AMATH740/CM770/CS770
Fall 2020: Computational Assignment B

Instr.: Hans De Sterck e-mail: *hdesterck@uwaterloo.ca* Office hrs: Wed 9:00-9:45am, Thu 11am-12
TA.: Mohammad Aali e-mail: *mohammad.aali@uwaterloo.ca* Office hrs: Tue 11am-12, Thu 9-10am

Assignment due date: Monday November 16, 5:00pm (Waterloo time), Crowdmark (for the answer package) and LEARN (for the program code files)

Submission instructions: Dual submission

- **LEARN submission**: Please submit your program code files to the Dropbox on LEARN (only the program code files; no program output or answers to questions).
- **Crowdmark submission**: Please submit the answers to each assignment question on Crowdmark, including written or typed answers to the questions asked, with relevant computer output (plots, tables or number output) as requested in the assignment questions. Please also include, for each question, a pdf file or screenshot with your program code (to facilitate TA feedback on your code).
- one common submission per group
- MATLAB is the recommended computer language, but you can choose a different language like python or Julia or C or C++ (contact the instructor if you want to choose a language not listed before; Maple or Mathematica are not suitable).
- The questions are written assuming you use MATLAB, and some short MATLAB code fragments are provided on LEARN for download, but it should not be difficult to translate this to another language like python, and I don't think there should be much extra work when using another language; let me know if you encounter issues or have questions about this
- there are some potentially useful tools for programming in teams that you may consider to use, such as version control tools like git (e.g., you can create a git repository for your group on github or bitbucket) (but the computer code you will write for the programming questions will generally not be long or complicated, so it is by no means necessary to use that kind of tools); collaboration tools like slack could also be useful (or you can communicate and share files via tools from Google, or MS Teams, etc.)

1. Implementation of the Gauss-Seidel method. (10 marks)
   Implement the Gauss-Seidel algorithm of Eq. (4.8) in the course notes in MATLAB using only primitive commands (*i.e.* do not appeal to the built-in linear algebra functionality in MATLAB). You should use the function headers given below and submit the code in separate m-files to the LEARN drop box on the course webpage. You can implement the algorithm for a general dense matrix A (not assuming A is sparse), and you can directly use the formulation as in Eq. (4.8) (there is no need to implement this using the interpretation of a stand-alone preconditioner, as in Eq. (4.6)).

```
function u = GaussSeidel(A, u0, f, tol, maxit)
% Usage: u = GSSequence(A, u0, f, tol, maxit)
% Perform a sequence of Gauss-Seidel iterations on
% A u = f using the initial estimate u0, until the 2-norm
% of the residual has been reduced by a factor
% of at least tol, or until the number of iterations
% reaches maxit.
```

You can download `VerifyGaussSeidel.m` from LEARN to check your implementation. Please report on the output for input $n = 100$. (It should be a small number, with magnitude around 1e-14 or so.)
Also, include a textual copy of your code in the assignment package submitted on Crowdmark.
Note: to obtain good performance for large problems (see next question), it may be best to vectorize the implementation, which allows you to avoid the inner loop of the double loop in the Gauss-Seidel algorithm.

2. Gauss-Seidel applied to a special dense matrix. (10 marks)
Consider the $n \times n$ linear system

$$\begin{pmatrix} n & 1 & 1 & \cdots & 1 \\ 1 & n & 1 & \cdots & 1 \\ 1 & 1 & n & \cdots & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & 1 & \cdots & n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_4 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 3 \\ \vdots \\ n \end{pmatrix}. \tag{1}$$

Write a Matlab script that performs the following computations, and provide the output and answer the questions in your Crowdmark submission:

(a) Compute the solution to this system for $n = 250, 500, 1000, 2000, 4000, 8000, 16000$ using LU decomposition (you can use Matlab's `linsolve`). (You can use smaller n if your computer runs out of memory.) Determine and tabulate the amount of compute time required for each value n. (Note that Matlab's `linsolve` uses a highly optimized implementation in a high-level programming language that is much faster than execution of code written in the Matlab language itself.)

(b) Compute the solution to this system using your implementation of Gauss-Seidel from the previous question for $n = 250, 500, 1000, 2000, 4000$ with $x_0 = [0\ 0\ 0\ \cdots\ 0\ 0]^T$ and $tol_{rel} = 10^{-8}$. Determine the amount of compute time required for this solution. How many iterations do you need to achieve convergence as a function of n? (Note that what you find here is very specific for this special type of matrix.)

(c) Compare the results from parts (a) and (b). How do you expect the execution time of each algorithm to scale asymptotically as a function of n? To which extent does this agree with what you observe numerically (compute and tabulate ratios of consecutive execution times as n doubles)? Which algorithm/implementation would be best for small or medium problem sizes? And for very large problem sizes?

Submit your code to LEARN, and include a textual copy of your code in the assignment package submitted on Crowdmark.

3. Implementation of the steepest descent and conjugate gradient methods. (15 marks)
You are asked to implement the steepest descent and conjugate gradient methods for $A\vec{x} = \vec{b}$ in Matlab, and apply the methods to the 2D model problem (you can download `build_laplace_2D_kron.m` from LEARN to build the 2D Laplacian matrix).

(a) Implement the following in m-file `steepest.m`:

```
function [x res steps]=steepest(A,x0,b,tol,maxit)
% Performs a sequence of steepest descent iterations on
% A x = b using the initial estimate x0, until the 2-norm
% of the residual is smaller than tol (relative
% to the initial residual), or until the number of iterations
% reaches maxit. 'steps' is the number of steps
% that were performed, and 'res' is a vector with the
% residual size after every interation (the 2-norm of
% the residual vector).
```

(b) Implement the following in m-file `conjGrad.m`:

```
function [x res steps]=conjGrad(A,x0,b,tol,maxit)
% Performs a sequence of conjugate gradient iterations
% on A x = b using the initial estimate u0, until the 2-norm
% of the residual is smaller than tol (relative
% to the initial residual), or until the number of iterations
% reaches maxit. 'steps' is the number of steps
% that were performed, and 'res' is a vector with the
% residual size after every interation (the 2-norm of
% the residual vector).
```

Submit your m-files `steepest.m` and `conjGrad.m` to the LEARN drop box, and provide a printout in your assignment package to be submitted in Crowdmark.

(c) Download `test_steepest_cg.m` to test the correctness of (a) and (b). Report on the error and number of steps for each method. (The errors should be smaller than 10^{-10}.)

(d) Make a driver program `CG_driver.m` that applies `steepest.m` and `conjGrad.m` to $A\vec{x} = \vec{b}$ with A being the 2D model problem matrix generated by `build_laplace_2D_kron.m`, and $\vec{b}$ a vector with all twos. Use maxit=500 and tol=1e-8. Generate a plot in which you compare, for $N = 32$, the residual convergence for the steepest descent and conjugate gradient methods (starting from a zero initial guess), as a function of the iteration. For each of the methods, plot the 10-log of the residuals as a function of iteration number. Which method requires the least iterations to obtain an accurate solution, and is this as you expected?

(e) Provide a table in which you list, for steepest descent and conjugate gradient, how many iterations you need to reduce the residual by $1e-6$, for $N = 16, 32, 64$. (You can use `maxit`= $20,000$.) What are the total number of unknowns and the approximate condition number for each problem size? (You can use $\kappa \approx 4n/\pi^2$, where $n = N^2$, which can be derived from a formula for the eigenvalues of A.) For each method, do you see the expected behaviour in the number of required iterations as a function of the total number of unknowns? (Explain.) Using these numerical results, briefly discuss the computational cost/complexity of each of the methods as a function of the total number of unknowns (discuss the cost per iteration, the number of iterations required, and the total cost, as a function of total problem size). Which method is best for large problem size? (Use what you learned in Example 4.14 of the course notes.)

4. Implementation of the preconditioned CG and GMRES methods. (15 marks)
You are asked to implement the preconditioned CG and GMRES methods for $A\,\vec{x} = \vec{b}$ in Mat-

lab, and apply the methods to the 2D model problem (download `build_laplace_2D_kron.m` from LEARN).

(a) Implement the following in m-file `myGMRES.m`:

```
function [x res steps]=myGMRES(A,x0,b,tol,maxit)
% Performs a sequence of GMRES iterations on
% A x = b using the initial estimate x0, until the 2-norm
% of the residual is smaller than tol (relative
% to the initial residual), or until the number of iterations
% reaches maxit. ‘steps’ is the number of steps
% that were performed, and ‘res’ is a vector with the
% residual size after every interation (the 2-norm of
% the residual vector).
```

(b) Implement the following in m-file `myGMRES_SSOR.m`:

```
function [x res steps]=myGMRES_SSOR(A,x0,b,tol,maxit)
% Performs a sequence of right-preconditioned GMRES iterations
% on A x = b with the initial estimate x0, using the SSOR preconditioner,
% until the 2-norm of the residual is smaller than tol (relative
% to the initial residual), or until the number of iterations
% reaches maxit. ‘steps’ is the number of steps
% that were performed, and ‘res’ is a vector with the
% residual size after every interation (the 2-norm of
% the residual vector).
```

(c) Implement the following in m-file `myCG_SSOR.m`:

```
function [x res steps]=myCG_SSOR(A,x0,b,tol,maxit)
% Performs a sequence of preconditioned CG iterations
% on A x = b with the initial estimate x0, using the SSOR preconditioner,
% until the 2-norm of the residual is smaller than tol (relative
% to the initial residual), or until the number of iterations
% reaches maxit. ‘steps’ is the number of steps
% that were performed, and ‘res’ is a vector with the
% residual size after every interation (the 2-norm of
% the residual vector).
```

Implementation notes:

- for GMRES, you can solve the small least-squares problem min $\|M\vec{y} - \vec{f}\|$ by using Matlab’s backslash operator as in $\vec{y} = M\backslash\vec{f}$ (which solves over-determined systems in the least-squares sense using QR decomposition), or you can use the normal equations
- to build the preconditioner, you can use `tril` and `triu` to extract the strictly lower and upper triangular parts of A; you can use `spdiags` to extract the diagonal of A, and again to build a sparse diagonal matrix containing the diagonal of A
- take $\omega = 1.9$ in SSOR
- as was discussed in the course notes, since for SSOR the matrix P contains inverses of sparse triangular matrices, it is a bad idea to form P explicitly, because inverting

the sparse triangular matrices will result in dense matrices, and also, inverting a matrix is in general substantially more expensive than solving a linear system with that matrix; so rather than forming the P matrices directly and computing the preconditioned residual $P\vec{r}_k$ as a matrix-vector product, you should compute the preconditioned residual $\vec{q}_k$ in

$$\vec{q}_k = P\vec{r}_k$$

by solving the linear system

$$P^{-1}\vec{q}_k = \vec{r}_k$$

with P^{-1} a product of sparse matrices; to solve this system for the SSOR preconditioner, you need to do two consecutive triangular solves, and you can again use backslash in Matlab for each of these solves (which recognizes sparse and triangular matrices, and solves the system uses forward or backward substitution, as appropriate), or use forward and backward substitution solvers

Submit your m-files to the LEARN drop box, and provide a printout in your assignment package to be submitted in Crowdmark.

(d) Download `test_iterative.m` to test the correctness of (a–c). Report on the errors and number of steps. (The errors should be close to 1e-12.)

(e) Make a driver program `driverPCG_PGMRES.m` that applies `myGMRES.m`, `myGMRES_SSOR.m`, and `myCG_SSOR.m` to $A\vec{x} = \vec{b}$ with A being the 2D model problem matrix generated by `build_laplace_2D_kron.m`, and $\vec{b}$ a vector with all ones. Use maxit=400 and tol=1e-10. Generate a plot in which you compare, for $N = 32$, the residual convergence for the three methods (starting from a zero initial guess), as a function of the iteration. On this plot, for each of the methods, plot the 10-log of the residuals as a function of iteration number. Also compare with CG, using your file `conjGrad.m` from Question 4 (or, a simplification of `myCG_SSOR.m`, without preconditioner).

(f) Extending `driverPCG_PGMRES.m`, for problem sizes `N` equal to `[80 90 100 110 120 130 140]`, make a `loglog` plot of the number of iterations required for convergence as a function of total problem size $n = N^2$, for each of the 4 methods (CG, CG+SSOR, GMRES, GMRES+SSOR), all on the same figure. Use the `polyfit` command to fit a polynomial of degree 1 to the measured (n, iterations) points of problem size versus iterations, for each of the methods. The fitted slope for CG and GMRES should be close to 1/2, because the number of iterations is $O(\sqrt{n})$ for these methods. It can be shown that SSOR is an *optimal* preconditioner for CG for the 2D model problem in terms of asymptotic order, with number of iterations $O(n^{1/4})$.
Note: This means that the total work is $O(n^{5/4})$ when using the SSOR preconditioner, and $O(n^{3/2})$ without it (both for CG and GMRES). GMRES gives very similar results for the 2D model problem than CG, in terms of the number of required iterations. However, GMRES is much more versatile since it can also be applied to non-SPD systems.

5. Question on neural network implementation will be added later this week as the topic is being covered in class. (xx marks)

6. Question on neural network implementation will be added later this week as the topic is being covered in class.. (xx marks)