3. (a)

Here is the code:

```
% Tak a square matrix as an input
% Calculate L, lower trianglar matrix
% Calculate U, upper triangular matrix
function [L,U]=luFactorNoPivoting(A)
n=length(A);
U=A;
L=eye(n,n);
for j=1:n-1
    for i= j+1:n
        L(i,j)=U(i,j)/U(j,j);
        U(i,j:n)=U(i,j:n)-L(i,j)*U(j,j:n);
    end
end
return
```

Here is the output:

```
>> A = [2 1 1 0;4 3 3 1;8 7 9 5;6 7 9 8]

A =

     2     1     1     0
     4     3     3     1
     8     7     9     5
     6     7     9     8

>> [L,U]=luFactorNoPivoting(A)

L =

     1     0     0     0
     2     1     0     0
     4     3     1     0
     3     4     1     1


U =

     2     1     1     0
     0     1     1     1
     0     0     2     2
     0     0     0     2
```

(b) Here is the code:

```
% Take a matrix as an input,  b
% Take a lower trianglar matrix as an input, L
% Take a upper trianglar matrix as an input, U
function x=luSolveNoPivoting(b, L, U)
n = length(L);
y = b;
for j = 2:n
    for i= 1:j-1
        y(j) = y(j) - y(i) * L(j,i);
    end
end
x = y;
x(n) = y(n) / U(n,n);
for j = n - 1:-1 : 1
    for i = j + 1:n
        x(j) = x(j) - x(i) * U(j, i);
    end
    x(j) = x(j) / U(j, j);
end
return
```

Here is the output:

```
>> b = [7, 23, 69, 79]'

b =

     7
    23
    69
    79

>> x = luSolveNoPivoting(b, L, U)

x =

     1
     2
     3
     4
```

4.

(a)

Here is the code, to calculate different n values, we simply change it by typing into command window.

```
>> % form a matrix of size n
n = 10;
A=zeros(n,n);
for i=1:n
    for j=1:n
        A(i,j)=abs(i-j)+1;
    end
end
% Factor it using previous code
[L,U] = luFactorNoPivoting(A);
x = ones(n, 1);
b = A * x;
xa = U \ (L\b);
% Solve for RFE, RBE, EMF and kappa
RFE = norm(x - xa, inf)/norm(x, inf);
RBE = norm(b - A * xa, inf) / norm(b, inf);
EMF = RFE / RBE;
kappa = norm(A, inf) * norm(inv(A), inf);
fprintf("n=%5d : RFE=%8.2e RBE=%8.2e EMF=%8.2e kappa(A)=%8.2e\n",n,RFE,RBE,EMF,kappa);
```

Here is the output:

n=   10 : RFE=2.95e-14 RBE=6.46e-16 EMF=4.57e+01 kappa(A)=1.10e+02

n= 100 : RFE=5.40e-11 RBE=4.14e-14 EMF=1.30e+03 kappa(A)=1.01e+04

n= 1000 : RFE=5.21e-08 RBE=4.79e-13 EMF=1.09e+05 kappa(A)=1.00e+06

(b)

From the output result, we can see that our errors are all relatively small for all n value. In this case, I do think that it appears to be a stable algorithm to use.

(c)

We can see that, as n increases, kappa increases accordingly; as n gets bigger, our kappa approaches to n^2.

(d)

I do think it is a reasonable estimate for EMF because in all three cases, it is all bigger than EMF value.

(e)

$If\ our\ result\ is\ k_{\infty}(A) = 10^{n}\ and\ k_{\infty}(A)\ lies\ between\ RFE\ and\ RBE,$

$then\ we\ can\ use\ this\ n\ to\ approximate\ the\ number\ of\ significant\ digits.$

$If\ we\ have\ a\ n\ satisfy\ above\ condition,$

$we\ can\ use\ (preicison - n)\ to\ approximate\ it.$

$Double\ precision: 16 - n.$

$Single\ precision: 8 - n.$