APPENDIX

Before using functions, the *linalg* package is need, so we should start with the following command:

```
>restart:with(linalg):
```

The function **diagmatrix(x, m, n, tol)** computes an **m** $\times$ **n** diagonal matrix with elements of reciprocal **x** on diagonal.

```
> diagmatrix:=proc(x,m,n,tol)
> local i,j,k,y,X;
> k:=vectdim(x);
> y:=array(1..k);
> for i from 1 to k do
> if abs(x[i])>tol then y[i]:=1/x[i] else y[i]:=0 fi
> od;
> X:=array(1..m,1..n);
> for i from 1 to m do
> for j from 1 to n do
> if i=j and i<=k then X[i,j]:=y[i] else X[i,j]:=0 fi
> od
> od;
> evalm(X);
> end:
```

Algorithm for Newton Method(**NewtonM1**).

To approximate the solution of the nonlinear system $\mathbf{f}(\mathbf{x}) = 0$ given an initial approximation $\mathbf{x}^0$:

INPUT : function $\mathbf{f}(\mathbf{x})$, initial approximation $\mathbf{x}^0 = (x_1^0, x_2^0, \ldots, x_n^0)^T$, tolerance ε, maximum number of iterations N.

OUTPUT : approximate solution $\mathbf{x}^k = (x_1^k, x_2^k, \ldots, x_n^k)^T$ and $\mathbf{f}(\mathbf{x}^k)$.

Step 1. Set $k = 0$.

Step 2. Print $\mathbf{x}^0, \mathbf{f}(\mathbf{x}^0)$.

Step 3. While $(k \leq N)$ do steps 4-11.

Step 4. Calculate $\mathbf{f}(\mathbf{x}^k)$ and $J_{\mathbf{f}}(\mathbf{x}^k)$.

Step 5. Decompose SVD of $J_{\mathbf{f}}(\mathbf{x}^k) = U\Sigma V^T$.

Step 6. Construct $\Sigma^{(2)}$ by using (**??**).

Step 7. Compute $\{2\}$-inverse of $J_{\mathbf{f}}(\mathbf{x}^k)$, $T_{\mathbf{f}}(\mathbf{x}^k) = V\Sigma^{(2)}U^T$.

Step 8. Compute $\mathbf{x}^{k+1} = \mathbf{x}^k - T_{\mathbf{f}}(\mathbf{x}^k)\mathbf{f}(\mathbf{x}^k)$.

Step 9. Print $\mathbf{x}^{k+1}, \mathbf{f}(\mathbf{x}^{k+1})$.

Step 10. If $\|\mathbf{x}^{k+1} - \mathbf{x}^k\| \leq \varepsilon$ then

STOP.

Step 11. Set $k = k + 1$.

Step 12. STOP.

The function **NewtonM1(f, x, x0, tol, N)** computes **N** iteration of the Newton Method starting with **x0**, the tolerance is fixed.

```
> NewtonM1:=proc(f,x,x0,tol,N)
> local valf,m,n,i,jb,sigma,U,V,SIGMA,T,soln,err;
> n:=vectdim(x);
> m:=vectdim(f);
> valf:=eval(subs(x=x0,f)):
> lprint(iteration,0):print(x0);
> lprint(function):print(valf);soln[0]:=x0;
> for i from 1 to N do
> jb:=eval(jacobian(f,x),x=soln[i-1]);
> sigma:=evalf(Svd(jb,U,V));
> U:=evalm(U);V:=eval(V);
> SIGMA:=diagmatrix(sigma,n,m,tol);
> T:=evalm(V&*SIGMA&*transpose(U));
> soln[i]:=evalm(soln[i-1]-T&*valf);
> valf:=eval(subs(x=soln[i],f)):
> lprint(iteration,i):print(soln[i]);
> lprint(function):print(valf);
> if (sqrt(norm(soln[i]-soln[i-1],2))<tol) then break fi:
> od;
> lprint(solution ):print(soln[i]);
> lprint(function):print(valf);
> end:
```

Algorithm for Newton Method(**NewtonM2**).

To approximate the solution of the nonlinear system $\mathbf{f}(\mathbf{x}) = 0$ given an initial approximation $\mathbf{x}^0$:

INPUT : function $\mathbf{f}(\mathbf{x})$, initial approximation $\mathbf{x}^0 = (x_1^0, x_2^0, \ldots, x_n^0)^T$, tolerance ε ,maximum number of iterations N.

OUTPUT : approximate solution $\mathbf{x}^k = (x_1^k, x_2^k, \ldots, x_n^k)^T$ and $\mathbf{f}(\mathbf{x}^k)$.

Step 1. Set $k = 0, \varepsilon^0 = 100$.

Step 2. Print $\mathbf{x}^0, \mathbf{f}(\mathbf{x}^0)$.

Step 3. While $(k \leq N)$ do steps 4-11.

Step 4. Calculate $\mathbf{f}(\mathbf{x}^k)$ and $J_{\mathbf{f}}(\mathbf{x}^k)$.

Step 5. Decompose SVD of $J_{\mathbf{f}}(\mathbf{x}^k) = U\Sigma V^T$.

Step 6. Construct $\Sigma^{(2)}$ by using (**??**).

Step 7. Compute $\{2\}$-inverse of $J_{\mathbf{f}}(\mathbf{x}^k)$, $T_{\mathbf{f}}(\mathbf{x}^k) = V\Sigma^{(2)}U^T$.

Step 8. Compute $\mathbf{x}^{k+1} = \mathbf{x}^k - T_{\mathbf{f}}(\mathbf{x}^k)\mathbf{f}(\mathbf{x}^k)$.

Step 9. If $\|\mathbf{x}^{k+1} - \mathbf{x}^k\| = 0$ then $\varepsilon^k = \dfrac{\varepsilon^k}{10}$.

Repeat steps 6-9.

Step 10. Print $\mathbf{x}^{k+1}$, $\mathbf{f}(\mathbf{x}^{k+1})$.

Step 11. If $\|\mathbf{x}^{k+1} - \mathbf{x}^k\| \leq \varepsilon$ then

STOP.

Step 12. $\varepsilon^{k+1} = \dfrac{\varepsilon^k}{10}$

Step 13. Set $k = k + 1$.

Step 14. STOP.

The function **NewtonM2(f, x, x0, tol, N)** computes **N** iteration of the Newton Method starting with **x0**, the tolerance is updated.

```
> NewtonM2:=proc(f,x,x0,tol,N)
> local valf,m,n,i,jb,sigma,U,V,SIGMA,T,soln,newtol;
> n:=vectdim(x);
> m:=vectdim(f);
> newtol:=100;
> soln[0]:=x0;
> valf:=eval(subs(x=x0,f)):
> lprint(iteration,0):print(x0);
> lprint(function):print(valf);
> jb:=eval(jacobian(f,x),x=soln[0]);
> sigma:=evalf(Svd(jb,U,V));
> U:=evalm(U);V:=evalm(V);
> SIGMA:=diagmatrix(sigma,n,m,newtol);
> T:=evalm(V&*SIGMA&*transpose(U));
> soln[1]:=evalm(soln[0]-T&*valf);
> while (norm(soln[1]-soln[0],2)=0) do
> newtol:=newtol/10;
> SIGMA:=diagmatrix(sigma,n,m,newtol);
> T:=evalm(V&*SIGMA&*transpose(U));
> soln[1]:=evalm(soln[0]-T&*valf);
> od:
> valf:=eval(subs(x=soln[1],f)):
> lprint(iteration,1):print(soln[1]);
> lprint(function):print(valf);
> for i from 2 to N do
> jb:=eval(jacobian(f,x),x=soln[i-1]);
```

```
> sigma:=evalf(Svd(jb,U,V));
> U:=evalm(U);V:=eval(V);
> SIGMA:=diagmatrix(sigma,n,m,newtol);
> T:=evalm(V&*SIGMA&*transpose(U));
> soln[i]:=evalm(soln[i-1]-T&*valf);
> if (newtol>tol) then newtol:=newtol/10 fi:
> valf:=eval(subs(x=soln[i],f)):
> lprint(iteration,i):print(soln[i]);
> lprint(function):print(valf);
> if (sqrt(norm(soln[i]-soln[i-1],2))<tol) then break fi:
> od:
> lprint(solution ):print(soln[i]);
> lprint(function):print(valf);
> end:
```

In the examples below the equations in all systems have zero on the right hand side, so the values of the functions give an indication of the error. Also, the functions use a vector variable **x** of unspecified dimension, making it necessary to define the dimension, say

```
> x:=array(1..3):
```


Example 1

$$\begin{aligned} 3x_1^2 - x_2 &= 0 \\ e^{1-x_1-x_2-x_3} - 1 &= 0 \end{aligned}$$

```
> x:=array(1..3);
```

$$x := \text{array}(1..3, [])$$

```
>NewtonM1([3*x[1]^2-x[2],exp(1-x[1]-x[2]-x[3])-1],x,[1.,1.,1.2],10^(-12)
,15);  iteration, 0
```

$$[1.,\ 1.,\ 1.2]$$

function

[2, -.8891968416]

iteration, 15

[-.7096950372, 1.511001137, .1986939000]

function

[0., 0.]

Example 2

$$3x_1^2 - x_2 = 0$$

$$e^{1-x_1-x_2-x_3} - 1 = 0$$

```
> x:=array(1..3);
```

$x := \text{array}(1..3, [])$

```
>NewtonM1([3*x[1]^2-x[2],exp(1-x[1]-x[2]-x[3])-1],x,[1.2,1.1,1.0],10^
(-12),15); iteration, 0
```

[1.2, 1.1, 1.0]

function

[3.22, -.8997411563]

iteration, 13

[.6271689951, 1.180022845, -.8071918400]

function

[0., -.1 10^{-9}]

Example 3

$$x_1 - \cos x_2 = 0$$

$$x_2 - \cos x_3 = 0$$

```
> x:=array(1..3);
```

$x := \text{array}(1..3, [])$

>NewtonM1($[x[1]-cos(x[2]), x[2]-cos(x[3])], x, [1.2, 1.2, 1.5], 10\text{^}(-12), 15$);

iteration, 0

$$[1.2,\ 1.2,\ 1.5]$$

function

$$[.8376422455,\ 1.129262798]$$

iteration, 8

$$[.8790143305,\ .4970053623,\ 1.050652023]$$

function

$$[0.,\ 0.]$$

Example 4

$$\begin{aligned} x_1 - \cos x_2 &= 0 \\ x_2 - \cos x_3 &= 0 \end{aligned}$$

> x:=array(1..3);

$$x := \text{array}(1..3, [])$$

>NewtonM2($[x[1]-cos(x[2]), x[2]-cos(x[3])], x, [1.2, 1.2, 1.0], 10\text{^}(-12), 15$);

iteration, 0

$$[1.2,\ 1.2,\ 1.0]$$

function

$$[.8376422455,\ .6596976941]$$

iteration,6

$$[.7826748625,\ .6718445996,\ .8340999703]$$

function

$$[0.,\ 0.]$$

Example 5

$$\begin{aligned} x_1^2 - x_2 &= 0 \\ x_2^2 - x_3 &= 0 \\ e^{1-x_1-x_2-x_3} - 1 &= 0 \end{aligned}$$

```
> x:=array(1..3);
```

$$x := \text{array}(1..3, [])$$

```
>NewtonM2([x[1]^2−x[2], x[2]^2−x[3],exp(1−x[1]−x[2]−x[3])−1], x, [1.2, 1.1, 1.0],
10^(−12), 15);
iteration, 0
```

[1.2, 1.1, 1.0]

```
function
```

[.34, .21, -.8997411563]

```
iteration,14
```

[.5698402909, .3247179571, .1054417517]

```
function
```

[0., 0.,0.]

VITA

Name	:	Mr. Prapart Pue-on
Date of Birth	:	May 27, 1979
Institutions Attended	:	Bachelor of science (B.S.)(Mathematics) from Khon Kaen University in 2002 Khon Kaen, Thailand.
Scholarship	:	UDC 2002-2003.