CMPS251 - Assignment 2

Kinan Dak Al Bab, 201205052

September 19, 2014

Note: MATLAB R2010b (the version in the labs) was used to solve this assignment.

1 Sine computation

1.1 Question

Write a matlab function $mysin2(x, d)$ that computes the sine of a number to d digits of accuracy, i.e., to an error smaller than 0.5×10^{-d}

1.2 Code

```
% Sine computation with a certain error %

function result = mysin2(x, d)
% calculates sin(x) up to d digits of accuracy %

max_value = 1; % sin or any of its derivatives (they alternate ...
    between sin and cos) range between -1 and 1
n = 1; % start with n equals one (linear taylor)

upper_error = (max_value / factorial(n)) * (x^n); % Upper bound ...
    for absolute value of error
while upper_error > 0.5 * 10^(-d)
    n = n + 1;
    upper_error = (max_value / factorial(n)) * (x^n);
end

result = mysin(x, n);
end
```

1.3 Output

```
>> sin(pi/4)

ans =

   0.707106781186547

>> mysin(pi/4, 1)

ans =

   0.785398163397448

>> mysin(pi/4, 2)

ans =

   0.704652651209168

>> mysin(pi/4, 3)

ans =

   0.707143045779360

>> mysin(pi/4, 13)

ans =

   0.707106781186547
```

1.4 Comment

Compare the answers to that of the built-in sin function. Increasing d yeild in increasing the precision , for d = 1, 2 we actually get only the first and second digits after the decimal point correctly, with d = 3 we happen to get one extra digit to be correct (which is fine because we can trust the first 3 digits which is the point), as we get closer to machine percision we get an answer equal to that of the built in sin (d = 13).

2 Solution cost

2.1 Question

Exercise 3 on page 134
Let A and T be two nonsingular, $n \times n$ real matrices. Furthermore, suppose we are given two matrices L and U such that L is unit lower triangular, U is upper triangular, and

$$TA = LU.$$

Write an algorithm that will solve the problem

$$Ax = b$$

for any given vector b in $O(n^2)$ complexity. explain why the algorithm requires only $O(n^2)$ flops, and specify your algorithm in details.

2.2 Solution Guideline

We want to solve:

$$Ax = b$$

Multiply both side by T:

$$TAx = Tb$$

But we already have:

$$TA = LU$$

We get:

$$LUx = Tb$$

Take:

$$y = Ux$$

We get

$$Ly = Tb$$

$$Ux = y$$

We solve the first equation by a forward pass, and the second one by a backwards pass.

2.3 Algorithm (in pseudo code)

Forward Pass:

for i from 1 to n
    $sum := 0$
    for j from 1 to $i-1$
        $sum := sum + (L_{i,j} \times y_j)$
    end_for
    $y_i := \frac{\bar{b}_i - sum}{L_{i,i}}$
end_for

Notice that $\bar{b}$ in the above algorithm is actually the matrix b in the solution guideline after applying the permutations in T to it.

Backwards Pass:

for i from n to 1
    $sum := 0$
    for j from $i+1$ to n
        $sum := sum + (U_{i,j} \times x_j)$
    end_for
    $x_i := \frac{y_i - sum}{U_{i,i}}$
end_for

2.4 Solution Cost

- Forward pass and Backwards pass are nearly identical, the i counter moves across the same range (in opposite direction), and the j counter moves across similar ranges; in forwards pass it begins with an empty range at $i = 1$ then ends with a range of size $n-1$ at $i = n-1$, in backwards pass it begins with an empty range (from $i+1 = n+1$ to n then ends with a range of size $n-1$ (from $i+1 = 2$ to n). So Forward pass and Backwards pass have the same cost.
- Let us analyze one of them, let it be Forward Pass:
  - In the inner most loop we have an addition and a multiplication (we can say 2 flops). The inner most loop executes zero times at the begining and $n-1$ times at the end (so nearly $\frac{n}{2}$ times on average), so we can say that the inner loop costs on average $2 \times \frac{n}{2}$ flops.
  - The outer loop contains the inner loop ($2 \times \frac{n}{2}$ flops on average), one subtraction and one divison, so we can say it contains ($2+2 \times \frac{n}{2}$ flops on average). it gets executed n times (i moves from 1 to n inclusive).
  - So in total, we have around $(2+2\times\frac{n}{2})\times n = 2\times n+n^2$ flops. Which is something in the order of $O(n^2)$. and the same applies to Backwards pass.

- The Last thing is to analyze how much time it requires to permute b according to T. It turns out that this too is in order of $O(n^2)$. We have to swap a maximum of n rows, each row containing n elements, so we have data movements in order of $O(n^2)$.
- Therefore the algorithm as a whole is in order of $O(n^2)$ flops.

3 Forward Pass

3.1 Question

Write a matlab script that solves a system of equations $Lx = b$ with a lower triangular coefficent matrix.

3.2 Code

```
% generate a random coefficient matrix and right hand side %
n = 10;
L = tril(rand(n, n))
b = rand(n, 1)

% my solution %
x = zeros(n, 1); % create a vector for unknowns %
for i = 1:n
    sum = 0;

    for j = 1:i-1
        sum = sum + L(i, j) * x(j);
    end

    x(i) = (b(i) - sum) / L(i, i);
end

% check that the solution is correct %
norm(L * x - b)
```

3.3 Output

```
ans =

    1.110223024625157e-016
```

3.4 Comment

The norm is very close to zero (machine precision), which indicates that x is getting a valid solution. the norm is not exactly zero due to possible accumulation of precision errors.

4 Backwards Pass

4.1 Question

Write a matlab script that solves a system of equations $Lx = b$ with a lower triangular coefficent matrix.

4.2 Code

```
% generate a random coefficient matrix and right hand side %
n = 10;
U = triu(rand(n, n));
b = rand(n, 1);

% my solution %
x = zeros(n, 1); % create a vector for unknowns %
for i = n:-1:1
    sum = 0;

    for j = i+1:n
        sum = sum + U(i, j) * x(j);
    end

    x(i) = (b(i) - sum) / U(i, i);
end

% check that the solution is correct %
norm(U * x - b)
```

4.3 Output

```
ans =

    2.626240719515172e-015
```

4.4 Comment

The norm is very close to zero (machine precision), which indicates that x is getting a valid solution. the norm is not exactly zero due to possible accumulation of precision errors.

5 LU Factorization

5.1 Question

Write a matlab function to compute the L and U factors of a matrix without pivoting.

$$function[L, U] = myLU(A)$$

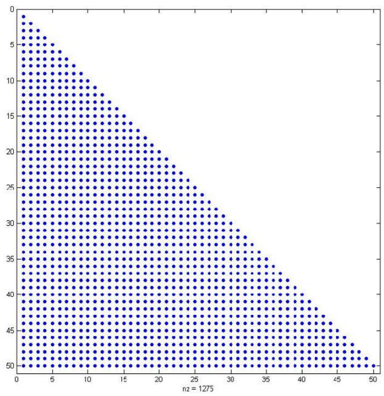
What is the cost of the factorization as a function of matrix size? (Write the leading term only).

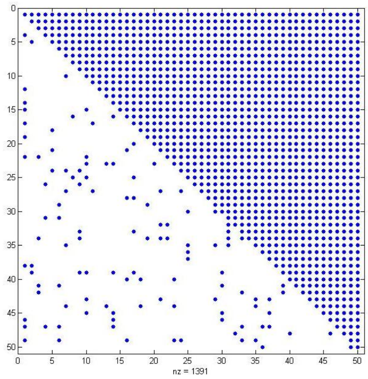
5.2 Code

```
function [L, U] = myLU(A)
n = size(A);
n = n(1);

U = A; % save a copy of A and use to generate U
L = eye(n, n); % identity matrix

for i = 1:n-1
    for j = i+1:n
        multiplier = U(j, i) / U(i, i); % get multiplier
        U(j, i:n) = U(j, i:n) - multiplier * U(i, i:n); % Hidden triple nested loop
        L(j, i) = multiplier;
    end
end
end


% Script for Testing myLU %
% Testing myLU with a random matrix %
A = rand(50, 50); % Generate random matrix of size 50x50
[L U] = myLU(A); % get LU factorization

norm(A - (L * U)) % L * U should be equal to x (with some precision errors).

spy(U)
spy(L)
```

5.3 Output

```
ans =

    4.159036827023391e-014
```





5.4 Comment

- As expected the norm is very close to zero, so $L \times U$ is nearly equal to A.
- By using the spy command, we can see that L and U are lower and upper triangular matrices (U has few entries bellow the diagnoal but they are due to precision error in the multiplier).
- The cost of this algorithm is in the order of $O(\frac{2}{3} \times n^3)$, there are three nested loops (inner most is hidden by smart usage of Matlab). However, these loops range is in most cases less than n, which is why we get the $\frac{2}{3}$ factor, we save up some flops by not doing calculations on the zeros over and over again.

6 Cost of Gaussian elimination

6.1 Question

Consider using a 1 GFLOP/s machine for performing Gaussian elimination to solve the linear system $Ax = b$.

- How long will it take to factor a square matrix A of size $n = 2500$?
- How long will it take to perform forward and backward passes after factorization?
- How long will it take to solve 100 systems of equations with the same coecient matrix A but different right hand sides b.

6.2 Solution

- Factorization have flops in around order of $O(\frac{2}{3} \times n^3)$, so for $n = 2500$ we get around
$$2500^3 \times \frac{2}{3} \approx 10410000000 = 10.41 \times 10^9$$
$Time = \frac{10.41 \times 10^9}{10^9} = 10.41$ seconds

- From Question 2 we get that Both Forward and Backwards passes have around $n^2 + 2 \times n$ flops, so for $n = 2500$ we get $6255000 = 6.255 \times 10^6$ flops.
  $Time = \frac{6.255 \times 10^6}{10^9} = 6.255 \times 10^{-3}$ seconds (around 6 milliseconds) for each pass.

- Solving 100 system of equations:
  - Factorization is done once, costing around 10.5 seconds as we saw earlier.
  - Each time we need to do a forward and a backwards pass to solve one system, costing around 12 milliseconds for each system. so in total 1200 milliseconds (around 1.2 seconds).
  - So in total we need around 11.7 seconds. most of the time being spent on factorization.
  - We might need to take into consideration the permuting of the new b matrix each time according to P, but that is in order of n^2, so it would take time close to backwards or forward pass.

7 Cost of Gaussian elimination

7.1 Pivoting

Consider the 3×3 system of equations in Example 5.8 on page 106. Following the discussion, solve this system "by hand" (using 5 signicant digits only) with and without partial pivoting. Comment

7.2 Solution

$$x_1 + x_2 + x_3 = 1$$

$$x_1 + 1.0001x_2 + 2x_3 = 1$$

$$x_1 + 2x_2 + 2x_3 = 1$$

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1.0001 & 2 \\ 1 & 2 & 2 \end{pmatrix}$$

$$b = \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix}$$

Without Pivoting:

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} U = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1.0001 & 2 \\ 1 & 2 & 2 \end{pmatrix}$$

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} U = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0.0001 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 10000 & 1 \end{pmatrix} U = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0.0001 & 1 \\ 0 & 0 & -9999 \end{pmatrix}$$

$$L \times U \times x = b$$

$$L \times y = b$$

$$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 10000 & 1 \end{pmatrix} \times \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix}$$

$$y_1 = 1$$

$$y_2 = 2 - y_1 = 1$$

$$y_3 = 1 - 10000 \times y_2 - y_1 = -10000$$

$$\begin{pmatrix} 1 & 1 & 1 \\ 0 & 0.0001 & 1 \\ 0 & 0 & -9999 \end{pmatrix} \times \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ -10000 \end{pmatrix}$$

$$x_3 = \frac{-10000}{-9999} = 1.0001$$

$$x_2 = \frac{1 - x_3}{0.0001} = -1$$

$$x_1 = 1 - x_3 - x_2 \times x_2 = 0.9999$$

With Partial Pivoting:

$$P = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} L = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} U = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1.0001 & 2 \\ 1 & 2 & 2 \end{pmatrix}$$

$$P = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} L = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} U = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0.0001 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

$$P = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} L = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} U = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0.0001 & 1 \end{pmatrix}$$

$$P = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} L = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0.0001 & 1 \end{pmatrix} U = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 0.9999 \end{pmatrix}$$

$$L \times U \times x = b$$
$$L \times y = P \times b$$
$$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0.0001 & 1 \end{pmatrix} \times \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \times \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 2 \end{pmatrix}$$

$$y_1 = 1$$
$$y_2 = 1 - y_1 = 0$$
$$y_3 = 2 - 0.0001 \times y_2 - y_1 = 1$$

$$\begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 0.9999 \end{pmatrix} \times \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$$

$$x_3 = \frac{1}{0.9999} = 1.0001$$
$$x_2 = 0 - x_3 = -1.0001$$
$$x_1 = 1 - x_3 - x_2 = 1$$

7.3 Comment

Without Pivoting, we got a big number in L (10000), and we got a negative number in U (-9999), that yeilded to more complex calculations later on, and might in more exterme cases cause precision errors and under/over flows.

With pivoting we are able to eliminate this problem.

Notice that the solutions to the linear system is nearly the same with or without pivoting, there is a small difference in some answers (0.0001) which is in the order of our precision.