

# CS412 Spring Semester 2013

## Homework Assignment #4

Due Thursday April 25th 2013, in class

Sum of all problems : 120%, Maximum possible score : 100%.

1. [50%] In this problem you will use **MATLAB** to assess in practice the problems that arise when attempting to solve an ill-conditioned linear system of equations. For each of the values  $k = 5, 6, 7, \dots, 14, 15$  perform the following:

- Form the Hilbert matrix  $\mathbf{H}_n$  with entries  $[H_n]_{ij} = \frac{1}{i+j-1}$ . You can use the **MATLAB** function `hilb(k)` to generate this matrix.
- Define the test vector

$$\mathbf{x}_k = \begin{pmatrix} \sin(1) \\ \sin(2) \\ \vdots \\ \sin(k) \end{pmatrix}.$$

- Construct the vector  $\mathbf{y}_k = \mathbf{H}_k \mathbf{x}_k$
- Compute the vector  $\mathbf{z}_k = \mathbf{H}_k^{-1} \mathbf{y}_k$  using the **MATLAB** backslash operator (`\`). In theory,  $\mathbf{z}_k$  should be identical to  $\mathbf{x}_k$ , but this will not be the case here, due to numerical errors.
- Compute the absolute error  $e_k = \|\mathbf{x}_k - \mathbf{z}_k\|_\infty$ . You can use the **MATLAB** function `norm(v, inf)` to compute the infinity norm of a vector  $\mathbf{v}$ .

Plot the error values  $e_k$  as a function of the matrix size  $k$ . Use a logarithmic axis for the error, to make this plot easier to parse; the **MATLAB** function `semilogy` can be used for this purpose: The syntax is the same as the `plot` function, but it generates a plot with a logarithmic  $y$ -axis.

Turn in a printout of this plot, along with the code used to generate it.

2. [70%] In this problem you will compare the LU decomposition to the Gauss-Seidel method, as two alternative methodologies for solving a certain *sparse* linear system  $\mathbf{A}\mathbf{x} = \mathbf{b}$ . You will also be exposed to **MATLAB**'s notion of a *sparse* matrix, as well as the `tic/toc` commands for measuring the run time of numerical procedures.

First let us define a benchmark linear system, on which the two different methodologies will be evaluated. For any value of the integer  $n$ , we define an  $n \times n$  linear system of equations  $\mathbf{A}_n \mathbf{x}_n = \mathbf{b}_n$ .

The matrix  $\mathbf{A}_n$  has the form:

$$\mathbf{A}_n = \begin{pmatrix} -3 & 1/n & 1/n & 1/n & 1/n & \cdots & 1/n & 1/n & 1/n \\ 1/n & -3 & 1 & & & & & & \\ 1/n & 1 & -3 & 1 & & & & & \\ 1/n & & 1 & -3 & 1 & & & & \\ 1/n & & & 1 & -3 & 1 & & & \\ \vdots & & & & \ddots & \ddots & \ddots & & \\ \vdots & & & & & \ddots & \ddots & \ddots & \\ 1/n & & & & & & 1 & -3 & 1 \\ 1/n & & & & & & & 1 & -3 \end{pmatrix} \in \mathbf{R}^{n \times n}$$

The right hand side  $\mathbf{b}_n$  is given by:

$$\mathbf{b}_n = \begin{pmatrix} -2 - 1/n \\ -2 + 1/n \\ -1 + 1/n \\ -1 + 1/n \\ \vdots \\ -1 + 1/n \\ -1 + 1/n \\ -2 + 1/n \end{pmatrix} \in \mathbf{R}^n$$

This specific  $\mathbf{b}_n$  is constructed in such a way, that the solution  $\mathbf{x}_n$  will be a vector with all entries equal to one.

You can find a MATLAB function that builds  $\mathbf{A}_n, \mathbf{b}_n$  under the link:

[http://pages.cs.wisc.edu/~cs412-1/hw/test\\_problem\\_hw4.m](http://pages.cs.wisc.edu/~cs412-1/hw/test_problem_hw4.m)

This MATLAB M-file implements a function `test_problem_hw4`, which returns 5 results, as follows:

$$[\mathbf{A} , \mathbf{b} , \mathbf{D} , \mathbf{L} , \mathbf{U}] = \text{test\_problem\_hw4}(n)$$

Here,  $n$  is the desired size of the problem,  $\mathbf{A}, \mathbf{b}$  are the matrix  $\mathbf{A}_n$  and right hand side  $\mathbf{b}_n$ , respectively. The matrices  $\mathbf{D}, \mathbf{L}, \mathbf{U}$  are the ones used in the decomposition of  $\mathbf{A}_n$  as  $\mathbf{A} = \mathbf{D} - \mathbf{L} - \mathbf{U}$ , used in the Jacobi or Gauss-Seidel procedure.

- (a) [10%] The matrix  $\mathbf{A}$  returned by the provided function is created and stored in MATLAB's internal format for *sparse* matrices, where only the non-zero entries of  $\mathbf{A}$  are kept in memory. We can convert from this sparse format to a full (dense) matrix using the built-in function `full(A)`.

For  $n=100, 1000, 10000$  have MATLAB compute the matrix-vector product  $\mathbf{A}_n \mathbf{b}_n$  using (a) the sparse matrix  $\mathbf{A}$  returned by the provided function, and (b) the equivalent dense matrix `A_full=full(A)`. Report the elapsed time for each matrix-vector multiplication, measured using MATLAB's `tic/toc` commands. The general mode of use for these commands is:

```
tic; <some commands> ; t=toc
```

At the end of this statement, `t` will contain the time (in seconds) that the commands which intervened between the `tic/toc` construct took to execute.

- (b) [25%] For  $n=100, 1000, 2000$  use the LU decomposition to solve the system  $\mathbf{A}_n \mathbf{x}_n = \mathbf{b}_n$ . For every value of  $n$ , use the `tic/toc` commands to measure how much time was required to compute the LU factorization, and solve the two forward/backward substitution problems to determine  $\mathbf{x}_n$ . It is recommended that you test the computed solution against the exact solution (the vector of all ones) as a sanity check.

**Note:** Beware of the fact that the matrices  $\mathbf{L}$  and  $\mathbf{U}$  in the expression  $\mathbf{A}=\mathbf{D}-\mathbf{L}-\mathbf{U}$  are *not* the same as the factors produced by the LU decomposition of  $\mathbf{A}$ . If we denote the latter by  $\hat{\mathbf{L}}, \hat{\mathbf{U}}$  for clarity, then these two matrices satisfy  $\mathbf{A} = \hat{\mathbf{L}}\hat{\mathbf{U}}$  and need to be computed using MATLAB's `lu` function as discussed in class.

- (c) [35%] For  $n=100, 1000, 2000, 10000$  use 10 iterations of the Gauss-Seidel method to generate an approximation for the solution  $\mathbf{x}_n$ . At each step, you will need to solve the lower-triangular system

$$(\mathbf{D} + \mathbf{L})\mathbf{x}^{(k+1)} = \mathbf{U}\mathbf{x}^{(k)} + \mathbf{b}$$

to generate the next iterate in your approximation sequence (you may use the MATLAB backslash operator to solve this triangular system). Use the `tic/toc` commands to measure how long MATLAB took to complete these 10 Gauss-Seidel iterations. You can use the vector of all zeroes as your initial guess.

Also, report the infinity-norm of the *error* after completing the 10 Gauss-Seidel iterations, for each problem size. The error vector can be computed by subtracting the known exact solution (vector of all ones) from the computed approximation.