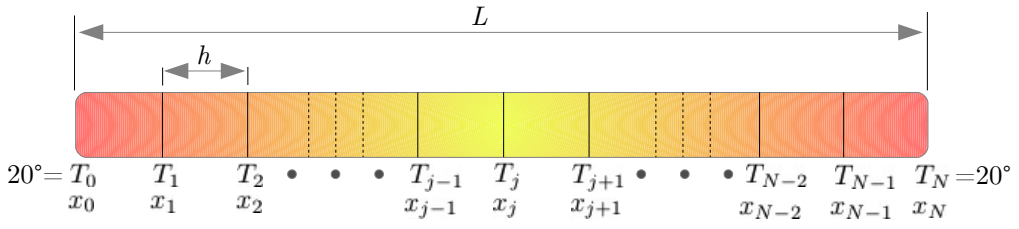

Figure 2: Schematic for the heated metal rod.

(a) Using either the `tridisolve` (from the NCM course materials) or `spdiags` function and backslash (\) in MATLAB , set up a linear system to solve for the steady state temperature distribution in the rod of length $L = 1$ and $N = 200$. This linear system should be of size 199-by-199 to compute the solution at all the interior nodes.

(b) Solve the system for a thermal diffusivity of $\beta = 10^{-3}$, heat rate $S = 1 \times 10^4$, heat capacity of $c_p = 200$, density of $\rho = 10$ and a scaling factor $\varepsilon = 5 \times 10^{-2}$.

Check your solution : The temperature T_2 is 295.3655673 K and T_3 is 296.47335097 K.

(c) Plot the resulting temperature distribution as a function of x. Be sure to add labels and a legend to your graph.

(d) How close to the center of the rod could you touch with your finger without getting burned? Give the value of x and indicate where this point is on the graph from the previous part. Note that you need to find the approximate temperature at which skin burns (be reasonable here, and assume your skin will burn in 1 second or less of being placed on the rod).

7. (**PageRank**, 20 points) Download the `bsusurfer.m` function from the course webpage. This is modified version of the NCM function `surfer.m` that restricts the webpages it visits to those that contain the word *boisestate* in their address (note that it is by no means perfect).

(a) Using the `bsusurfer.m` function collect 1000 webpages. This task will take awhile to complete, so it is best to do it once and save the results. You can do this with the MATLAB commands:

```
[U,G] = bsusurfer(n);
save BSUSurferResults U G;
```

When you want to use `U` and `G` later, you can simply use the command

```
load BSUSurferResults;
```

which loads `U` and `G` into the workspace.

(b) Display the connectivity matrix for this collection of webpages using the `spy` command and compute and display its sparsity ratio in the title of the plot.

(c) Using the function `pagerank.m` posted on the course webpage, compute the PageRank of the webpages from (a) with the default damping factor $\alpha = 0.85$ (the book and function call this value p).

(d) Report the 10 pages with the highest PageRank and the 10 pages with the lowest PageRank. By default the `pagerank` function displays the 10 pages with the highest PageRank. You need to figure out how to make it also display the lowest ranked pages.

(e) Repeat part (c) and (d), but with a damping factor of $\alpha = 0.95$.

(f) Compare the results of part (e) with part (d). (Note that compare means to actually write something meaningful about the differences or similarities you see).

For reference see the slides on PageRank posted on the course webpage and refer to Section 2.11 of the NCM book.

8. (**Reducing fill-in**, 10pts) As we discussed briefly in class (see also problem 2.5 from the book), if A is a symmetric positive definite matrix then A can be decomposed into the form $A = LL^T$, where L is a lower triangular matrix (or $A = R^T R$, where R is and upper triangular matrix). This decomposition is called the Cholesky decomposition.

For this problem you will use the Cholesky function built into MATLAB (`chol`), which is a very sophisticated implemenation 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 "bcsstk36.mat" from the course web page and save it to your working directory in MATLAB . This file contains a sparse matrix that arises from a structural analysis of an automobile shock absorber assembly[1]. Load the matrix into matlab using the following series of commands:

[1] see http://www.cise.ufl.edu/research/sparse/matrices/Boeing/bcsstk36.html

```
>>load bcsstk36;
>>A = Problem.A;
```

Use the `spy` command in MATLAB to generate a plot showing the sparsity pattern of the matrix `A`. Compute the sparsity ratio of the `A` matrix by comparing the number of non-zero entries in `A` (use the `nnz` command) to the total number of entries in `A`. Report this number 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) using the "lower" option in the `chol` function (this is faster for sparse matrices than the default setting). Plot the sparsity pattern of the lower 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 lower 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 more popular (and good) of these algorithms is the Symmetric Approximate Minimum Degree Permutation method. MATLAB contains a function for this method called `symamd`. Your goal is to use this function 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".