交大資工 110550062
朱致伶

Numerical Methods, 2023 Spring

Assignment 2

1. (25%) Solve the system

$$2.51x + 1.48y + 4.53z = 0.05,$$
$$1.48x + 0.93y - 1.30z = 1.03,$$
$$2.68x + 3.04y - 1.48z = -0.53.$$

(a) Use Gaussian elimination, but use only three significant digits and do no interchanges. Observe the small divisor in reducing the third column. The correct solution is $x = 1.45310$, $y = -1.58919$, $z = -0.27489$.
(b) Repeat part (a) but now do partial pivoting.
(c) Repeat part (b) but now chop the numbers rather than rounding.
(d) Substitute the solutions found in (a), (b), and (c) into the equations. How welldo these match the original right-hand sides?

The implement of part (a), (b), (c) are in the file Q1_110550062.m

With the results on the right, it shows that Gaussian elimination with partial pivoting have more precise resolution compared with no interchange. However, the two methods in (a) and (b) have the values approximately the same as the ones on original right-hand sides, which match well.

The solutions in (c) have the worst results among the three. The difference of right-hand sides value and calculated solutions are at the third decimal place, which is unignorable and large.

```
>> Q1_110550062
Part (a) - Gaussian elimination without interchanges
Solution:
   1.453100102690179  -1.589194864435479  -0.274894670725793

RHS values:
   0.050000000000000   1.030000000000000  -0.530000000000004

Part (b) - Gaussian elimination with partial pivoting
Solution:
   1.453100102690178  -1.589194864435477  -0.274894670725793

RHS values:
   0.050000000000000   1.030000000000000  -0.530000000000000

Part (c) - Gaussian elimination with partial pivoting and chopping
Solution:
   1.453000000000000
  -1.589000000000000
  -0.274000000000000

RHS values:
   0.054090000000000   1.028870000000000  -0.531000000000000
```

2. (25%) The system of Exercise 34 is an example of a symmetric matrix. Because the elements at opposite positions across the diagonal are exactly the same, it can be stored as a matrix with n rows but only three columns. (Please submit your code to E3)

▶34. Given this tridiagonal system:

$$\begin{bmatrix} 4 & -1 & 0 & 0 & 0 & 0 & 100 \\ -1 & 4 & -1 & 0 & 0 & 0 & 200 \\ 0 & -1 & 4 & -1 & 0 & 0 & 200 \\ 0 & 0 & -1 & 4 & -1 & 0 & 200 \\ 0 & 0 & 0 & -1 & 4 & -1 & 200 \\ 0 & 0 & 0 & 0 & -1 & 4 & 100 \end{bmatrix}.$$

(a) Write an algorithm for solving a symmetric tridiagonal system that takes advantage of such compacting.

"a1", "a2", "a3" store the subdiagonal, diagonal, superdiagonal elements in the system of the tridiagonal matrix in compact storage form.

The algorithm consists a forward elimination and a back substitution.

In forward elimination step, update the elements of the tridiagonal matrix by subtracting the subdiagonal elements and the right-hand side vector in place. After the step, the tridiagonal matrix has been reduced to an upper triangular matrix.

Update the elements of tridiagonal matrix by subtracting proper amount of superdiagonal elements in the backward elimination step. After the step, we obtain the solution vector.

```
a1 = -1*ones(n-1, 1); % subdiagonal
a2 = 4*ones(n, 1); % diagonal
a3 = -1*ones(n-1, 1); % superdiagonal
b = [100; 200*ones(n-2, 1); 100];

% (a) Algorithm for solving a symmetric tridiagonal system
% forward substitution
for k = 2:n
    m = a1(k-1) / a2(k-1);
    a2(k) = a2(k) - m*a3(k-1);
    b(k) = b(k) - m*b(k-1);
end

x = zeros(n, 1);
x(n) = b(n) / a2(n);
% backward substitution
for k = n-1:-1:1
    x(k) = (b(k) - a3(k)*x(k+1)) / a2(k);
end
```

(b) Use the algorithm from part (a) to solve the system in Exercise 34.

```
>> Q2_110550062
Solution:
  46.341463414634148
  85.365853658536579
  95.121951219512198
  95.121951219512184
  85.365853658536594
  46.341463414634148
```

(c) How many arithmetic operations are needed with this algorithm for a system of n equations?

There are 7n-6 arithmetic operations in the algorithm.
For the operational count, in the reduction phase, there are n-1 rows for which there are two multiplies and two subtracts, which yields a total of 4(n-1) operations.
In the back substitution phase, there is one division in row *n*. For rows n-1 to row 1, there is on multiplication, subtraction, and division, so there are 3(n-1)+1 operations for this phase.

3. (25%) Solve this system of equations, starting with the initial vector of [0, 0, 0]:

$$4.63\,x_1 - 1.21\,x_2 + 3.22\,x_3 = 2.22,$$
$$-3.07\,x_1 + 5.48\,x_2 + 2.11\,x_3 = -3.17,$$
$$1.26\,x_1 + 3.11\,x_2 + 4.57\,x_3 = 5.11.$$

Results with max iteration=5000 and tolerance=1e-7:

```
>> Q3_110550062
Jacobi method: -8.98932     -9.48447       10.051
Gauss-Seidel method: -8.98932     -9.48448       10.051
```

(a) Solve using the Jacobi method.

$$x_{new} = D^{-1}(b - (L + U)x)$$

```
% Part (a): Solve using the Jacobi method
D = diag(diag(A));
L = tril(A,-1);
U = triu(A,1);
%inv_D = inv(D);

% Set the initial guess vector
x = [0; 0; 0];

% Iterate until convergence or max number of iterations
max_iter = 5000;
tol = 1e-7;
for iter=1:max_iter
    x_new = D \ (b - L * x - U * x); % D\ same as inv(D)
    if norm(x_new - x) < tol
        break;
    end
    x = x_new;
end
```

(b) Solve using the Gauss-Seidel method.

$$x_{new} = (D + L)^{-1}(b - Ux)$$

```
% Part (b): Solve using the Gauss-Seidel method
D = diag(diag(A));
L = tril(A,-1);
U = triu(A,1);
DL = D + L;
%inv_DL = inv(DL);

% Set the initial guess vector
x = [0; 0; 0];

% Iterate until convergence or max number of iterations
max_iter = 5000;
tol = 1e-7;
for iter=1:max_iter
    x_new = DL \ (b - U * x_new);
    if norm(x_new - x) < tol
        break;
    end
    x = x_new;
end
```

4. (10%) Find a good value of the overrelaxation factor that speeds the convergence of 3. (b) mostly by trying different values.

The result:

```
>> Q4_110550062
Optimal overrelaxation factor: 1.435000
Converges in 29 iterations
```

The code:

Since the overrelaxation factor are commonly between 1 and 2, I run through 1 to 2 by adding 0.001 each iteration with tolerance 1e-7. Choose the factor with minimum iteration to converge as the optimal overrelaxation factor.

```
tol = 1e-7;  % convergence tolerance
max_iter = 5000;  % maximum number of iterations
w_vals = 1:0.001:2;
conv_iters = zeros(size(w_vals));

for j = 1:length(w_vals)
    w = w_vals(j);
    x_new = [0; 0; 0];

    for iter = 1:max_iter
        x = x_new;
        for i = 1:length(b)
            x_new(i) = x(i) + w*(b(i) - A(i,1:i-1)*x_new(1:i-1) - A(i,i:end)*x(i:end))/A(i,i);
        end
        if (norm(x_new - x) < tol)  % check for convergence
            conv_iters(j) = iter;
            break;
        end
        if (norm(x_new - x) >= tol && iter==max_iter)  % check for convergence
            conv_iters(j) = inf;
            break;
        end
    end
end
conv_iters(conv_iters==0)=inf;
% disp(conv_iters);
[min_iters, min_idx] = min(conv_iters);  % find minimum number of iterations and index of optimal w
optimal_w = w_vals(min_idx);
```

交大資工 110550062
朱致伶

5. (15%) Compute the condition number of the following matrices and classify each of them as well-conditioned or ill-conditioned.

(a) $\begin{bmatrix} 10^{10} & 0 \\ 0 & 10^{-10} \end{bmatrix}$

(b) $\begin{bmatrix} 10^{10} & 0 \\ 0 & 10^{10} \end{bmatrix}$

(c) $\begin{bmatrix} 10^{-10} & 0 \\ 0 & 10^{-10} \end{bmatrix}$

(d) $\begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}$

$$\boldsymbol{cond}[\boldsymbol{A}] = ||\boldsymbol{A}||_\infty \left|\left|\boldsymbol{A^{-1}}\right|\right|_\infty = \boldsymbol{normofA} * \boldsymbol{normofA^{-1}}$$

$$\boldsymbol{normofA} = \boldsymbol{Maximum\ sum\ of\ absolute\ value\ of\ a\ row}$$

To determine whether the system is well-conditioned or ill-conditioned, we calculate the condition number of matrix.
If cond[A] ≈ 1.0, then the system is well-conditioned.
If cond[A] > 1.0, then the system is ill-conditioned.

(a) $A^{-1} = [\ 10^{-10}, 0; 0, 10^{10}\ ]$
$normofA = 10^{10}, \quad normofA^{-1} = 10^{10}$
$cond[A] = 10^{10} * 10^{10} = 10^{20} \gg 1$
It is ill-conditioned.

(b) $A^{-1} = [\ 10^{-10}, 0; 0, 10^{-10}\ ]$
$normofA = 10^{10}, \quad normofA^{-1} = 10^{-10}$
$cond[A] = 10^{10} * 10^{-10} = 1$
It is well-conditioned.

(c) $A^{-1} = [\ 10^{10}, 0; 0, 10^{10}\ ]$
$normofA = 10^{-10}, \quad normofA^{-1} = 10^{10}$
$cond[A] = 10^{-10} * 10^{10} = 1$
It is well-conditioned.

(d) $A^{-1}\ doesn't\ exist, but\ we\ know\ normofA^{-1} = \infty$
$normofA = 6$
$cond[A] = \infty \gg 1$
It is ill-conditioned.

6. (15%) (Bonus) Derive an algorithm and write a program to solve a linear system Ax=b, where A is an *N*x*N* band matrix with bandwidth *W*. You can generalize the algorithm you develop in 2. to handle a band matrix. (Please submit your code to E3)

Instead of storing the whole matrix A to compute the resolution of the system, storing only the compacted form of matrix A similar as the solution of question2 is more efficient. Since there are a lot of 0's in the matrix A. it is unnecessary to compute over 0's. I store the compacted form of the matrix and rebuild as a sparse matrix with spdiags() function.

After rebuilding the sparse matrix of A, I create a banded_matrix_solver() function using LU to compute the right-hand side value. To be noticed, the range of the for loop has been modified with the bandwidth W. Finally, we get the solution.

Outputs are the sparse matrix A and the solution of question 2 using banded_matrix_solver.

```
% Define the bandwidth and matrix size
W = 1;
N = 6;
% Define the vector b
b = [100; 200; 200; 200; 200; 100];
% Create a banded matrix A
diagonal = 4*ones(N,1);
upper_diagonal = -1*ones(N-1,1);
upper_diagonal = [upper_diagonal; 0];
lower_diagonal = -1*ones(N-1,1);
lower_diagonal = [0; lower_diagonal];
A = spdiags([upper_diagonal diagonal lower_diagonal], -W:W, N, N);
disp(A);
% Solve the linear system using the banded matrix solver
x = banded_matrix_solver(A, b, W);
disp(x);

function x = banded_matrix_solver(A, b, W)
% Get the size of the matrix A and Set up the output vector x
N = size(A, 1);
x = zeros(N, 1);
% Factor the matrix A into the product of two triangular matrices L and U,
for k = 1:N-1
    for i = k+1:min(k+W,N)
        A(i,k) = A(i,k) / A(k,k);
        for j = k+1:min(k+W,N)
            A(i,j) = A(i,j) - A(i,k)*A(k,j);
        end
    end
end
% Solve the lower triangular system Lx = b using forward substitution
for i = 1:N
    x(i) = b(i);
    for j = max(1,i-W):i-1
        x(i) = x(i) - A(i,j)*x(j);
    end
end
% Solve the upper triangular system Ux = y using backward substitution
for i = N:-1:1
    for j = i+1:min(i+W,N)
        x(i) = x(i) - A(i,j)*x(j);
    end
    x(i) = x(i) / A(i,i);
end
```

```
>> bonus_110550062
   (1,1)        4
   (2,1)       -1
   (1,2)       -1
   (2,2)        4
   (3,2)       -1
   (2,3)       -1
   (3,3)        4
   (4,3)       -1
   (3,4)       -1
   (4,4)        4
   (5,4)       -1
   (4,5)       -1
   (5,5)        4
   (6,5)       -1
   (6,6)        4

  46.282051282051285
  85.128205128205124
  94.230769230769226
  91.794871794871781
  72.948717948717956
  43.237179487179489
```