PSEUDOCODES

Incremental Searches:

```
Input: function, delta, # of iterations, initial point
Initialize variables:
F(x) =function
iter=0  # of iterations
x0= initial point
xi=xo+delta
counter=0
while counter different to 30:

        a=f(x0)
        b=f(xi)
        if a*b<0:

            print (there is a root in the interval a,b)

        xo=xi
   xi=x0+delta

        cont = cont +1
```

Bisection:

```
Input: function, interval (a,b), number of iterations
Initialize variables:
f(x)=function
x1=a # interval
x2=b #interval
xm=(x1+x2)/2
```

```
iter=1 # number of iteration
f(x1)=function evaluated in x1
f(xm)= function evaluated en xm
cont=1
while cont different of iter::
        if f(x1)*f(xm)<0:
                x1=x1
                 x2=xm
        if f(x1)*f(xm)>0:
                x1=xm
                x2=x2
        if f(x1)=o:
                print: in x1 there is one root.

        if f(xi)=o:
                print: in x1 there is one root.
```

Newton - Raphson:

```
input equal:# max iterations,function,initial approach, fuction derivate,error
 initialize variables and to declarate accountant
g equal value resulting from the formula
g1 equal value found in return to g1
c equal accountant
e equal calculated error
while calculated error  greater equal error and c less equal # max iterations
      value resulting from the formula equal value found in return to g1
      equal value found in return to g1 equal new found value of g
      e equal calculated error
      c equal c plus one
finished code
```

print calculated error and value found in return to g1

Secant Method Pseudocode:

Input: # of iterations, interval [x0, x1], function f, error.

Initialize variables and stablish accountant

C=accountant

f0 equal function evaluated in x0

f1 equal function evaluated in x1

xa equal x1-((f1*(x1-x0))/(f1-f0))

e=calculated error

e equal absolute value (xa-x1)

while e bigger or equal than error and c minor or equal than iterations

x0=x1

x1=xa

f0= f evaluated in x0

f1= f evaluated in x1

xa= x1-((f1*(x1-x0))/(f1-f0))

e=absolute value (xa-x1)

c=c+1

finish while

show e

show xa

Multiple Roots Pseudocode:

input equal # max iterations, initial approach,function,first derivative of the function, second derivative of the function, error

initialize variables and to declare accountant

variable 1 equal function evaluated in initial approach

variable 2 equal first derivative of the function evaluated in initial approach variable 3 equal second derivative of the function evaluated in initial approach

g equal formula value

e equal error equal 1

go equal accountant equal cero

c equal one

while error greater equal error and c less equal to # max iterations

variable 1 equal function evaluated in formula value

variable 2 equal first derivative of the function evaluated in formula value

variable 3 equal second derivative of the function evaluated in formula value

go equal formula value

g equal second formula value

e equal calculated error

c equal c plus one

finished program

print calculated error and formula value

Fixed Point Pseudocode:

Input: function, initial aproximation, number of iterations, maximum absolut error wanted.

F(x) =function

iter=0 # of iterations

x0= initial point

error equal to maximum absolut error wanted

counter = 1

e equal absolut value of x1 minus x0

g() equal function evaluated in x0

f() equal function evaluated in g()

while e bigger or equal to error and counter minor or equal to number of iterations:

g(x) = g(f(x))

f(x) = f(g(x))

e equal to absolut value of g(x) minus f(x)

counter equal to counter plus 1

False Rule Pseudocode:

input equal:# max iterations,#min intervl,2nd # interval,functin,max error

initialize variables and to declare accountant

e equal to function

c equal accountant

while function greater equal error and c less equal to # max interval

variable 1 equal to function evaluated in # min interval

variable 2 equal to function evaluated in 2nd # interval

variable 3 equal to new number to evaluated

variable 4 equal function evaluated in variable 3

if variable 1 x variable 4 is less cero

min interval equal # min interval and

2nd # interval equal variable 3

if not

min interval equal variable 3 and

2 nd # interval equal 2nd # interval

finished program

print function and accountant plus one

SOR Pseudocode:

Inputs

A= squared Matrix A

b= Vector b

x0= Vector with initial values X0

```
Tol= Tolerance

Nmax=Numbers of iterations

w= values for approx.

        L=identity matrix size (m);

        U= identity matrix size (n);

        if n = m

                if determinant (A) ≠ 0

                        D=diagonal of the matrix A

                        L=D-triangular higher(A)

                        U=D-triangular lower(A)

                        T=[(D-w*L)]^(-1)*((1-w)*D+w*U)

                        C=w* [(D-L)]^(-1)*b

                        Lambda=eig(T)

                        Radio=max |eig(T)|

                        if Radio >= a 1

                                display (‘the radio espectral is more than 1, the method
                                diverges ’)

                        else

                                i=0

                                Error=Tol+1

                        while Error > Tol e i < Nmax

                                i=i+1

                                x=T*(x0)+C

                                Error= norm(x-x0);

                                display (i)

                                for j = 1 until m

                                        display (x(j))

                                end for

                                display (Error)

                                x0=x
```

```
        end while
        if Error < Tol
            display ('The program have a Error')
            display (error)
        else
            display ('the program have a problem
            with the iterations')
            display (error)
        end if
        end if
        else
        display ('The determinant of the
            matrix is 0, the system has infinite solutions ')
        end if
        display ('check your inputs the the
        matrix is not square')
        display('Radio spectral)
        display(Radio)
```

Vandermonde Pseudocode:

```
Input: x values as a line (x), y values as a column (y)
n= size of x
A= zeros size(n,n)
For i=1,2,...,n
    For j=1,2,...,n
        A(i,j)=x(i)^(n-j)
    End for
End for
b= solution between A and y
show A
```

```
show “the polynomial coefficients are=”
show b
pol= zeros size(1,n)
for i=1,2,...,n
        pol(1,i)=b(i,1)
end for
show “the interpolator polynomial is”
c=n-1
for i=n,n-1,n-2,...,2
        show “(X^c)(pol(1,i)”
        c=c-1
end for
show pol(1,1)
```

LU Simple Pseudocode:

```
Enter matrix A and vector b
(n,m)= size of A
C=augmented matrix

if n equal m so then
d equal diag(C)
for k=1 until n-1
for i=1 until m
if (d(i) equal 0)
show change of rows, zeros in the diagonal
get out
end

end
if A(k,k) different 0
```

for I equal (k+1) until n

m(i,k)=A(i,k)/A(k,k)

disp(m(i,k))

for j=k until n

A(i,j)= A(i,j) - m(i,k)*A(k,j)

%Matriz L

if i>j

L(i,j) equal m(i,k)

end

if I equal j

L(i,j)equal 1

end

if i<j

L(i,j)= 0

end

end

end

end

end

end

if A(1,1) different 0

show L and U

End

LU with partial pivoting

Input: Matrix A, vector b

if matrix determinant = 0 then

break

M <- A

L <- Identity matrix of the same size of A

P <- Identity matrix of the same size of A

U <- Zero-filled matrix of the same size of A

for i from 0 to n-1 do

in the given column find the row with the greatest number (in abs)

if current row is different to the row found then

switch current M row and found row

switch current P row and found row

if i > 0 then

switch current L row and found row

for j from i+1 to n do

if M[j,i] is not zero then

get the multiplier MULT = M(j,i)/M(i,i)

M[j,n] = M[j,i] - MULT * M[i,n]

Pb <- P*b

LPb <- [L Pb]

z = apply forward_substitution to LPb

Uz <- [U z]

solution = apply back_substitution to Uz

Doolittle Pseudocode:

INSERT: Matrix A, Vector b as a column.

n= size of A

U= zeros size (n,n)

```
L= diagonal of ones size (n,n)
For i=1,2,...,n
U(1,i)=A(1,i)
L(i,1)=A(i,1)/U(1,1)
End for
Show “Stage 1= “
Show “Matrix L= “
Show L
Show “Matrix U= “
Show U
sumL=0
sumU=0
sumU1=0
sumnn=0
for k=2,3,...,n
show “stage k= “
for j=k+1,k+2,...,n
sumU=0
for p=1,2,...k-1
sumU=sumU+(L(k,p)*U(p,k))
end for
U(k,k)=A(k,k)-sumU
sumU1=0
for p=1,2,...,k-1
sumU1=sumU1+(L(k,p)*U(p,j))
end for
U(k,j)=A(k,j)-sumU1
sumL=0
for p=1,2,...,k-1
sumL=sumL+(L(i,p)*U(p,k))
```

```
end for
L(j,k)=(1/U(k,k))*(A(j,k)-sumL)
End for
If k=n
Sumnn=0
For p=1,2,...,n-1
Sumnn=sumnn+L(n,p)*U(p,n)
End for
U(n,n)=A(n,n)-sumnn

End if
Show “Matrix L= “
Show L
Show “Matrix U= “
Show U
End for
z=zeros size (n,1)
sumZ=0
z(1,1)=b(1,1)/L(1,1)
for i=2,3,...,n
sumZ=0
for s=1,2,...,n
sumZ=sumZ+L(i,s)*z(s,1)
end for
z(i,1)=(b(i,1)-sumZ)/L(i,i)
end for
show “Vector z= “
show z
x=zeros size (n,1)
sumX=0
```

```
x(n,1)=z(n,1)/U(n,n)
for i=n,1,n-2,...,1
sumX=0
for s=n,n-1,...,1
sumX=sumX+U(i,s)*x(s,1)
end for
x(i,1)=(z(i,1)-sumX)/U(i,i)
end for
show "Vector x= "
show x
show "X values are= "
for i=1,2,...,n
show xi= x(i,1)
end for
```

Crout:

INSERT: Matrix A, Vector b as a column.

```
n= size of A
L= zeros size (n,n)
U= diagonal of ones size (n,n)

    For i=1,2,...,n
        L(i,1)=A(i,1)
    End for

SumL=0
SumU=0
For k=1,2,...,n
Show "Stage k= "
```

```
          For i=k,k+1,...,n
                    sumL=0
                    for p=1,2,...,k-1
                              sumL=sumL+(L(i,p)*U(p,k))
                    end for
                    L(i,k)=A(i,k)-sumL
                    End for
                              For j=k+1,k+2,...,n
                                        sumU=0
                              for p=1,2,...,k-1
                                        sumU=sumU+(L(k,p)*U(p,j))
                              end for
          end for

show “Matrix L= ”
show L
show =Matrix U= “
show U
end for

z=zeros size (n,1)
sumZ=0
z(1,1)=b(1,1)/L(1,1)

for i=2,3,...,n
sumZ=0
for s=1,2,...,n
sumZ=sumZ+L(i,s)*z(s,1)
end for
z(i,1)=(b(i,1)-sumZ)/L(i,i)
```

```
end for

show “Vector z= “
show z
x=zeros size (n,1)
sumX=0
x(n,1)=z(n,1)/U(n,n)

        for i=n,1,n-2,...,1
        sumX=0
                for s=n,n-1,...,1
                        sumX=sumX+U(i,s)*x(s,1)
                end for
        x(i,1)=(z(i,1)-sumX)/U(i,i)
        end for

show “Vector x= “
show x
show “X values are= “
for i=1,2,...,n
        show xi= x(i,1)
end for
```

Cholesky:

INSERT: Matrix A, Vector b as a column.

```
n= size of A
L= zeros size (n,n)
sumL1=0
```

```
sumL2=0
sumL3=0
L(1,1)= root(1,1)

        For j=2,3,...,n
        L(j,1)=A(j,1)/L(1,1)
        End for

Show “Stage 1= ”
Show “ Matrix L= “
Show L
        For i=2,3,...,n-1

Show “Stage i= “
        SumL1=0

        For p=1,2,...,i-1
        sumL1=sumL1+(L(i,p)^2)
        end for

L(i,i)= root(1,1)

        For j=i+1,i+2,...,n
        sumL2=0
        for p=1,2,...,i-1
        sumL2=sumL2+(L(j,p)*L(i,p)
        end for
        L(j,i)=(1/L(i,i))*(A(j,i)-sumL2)
        End for
```

```
If i=n-1
sumL3=0

        for p=1,2,...,n-1
        sumL3=sumL3+(L(n,p)^2)
        end for
        L(n,n)= root(1,1)

End if

Show “Matrix L= “
Show L
End for

U= Transposed L
z=zeros size (n,1)
sumZ=0
z(1,1)=b(1,1)/L(1,1)

        for i=2,3,...,n
        sumZ=0
        for s=1,2,...,n
        sumZ=sumZ+L(i,s)*z(s,1)
        end for
        z(i,1)=(b(i,1)-sumZ)/L(i,i)
        end for

show “Vector z= “
show z
x=zeros size (n,1)
```

```
sumX=0
x(n,1)=z(n,1)/U(n,n)

        for i=n,1,n-2,...,1
        sumX=0

        for s=n,n-1,...,1
        sumX=sumX+U(i,s)*x(s,1)
        end for

        x(i,1)=(z(i,1)-sumX)/U(i,i)
end for

show “Vector x= “
show x
show “X values are= “
        for i=1,2,...,n
        show xi= x(i,1)
        end for
```

Jacobi Pseudocode:

```
        Inputs:
        A=squared Matrix
        b=Vector b
        X0= Vector of initial values X0
        Tol= Tolerance
        Nmax=numbers of iterations
        create table on TXT format
```

found the rule 2 of A matrix

D=diagonal of the matrix A

L=D-triangular higher (A)

U=D-triangular lower (A)

Tj= D^(-1)*(L+U)

respec=maximum|eig(Tj)|

if respec > 1 then

Display the spectral radio is higher than 1, the method diverges.

end if

C=D^(-1)*b

i=0

error=Tol+1

While error > Tol e i < Nmax do

xi=Tj*x+C

i=i+1

error=norm(xi-x)

x=xi

p(i)=error

display (i)

for j = 1 until n

display(xi(j))

end for

end while

display(‘Radio espectral’)

Gauss-Seidel

Inputs:

A=squared Matrix

b=Vector b

X0= Vector of initial values X0

Tol= Tolerance

Nmax=numbers of iterations

create table on TXT format

found the rule 2 of A matrix

D=diagonal of the matrix A

L=D-triangular higher (A)

U=D-triangular lower (A)

Tj= (D-L)^-1 * U

respec=maximum|eig(Tj)|

if respec > 1 then

Display the spectral radio is higher than 1, the method diverges.

end if

C= (D-L)^-1 * b

i=0

error=Tol+1

While error > Tol e i < Nmax do

xi=Tj*x+C

i=i+1

error=norm(xi-x)

x=xi

p(i)=error

display (i)

for j = 1 until n

display(xi(j))

end for

end while

display('Radio espectral')

Lineal Plotter Pseudocode:

```
Inputs:
x= enter the points in x
y= enter the points in x
siz=size(x)
n=siz in position (1,2)
M= 6 * 6 of 0 matrix
M IN POSITION(1,2)=1
M IN POSITION(1,1)=X in position(1,1)
M IN POSITION(2,1)=X in position(1,2)
M IN POSITION(2,2)=1
M IN POSITION(3,3)=X in position(1,3)
M IN POSITION(3,4)=1
M IN POSITION(4,5)=X in position(1,4)
M IN POSITION(4,6)=1
M IN POSITION(5,1)=M IN POSITION(2,1)
M IN POSITION(5,2)=M IN POSITION(2,2)
M IN POSITION(5,3)=-M IN POSITION(2,1)
M IN POSITION(5,4)=-M IN POSITION(2,2)
M IN POSITION(6,3)=M IN POSITION(3,3)
M IN POSITION(6,4)=M IN POSITION(3,4)
M IN POSITION(6,5)=-M IN POSITION(3,3)
M IN POSITION(6,6)=-M IN POSITION(3,4)
print(M)
B=[y
    0
    0]
fact=solution between M and B
syms x
```

```
f1=fact in position(1,1)*x+fact in position(2,1)
f2=fact in position(3,1)*x+fact in position(4,1)
f3=fact in position(5,1)*x+fact in position(6,1)
print(f1)
print(f2)
print(f3)
```

Quadratic Plotter Pseudocode

```
x= enter the points in x
y= enter the points in x
siz=size(x)
n=siz(1,2)
M= 9 * 9 of 0 matrix
B=[y
   0
   0
   0
   0
   0]
M in position(1,1)=X in position(1,1)^2
M in position (1,2)=X in position(1,1)
M in position (1,3)=1
M in position (2,1)=X in position(1,2)^2
M in position (2,2)=X in position(1,2)
M in position (2,3)=1
M in position (3,4)=X in position(1,3)^2
M in position (3,5)=X in position(1,3)
M in position (3,6)=1
M in position (4,7)=X in position(1,4)^2
M in position (4,8)=X in position(1,4)
```

M in position (4,9)=1

M in position (5,1)=X in position(1,2)^2

M in position (5,2)=X in position(1,2)

M in position (5,3)=X in position(1,2)^0

M in position (5,4)=-M IN POSITION(2,1)

M in position (5,5)=-M IN POSITION(2,2)

M in position (5,6)=-M IN POSITION(2,3)

M in position (6,4)=X in position(1,3)^2

M in position (6,5)=X in position(1,3)

M in position (6,6)=X in position(1,3)^0

M in position (6,7)=-M IN POSITION(3,4)

M in position (6,8)=-M IN POSITION(3,5)

M in position (6,9)=-M IN POSITION(3,6)

M in position (7,1)=2*X in position(1,2)

M in position (7,2)=1

M in position (7,4)=-M IN POSITION(7,1)

M in position (7,5)=-M IN POSITION(7,2)

M in position (8,4)=2*X in position(1,3)

M in position (8,5)=1

M in position (8,7)=-M IN POSITION(8,4)

M in position (8,8)=-M IN POSITION(8,5)

M in position (9,1)=2

print(M)

fact=solution between M and B

syms x

fun1=fact in position(1,1)*x^2+fact in position(2,1)*x+fact in position(3,1)

fun2=fact in position(4,1)*x^2+fact in position(5,1)*x+fact in position(6,1)

fun3=fact in position(7,1)*x^2+fact in position(8,1)*x+fact in

```
position(9,1)
print(fun1)
print(fun2)
print(fun3)
```

Cubic Plotter Pseudocode:

```
x= enter the points in x
y= enter the points in x
siz=size (x)
n=siz IN POSITION (1,2)
M= 12 * 12 of 0 matrix
B=[y
0
0
0
0
0
0
0
0]
M IN POSITION(1,1)=X in position(1,1)^3
M IN POSITION(1,2)=X in position(1,1)^2
M IN POSITION(1,3)=X in position(1,1)
M IN POSITION(1,4)=1
M IN POSITION(2,1)=X in position(1,2)^3
M IN POSITION(2,2)=X in position(1,2)^2
M IN POSITION(2,3)=X in position(1,2)
M IN POSITION(2,4)=1
M IN POSITION(3,5)=X in position(1,3)^3
M IN POSITION(3,6)=X in position(1,3)^2
```

M IN POSITION(3,7)=X in position(1,3)

M IN POSITION(3,8)=1

M IN POSITION(4,9)=X in position(1,4)^3

M IN POSITION(4,10)=X in position(1,4)^2

M IN POSITION(4,11)=X in position(1,4)

M IN POSITION(4,12)=1

M IN POSITION(5,1)=M IN POSITION(2,1)

M IN POSITION(5,2)=M IN POSITION(2,2)

M IN POSITION(5,3)=M IN POSITION(2,3)

M IN POSITION(5,4)=M IN POSITION(2,4)

M IN POSITION(5,5)=-M IN POSITION(2,1)

M IN POSITION(5,6)=-M IN POSITION(2,2)

M IN POSITION(5,7)=-M IN POSITION(2,3)

M IN POSITION(5,8)=-M IN POSITION(2,4)

M IN POSITION(6,5)=M IN POSITION(3,5)

M IN POSITION(6,6)=M IN POSITION(3,6)

M IN POSITION(6,7)=M IN POSITION(3,7)

M IN POSITION(6,8)=M IN POSITION(3,8)

M IN POSITION(6,9)=-M IN POSITION(3,5)

M IN POSITION(6,10)=-M IN POSITION(3,6)

M IN POSITION(6,11)=-M IN POSITION(3,7)

M IN POSITION(6,12)=-M IN POSITION(3,8)

M IN POSITION(7,1)=3*X in position(1,2)^2

M IN POSITION(7,2)=2*X in position(1,2)

M IN POSITION(7,3)=1

M IN POSITION(7,5)=-M IN POSITION(7,1)

M IN POSITION(7,6)=-M IN POSITION(7,2)

M IN POSITION(7,7)=-M IN POSITION(7,3)

M IN POSITION(8,5)=3*X in position(1,3)^2

M IN POSITION(8,6)=2*X in position(1,3)

M IN POSITION(8,7)=X in position(1,3)^0

M IN POSITION(8,9)=-M IN POSITION(8,5)

M IN POSITION(8,10)=-M IN POSITION(8,6)

M IN POSITION(8,11)=-M IN POSITION(8,7)

M IN POSITION(9,1)=6*X in position(1,2)

M IN POSITION(9,2)=2

M IN POSITION(9,5)=-M IN POSITION(9,1)

M IN POSITION(9,6)=-M IN POSITION(9,2)

M IN POSITION(10,5)=6*X in position(1,3)

M IN POSITION(10,6)=2

M IN POSITION(10,9)=-M IN POSITION(10,5)

M IN POSITION(10,10)=-M IN POSITION(10,6)

M IN POSITION(11,1)=6*X in position(1,1)

M IN POSITION(11,2)=2

M IN POSITION(12,9)=6*X in position(1,4)

M IN POSITION(12,10)=2

print(M)

fact=solution between M and B

syms x

fun1=fact in position(1,1)*x^3+fact in position(2,1)*x^2+fact in

position(3,1)*x+fact in position(4,1)

fun2=fact in position(5,1)*x^3+fact in position(6,1)*x^2+fact in

position(7,1)*x+fact in position(8,1)

fun3=fact in position(9,1)*x^3+fact in position(10,1)*x^2+fact in

position(11,1)*x+fact in position(12,1)

print(fun1)

print(fun2)

print(fun3)

Simple Gaussian Elimination

Input: Matrix A, vector b

if matrix determinant = 0 then

break

M <- [A b]

for i from 0 to n-1 do

if M[i,i] = 0 then

find one row in the same column where M[i,column] is not zero

switch rows M[i,column] and M[i,i]

for j from i+1 to n do

if M[j,i] is not zero then

get the multiplier MULT = M(j,i)/M(i,i)

M[j,n] = M[j,i] - MULT * M[i,n]

solution = apply back_substitution to M

Lagrange

Input**:** n, xi, yi

x=variable

for j=1 until n

product=1

for i=1 until j-1

product equal product*(x-xi(i))

end

product2=1

for i=j plus 1 until n

product2=product2*(x-xi(i))

end

product3=1

```
for i=1 until j-1
product3=product3*(xi(j)-xi(i))
end
product4=1
for i=j plus 1 until n
product4=product4*(xi(j)-xi(i))
end
L(j)=(product*product2)/(product3*product4)
Show j-1
Show l(j)
end
pn=0
for j=1:n
pn=pn+L(j)*yi(j)
end
x= point for approximate
y=eval(pn)
show proximation
```

Newton with divided differences:

Input: function, delta, # of iterations, initial point

```
matx=Enter the values for X vertically
maty=Enter the values for y vertically
n=size(matx)
n=n(1)
column 1 of mat=matx

mat0= n * n+1 of 0 matrix
column 1 of mato=matx
column 2 of mat0=maty
k=-1
l=2
for j in a range of 3 to n+1
        for i in a range of l to n
mat0 IN POSITION (i,j)=(mat0 IN POSITION (i,j-1)-mat0 IN POSITION
(i-1,j-1))/(mat0 IN POSITION (i,1)-mat0 IN POSITION (i+k,1))
        endfor
        l=l+1
        k=k-1
endfor

print(&#39;&#39;)
print(&#39;Coeff matrix: &#39;)
print(mat0)

fprintf(&#39;Polynomial: \n&#39;)
pol=0;
```

```
for i in a range of 2 to n
        if I is equal to 2
        pol=pol+mat0 IN POSITION (1,2)
        endif
        pol=pol+mat0 IN POSITION (i-1,i)*((x^i-1)-1)
endfor

print(pol)
```

Gaussian Elimination with partial pivoting

Input: Matrix A, vector b

```
if matrix determinant = 0 then
break
M <- [A b]
for i from 0 to n-1 do
in the given column find the row with the greatest number (in abs)
if current row is different to the row found then
switch current row and found row
for j from i+1 to n do
if M[j,i] is not zero then
get the multiplier MULT = M(j,i)/M(i,i)
M[j,n] = M[j,i] - MULT * M[i,n]

solution = apply back_substitution to M
```

Gaussian Elimination with total pivoting

Input: Matrix A, vector b

if matrix determinant = 0 then

break

M <- [A b]

for i from 0 to n-1 do

column_swap = []

auxM = submatrix of M without the first row and column

find the row and column with the greatest number in auxM

if current column is different to the column found then

switch current column and found column

save positions changed in column_swap

if current row is different to the row found then

switch current row and found row

for j from i+1 to n do

if M[j,i] is not zero then

get the multiplier MULT = M(j,i)/M(i,i)

M[j,n] = M[j,i] - MULT * M[i,n]

pre-solution = apply back_substitution to M

reorder pre-solution according to column_swap to get the solution

Conclusions:

As a main language we decided to use Python, because it has features like easy to learn, easy to read, the source code is quite easy to maintain, most of the library is very portable and compatible with various platforms in UNIX, Windows and Macintosh. It is also portable and scalable.

As a second language we use MATLAB, because it is a mathematical software and in this case has a lot of advantages over other programming languages.

For the development of the website we use the flask microframework, this is minimalist and also written in Python. Flask allows you to create web applications quickly and with a minimum number of lines of code. It is based on Werkzeug's WSGI specification and the Jinja2 template engine and has a BSD license. We felt comfortable working with this tool, because as described, it is easy to develop and we could integrate it into the code we already had done in Python.

All the code we developed for the first deliveries is available at

https://github.com/sbedoyac1/Numerical-methods-web

All the code corresponding to the web pages are hosted at:

https://github.com/asperezm/WebNumeric