Numerical Algorithm

Assignment 1

Mohammad Mahdi Fallah

Paper and Pencil:

1. Mantissa Length: 1 bit sign, 15 bit exponent => 128-15-1 = **112 bit**

Machine Epsilon : accuracy of floating point systems.

machine epsilon = 2^-112 = 1.93 * 10^**-34**

Significant Decimal Digits: the Equation above shows that this 128 bit word has **34** significant decimal digits

	Mantissa length	Machine epsilon
Quadruple precision	112	1.93 * 10^**-34**
Double precision	52	2.2 * 10^**-16**
Single precision	23	1.2 * 10^**-7**

This shows that quadruple precision is more than double time as accurate as double precision which is more than double time as accurate as single precision.

2. a)

$$\left.\begin{aligned} fl(a \text{ op } b) &= (a \text{ op } b)(1+\delta) \\ fl(b \text{ op } a) &= (b \text{ op } a)(1+\delta) \end{aligned}\right\} \Rightarrow fl(a \text{ op } b) = fl(b \text{ op } a)$$

b)

$$\left.\begin{aligned} fl(a+a) &= (\overbrace{a+a}^{2*a})(1+\delta) \\ fl(2*a) &= (2*a)(1+\delta) \end{aligned}\right\} \Rightarrow fl(a+a) = fl(2*a)$$

c)

$$\begin{aligned} fl((a+b)+c) &= ((a+b)(1+\delta)+c)(1+\delta) \\ &= ((a+b+c)+(a+b)\delta)(1+\delta) \\ &= (a+b+c)\left[1+\underbrace{\frac{a+b}{a+b+c}}_{\mathrm{I}}\delta(1+\delta)+\delta\right] \end{aligned}$$

$$\begin{aligned} fl(a+(b+c) &= (a+(b+c)(1+\delta))(1+\delta) \\ &= ((a+b+c)+(b+c)\delta)(1+\delta) \\ &= (a+b+c)\left[1+\underbrace{\frac{b+c}{a+b+c}}_{\mathrm{II}}\delta(1+\delta)+\delta\right] \end{aligned}$$

$$\Longrightarrow \mathrm{I} \neq \mathrm{II} \Longrightarrow fl((a+b)+c) \neq fl(a+(b+c))$$

3. $s_n = \sum_{i=1}^{n} x_i$

$\rightarrow s_1 = x_1 \qquad s_2 = x_1 + x_2 \qquad s_3 = (x_1 + x_2) + x_3$

$\rightarrow s_n = s_{n-1} + x_n$

$$fl(x \text{ op } y) = (x \text{ op } y)(1+\delta), \quad |\delta| \leq \varepsilon_m$$

$\rightarrow \hat{s}_2 = (x_1 + x_2)(1+\delta)$

$\hat{s}_3 = \hat{s}_2 + x_3 = ((x_1 + x_2)(1+\delta) + x_3)(1+\delta)$

$= (x_1 + x_2)(1+\delta)^2 + x_3(1+\delta)$

$\hat{s}_4 = \hat{s}_3 + x_4 = (x_1 + x_2)(1+\delta)^3 + x_3(1+\delta)^2 + x_4(1+\delta)$

$\rightarrow \hat{s}_n = fl(\hat{s}_{n-1} + x_n) = (\hat{s}_{n-1} + x_n)(1+\delta)$

$= (x_1 + x_2)(1+\delta)^{n-1} + \cdots + x_n(1+\delta)$

from Lecture: 02-ErrorAnalysis, Page 10

if $|\delta| \leq \varepsilon_m$ and $\rho_i = \pm 1$ and $n\varepsilon_m < 1$

$$\prod_{i=1}^{n} (1+\delta)^{\rho_i} = 1 + \theta_n$$

where $|\theta_n| \leq \dfrac{n\varepsilon_m}{1 - n\varepsilon_m} =: \gamma_n$

$\rightarrow \hat{s}_n = (x_1 + x_2)(1 + \theta_{n-1}) + \sum_{i=3}^{n} x_i (1 + \theta_{n-i+1})$

Forward Error $= |\hat{s}_n - s_n|$

$$= \left| (x_1 + x_2)(1+\theta_{n-1}) + \sum_{i=3}^{n} x_i (1+\theta_{n-i+1}) - \sum_{i=1}^{n} x_i \right|$$

$$= \left| x_1 + \cancel{x_2} + (x_1+x_2)(\theta_{n-1}) + \cancel{\sum_{i=3}^{n} x_i} + \sum_{i=3}^{n} x_i \theta_{n-i+1} - \cancel{\sum_{i=1}^{n} x_i} \right|$$

$$= \left| (x_1 + x_2)(\theta_{n-1}) + \sum_{i=3}^{n} x_i \theta_{n-i+1} \right|$$

$$\leq (|x_1| + |x_2|)\gamma_{n-1} + \sum_{i=3}^{n} |x_i| \gamma_{n-i+1}$$ $\rightarrow$ Absolute forward Error Bound

Backward Error: $|\hat{x} - x|$, $\hat{x}_i = x_i(1+\theta_{n-i+1})$

$$\rightarrow |\hat{x}_i - x_i| = |x_i(1+\theta_{n-i+1}) - x_i|$$

$$= |x_i + x\theta_{n-i+1} - x_i|$$

$$= |x_i \theta_{n-i+1}|$$

$$\leq |x_i| \gamma_{n-i+1}$$ $\rightarrow$ Backward Error Bound

Programming:

For this assignment I have used octave v6.3 . The experiments for three parts are implemented in **partI.m**, **partII.m** and **partII.m** as functions and I use **assignment.m** only to call these scripts.
For matrix generation I used randi() function, ad I used det() for checking singularity.

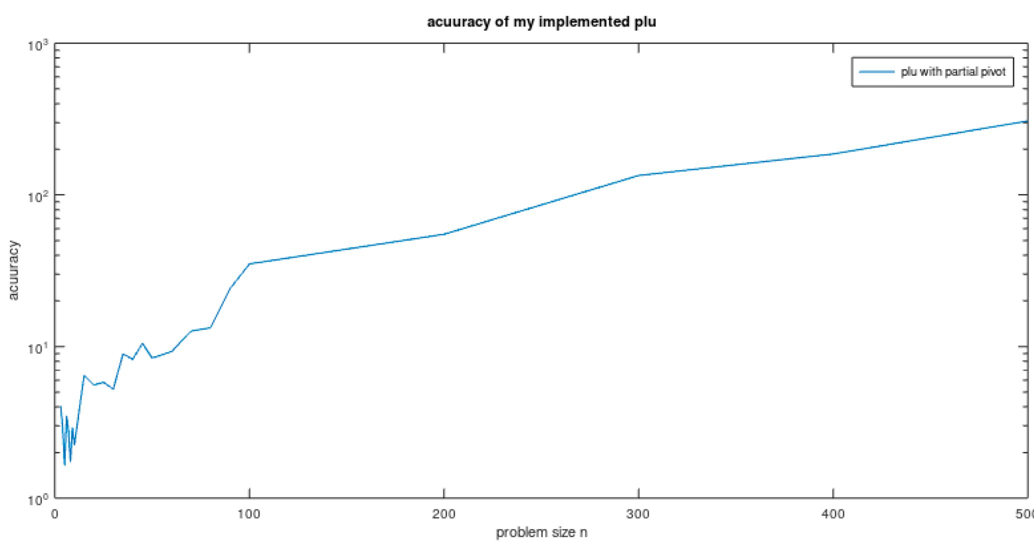
```
A = randi([1, 10], [n,n]);
% to prevent singular matix
while det(A) == 0
  A = randi([1, 10], [n,n]);
end
```

partI:

for this part I first implemented LU decomposition with and without partial pivoting in **plu.m** and **plu_nopivot.m** . plu(A, n) returns LU (which contains Lower and Upper Triangular matrix) and P permutation matrix. Then in **split.m** we retrieve the lower and upper triangular matrices from LU as follows

```
function [L, U] = split(LU, n)
  U = triu(LU);
  L = eye(n) + tril(LU,-1);
endfunction
```

Then by using **accuracy.m** I compare P.' * L * U with the original matrix A in intervals [2:10 15:5:50 60:10:100 200:100:500] and I plot the results with logarithmic scale on Y-Axis using **semilogy()**:



partII:

in this part we start by creating **solveL.m** and **solveU.m** which implement forward and backward substitution.

For generating random matrix I used the same implementation as partI, then we use plu() for LU decomposition, and we retrieve L and U using split().

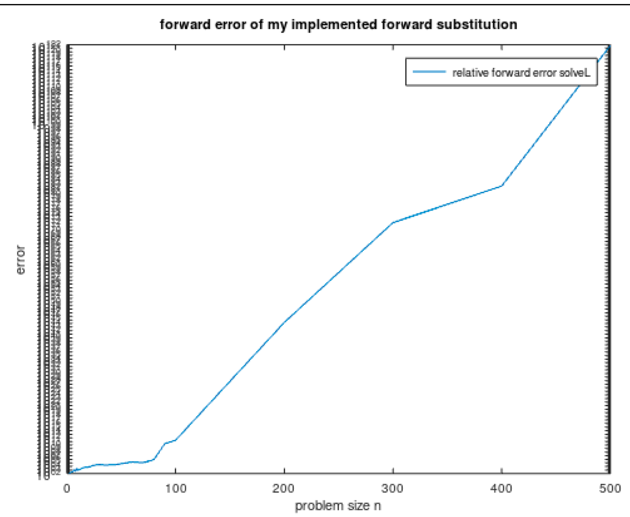
As the x must be a vector of only ones I use the definition below to assign b:

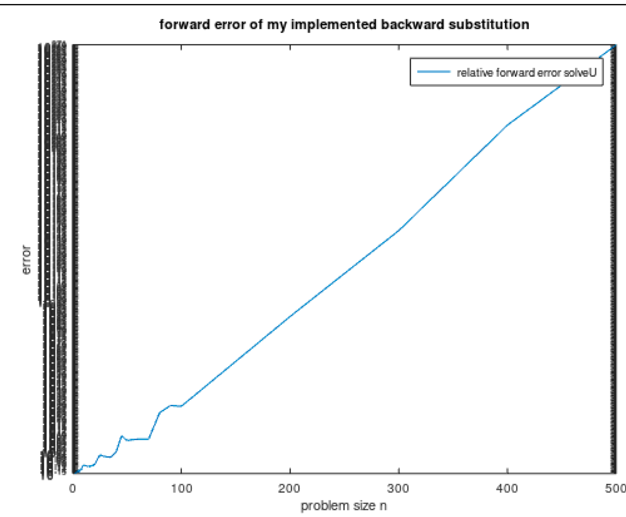
$$LUx = b$$

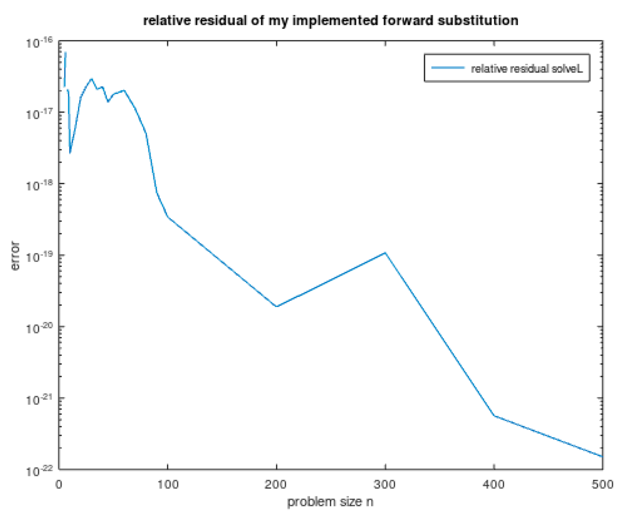
This means b can be assigned as follow:

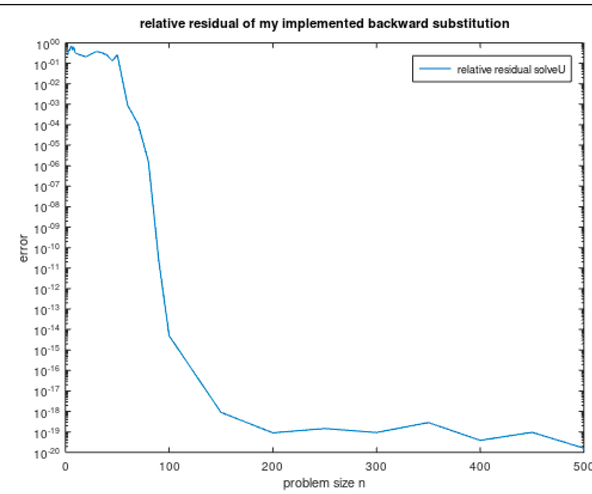
```
[LU, P] = plu(A,n);
[L, U] = split(LU, n);
b = L * U * ones(n,1);
```

Then we solve Ly = b with solveL() and then Ux = y with solveU() and we plot the results:









partIII:

in this part first we assign matrix S and H as follow:

```
% Matrix S
S = randi([-1, 1], [n,n]);
while det(S) == 0
  S = randi([-1, 1], [n,n]);
end
```

```
% Matrix H
H = zeros(n);
for i = 1:n
  for j = 1:n
    H(i,j) = 1/(i+j-1);
  endfor
endfor
```

Next by using **linsolve.m** which uses solveL and solveU to solve the system we solve S and H so that true x is a vector of ones as it was explained in partI.