The Abdus Salam
International Centre for Theoretical Physics

United Nations
Educational, Scientific and
Cultural Organization

IAEA
International Atomic Energy Agency

1856-57

2007 Summer College on Plasma Physics

30 July - 24 August, 2007

Numerical methods and simulations. Lecture 4: Solution of nonlinear systems of equations and nonlinear boundary value problems

B. Eliasson
Rhur University
Bochum, Germany

Summer College on Plasma Physics
ICTP
Trieste, Italy
30 July – 24 August 2007

Course: **Numerical methods and simulations**

Tutor: Bengt Eliasson
E-mail: bengt@tp4.rub.de
Internet address: `www.tp4.rub.de/∼bengt`

Lecture 4: Solution of nonlinear systems of equations and boundary value problems.

- We will study Newton'a method to solve nonlinear systems of equations that cannot be solved analytically.
- Nonlinear equations or systems of equations frequently occur in
  - Dispersion relations (to solve ω as a function of k)
  - Nonlinear boundary value problems, for example Poisson's equation with Boltzmann electrons

    $$\frac{d^2\phi}{dx^2} = \exp(\phi) - n_i(x) \tag{1}$$

    with the potential given at $x = 0$ and $x = L$.
  - Implicit time-stepping methods (e.g. Crank-Nicholson's scheme) where one has to solve an equation in each time-step to obtain the solution.

Newton's method is base on Taylor expansion of the nonlinear system. Assume that we want to solve the nonlinear equation $G(v^*) = 0$ to obtain the unknown value v^*. As an initial guess we use v, and we have that $v^* = v + \delta v$ where δv is the correction of v to obtain the real solution v^*. We can then Taylor expand $G(v)$ around v so that $G(v^*) = G(v + \delta v) \approx G(v) + \delta v G'(v)$. Since $G(v^*) = 0$ we have $\delta v \approx -G(v)/G'(v)$, and in the Newton algorithm we have the new value $v^{new} \leftarrow v^{\rm old} - G(v^{\rm old})/G'(v^{\rm old})$.

Example: Solve the nonlinear equation $x = \cos(x)$ numerically. First rewrite the equation as $G(x) = x - \cos(x) = 0$. Then $G'(x) = 1 + \sin(x)$, and we have the correction term $\delta x = -G(x)/G'(x) = -(x - \cos(x))/(1 + \sin(x))$. The Newton algorithm becomes $x^{new} \leftarrow x^{\text{old}} - (x^{\text{old}} - \cos(x^{\text{old}}))/(1+\sin(x^{\text{old}}))$. If we take the initial guess $x = 1$, the first few iterations give

1	$\delta x = -0.24963613215976$	$x = 0.75036386784024$
2	$\delta x = -0.01125097692888$	$x = 0.73911289091136$
3	$\delta x = -0.00002775752607$	$x = 0.73908513338528$
4	$\delta x = -0.00000000017012$	$x = 0.73908513321516$

After only 4 iterations, we have a solution with 10 digits accuracy! It is typical for the Newton method that the number of accurate digits in the solution doubles in each iteration. One can show that the correction term decreases as $|\delta x^{\text{new}}| = C|\delta x^{\text{old}}|^2$ for most problems, where C is a constant.

In multiple dimensions, we may want to solve the nonlinear system of equations $\vec{G}(\vec{v}^*) = \vec{0}$ to obtain the vector $\vec{v}^*$. With the initial guess $\vec{v}$ we have $\vec{v}^* = \vec{v} + \delta\vec{v}$ where $\delta\vec{v}$ is the correction of $\vec{v}$ to obtain $\vec{v}^*$. Taylor expanding $\vec{G}$ around $\vec{v}$ we have $\vec{G}(\vec{v}^*) = \vec{G}(\vec{v}+\delta\vec{v}) \approx \vec{G}(\vec{v}) + \mathbf{J}\delta\vec{v}$ where

$$\mathbf{J} = \begin{bmatrix} \frac{\partial G_1}{\partial v_1} & \frac{\partial G_1}{\partial v_2} & \cdots & \frac{\partial G_1}{\partial v_N} \\ \frac{\partial G_2}{\partial v_1} & \frac{\partial G_2}{\partial v_2} & & \\ \vdots & & \ddots & \vdots \\ \frac{\partial G_N}{\partial v_1} & & \cdots & \frac{\partial G_N}{\partial v_N} \end{bmatrix} \tag{2}$$

is the Jacobian matrix. Since $\vec{G}(\vec{v}^*) = 0$ we can solve for $\delta\vec{v}$ to obtain $\delta\vec{v} \approx -\mathbf{J}^{-1}\vec{G}(\vec{v})$ and in the Newton algorithm we have the new value $\vec{v}^{new} = \vec{v} - \mathbf{J}^{-1}\vec{G}(\vec{v})$. The iterations are stopped when the norm of the correction term, $||\delta\mathbf{v}||$, is smaller than some small value ϵ. The norm is a measure of the "length" of the vector. For example the Euclidian norm (`norm(x,2)` in Matlab), is defined as $||\vec{x}||_2 = \sqrt{x_1^2 + x_2^2 + \ldots + x_N^2}$, and the max norm (`norm(x,inf)` in Matlab) is defined as $||\delta\vec{x}||_\infty = \max(|x_1|, |x_2|, \ldots, |x_N|)$.

As an example, we will now make a program to solve the system of two coupled equations

$$x = y + \cos(x + y) \tag{3}$$

$$y = x + \sin(xy) \tag{4}$$

We first rewrite the system in the form $\vec{G}(x,y)=0$, or

$$\vec{G}=\begin{bmatrix}G_1\\G_2\end{bmatrix}=\begin{bmatrix}x-y-\cos(x+y)\\ y-x-\sin(xy)\end{bmatrix}=\begin{bmatrix}0\\0\end{bmatrix} \tag{5}$$

and the Jacobian then becomes

$$\mathbf{J}=\begin{bmatrix}\partial G_1/\partial x & \partial G_1/\partial y\\ \partial G_2/\partial x & \partial G_2/\partial y\end{bmatrix}=\begin{bmatrix}1+\sin(x+y) & -1+\sin(x+y)\\ -1-y\cos(xy) & 1-x\cos(xy)\end{bmatrix} \tag{6}$$

We now write the main program "main.m" for the solution of the two coupled equations. Select from the Matlab menu *File / New / M-file*, then from the menu of the new window, choose *File / Save as*, and Filename: `main.m`

We now write the program in the file:

==========

```
% Main program to solve two coupled nonlinear equations, main.m

% Initial guess
x=1; y=1;

% Create solution vector
v(1,1)=x;
v(2,1)=y;

% Solve with the Newton algorithm
v=Newton(v);

% Store the result
x=v(1,1);
y=v(2,1);

%Print the result
disp(['x=' num2str(x,'%0.10g') ' y=' num2str(y,'%0.10g')])
```

==========

Then choose from the menu *File / Save*.

Now we write the Newton subroutine: From the Matlab menu, choose *File / New / M-file*, then choose from the menu of the new file *File / Save as*, and Filename: `Newton.m`

In the new file, we write

==========

```
% The Newton algorithm: Newton.m

% As input we have the initial guess.
function v=Newton(v)

  dv=1; % dv is the norm of the correction term.
  epsilon=1e-6; % The accuracy limit.

  counter=0;

  %Iterate the Newton algorithm until
  %the correction is smaller than epsilon.
  while(norm(dv,inf)>epsilon)

    dv=-Jacobian(v)\G(v); % Solve the linear equation system J*delta=-F.
    v=v+dv; % Update the solution

    disp(['|dx|=' num2str(norm(dx,inf),'%0.10g') ' x=' ...
      num2str(v(1,1),'%0.10g') ' y=' num2str(v(2,1),'%0.10g')]);


    counter=counter+1;

    % Check if we iterate more than 20 times, then no convergence.
    if counter>20
      disp('No convergence!')
      return;
    end

  end
```

==========

Choose *File / Save*.

Next, we write the function that defines the system of nonlinear equations. From the Matlab menu, choose *File / New / M-file*, and then *File / Save as* and Filename: `G.m`

In the new file, we write

==========

```
% Definition of the nonlinear system G(x,y)=0;

function G=G(v)

x=v(1,1);
y=v(2,1);

% The system of nonlinearly coupled equations
G(1,1)=x-y-cos(x+y);
G(2,1)=y-x-sin(x*y);
```

==========

Choose *File / Save*.

Finally, we write the function that defines the Jacobian for our system of nonlinear equations. From the Matlab menu, choose *File / New / M-file*, and then *File / Save as* and Filename: `Jacobian.m`

In the new file, we write

==========

```
% The Jacobian matrix dG/dv.

function Jacobian=Jacobian(v)

  x=v(1,1);
  y=v(2,1);

  Jacobian(1,1)=1+sin(x+y);    % dG1/dv1
  Jacobian(1,2)=-1+sin(x+y);   % dG1/dv2
  Jacobian(2,1)=-1-y*cos(x*y); % dG2/dv1
  Jacobian(2,2)=1-x*cos(x*y);  % dG2/dv2
```

==========

Choose *File / Save*.

We now run the program. In the Matlab window, type in `main` (Enter).

The result is

```
||dv||=1.308454231 x=0.8442009784 y=2.308454231

||dv||=0.4010851826 x=0.9036509561 y=1.907369048
```

```
||dv||=0.0539587817 x=0.9576097378 y=1.927521852

||dv||=0.006653695218 x=0.9557371965 y=1.920868157

||dv||=3.898306889e-005 x=0.9557422362 y=1.920829174

||dv||=3.816997014e-010 x=0.9557422366 y=1.920829173

x=0.9557422366 y=1.920829173
```

We see that we have 10 significant digits after 6 iterations! Also here it is typical for the Newton method that the number of accurate digits in the solution doubles in each iteration. One can show that when $||\delta\mathbf{v}||$ is small enough, the correction term decreases as $||\delta\mathbf{v}^{\rm new}|| = C||\delta\mathbf{v}^{\rm old}||^2$ for most problems, where C is a constant.

It is necessary to give an initial guess that is close enough to the solution that we want. For example, the initial guess $x = y = 10$ gives different solution from that above, and the initial guess $x = 10$, $y = -10$ leads to no convergence!

In Matlab, there also exists a command *roots* to find all roots of a polynomial. This is very useful to find solutions of dispersion relations in fluid plasmas. Example: to find the complex roots of the polynomial

$$w^4 + 2w^3 + 3w^2 + w + 4 = 0 \tag{7}$$

we do the command in Matlab

```
roots([1 2 3 1 4])
```

The result is

```
ans =

  -1.3030 + 1.3062i
  -1.3030 - 1.3062i
   0.3030 + 1.0409i
   0.3030 - 1.0409i
```

i.e., the complex roots are

$w_1 = -1.3030 + 1.3062i$, $w_2 = -1.3030 - 1.3062i$, $w_3 = 0.3030 + 1.0409i$ and $w_4 = 0.3030 - 1.0409i$

Example 2/Assignment (Nonlinear boundary value problem):

Write a program that solves the nonlinear Poisson equation

$$\frac{\partial^2 \phi}{\partial x^2} = \exp(\phi) - n_i(x) \tag{8}$$

where $n_i = 1 + \exp(-3(x-5)^2)$, on the domain $0 \leq x \leq L$ with the boundary conditions $\phi(0) = 0$ and $\phi(L) = 0$ with $L = 10$.

First we write the equation as

$$\frac{\partial^2 \phi}{\partial x^2} - \exp(\phi) + n_i(x) = 0. \tag{9}$$

Then discretize the space into discrete points $x = x_0, x_1, \ldots, x_{Nx}$ and represent the space in a column vector

$$\vec{x} = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{Nx} \end{bmatrix} \tag{10}$$

Then discretize the solution as $\phi = \phi_0, \phi_1, \ldots, \phi_{Nx}$, where $\phi_j \approx \phi(x_j)$. Order the solution in a column vector as

$$\vec{\phi} = \begin{bmatrix} \phi_0 \\ \phi_1 \\ \vdots \\ \phi_{Nx} \end{bmatrix} \tag{11}$$

In order to obtain the $N_x + 1$ unknowns ϕ_j, we need $N_x + 1$ equations which we symbolically denote $\vec{G}(\vec{\phi}, x) = 0$, where the equations are ordered into a column vector

$$\vec{G} = \begin{bmatrix} G_0 \\ G_1 \\ \vdots \\ G_{Nx} \end{bmatrix} \tag{12}$$

We now find the equations. The left-hand boundary condition is $\phi_0 = 0$ so that $G_0 = \phi_0$, while the right-hand boundary condition is $\phi_{Nx} = 0$ so that $G_{Nx} = \phi_{Nx}$. Between the boundaries, we have the nonlinear Poisson equation, which we approximate with a centered difference method so that

$$\frac{\phi_{j-1} - 2\phi_j + \phi_{j+1}}{\Delta x^2} - \exp(\phi_j) + n_i(x_j) = 0, \quad j = 1, \ldots, N_x - 1, \tag{13}$$

hence

$$G_j = \frac{\phi_{j-1} - 2\phi_j + \phi_{j+1}}{\Delta x^2} - \exp(\phi_j) + n_i(x_j), \quad j = 1, \ldots, N_x - 1. \tag{14}$$

Then the Jacobain matrix becomes

$$\mathbf{J} = \begin{bmatrix} \frac{\partial G_0}{\partial \phi_0} & \frac{\partial G_0}{\partial \phi_1} & \cdots & \frac{\partial G_0}{\partial \phi_N} \\ \frac{\partial G_1}{\partial \phi_0} & \frac{\partial G_1}{\partial \phi_1} & & \\ \vdots & & \ddots & \vdots \\ \frac{\partial G_{Nx}}{\partial \phi_0} & & \cdots & \frac{\partial G_{Nx}}{\partial \phi_{Nx}} \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & \cdots & & 0 \\ \frac{1}{\Delta x^2} & -\frac{2}{\Delta x^2} - \exp(\phi_1) & \frac{1}{\Delta x^2} & & \\ 0 & \frac{1}{\Delta x^2} & -\frac{2}{\Delta x^2} - \exp(\phi_2) & \frac{1}{\Delta x^2} & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & & \frac{1}{\Delta x^2} & -\frac{2}{\Delta x^2} - \exp(\phi_{Nx-1}) & \frac{1}{\Delta x^2} \\ 0 & \cdots & & 0 & 1 \end{bmatrix} \tag{15}$$

We now write the program to solve the nonlinear system of equations. Fitst the main program, *main.m*:

```
=======================================

% Main program to solve two coupled nonlinear equations: main.m
clear

L=10; Nx=100; dx=L/Nx; x=(0:Nx)'*dx; ni=1+exp(-3*(x-5).^2);

% Initial guess
phi=log(ni);

dx
% Solve with the Newton algorithm
phi=Newton(phi,x,ni,dx,Nx);

%Plot the result
subplot(3,1,1) plot(x,phi) title('\phi') subplot(3,1,2)
plot(x,exp(phi)) title('n_e') subplot(3,1,3) plot(x,ni) title('n_i')

=======================================
```

Next we wtite the Newton algorithm in the file *Newton.m*

==========

```
% The Newton algorithm: Newton.m

% As input we have the initial guess.
function phi=Newton(phi,x,ni,dx,Nx)

  dphi=1; % dx is the norm of the correction term.
  epsilon=1e-6; % The accuracy limit.

  counter=0;

  %Iterate the Newton algorithm until
  %the correction is smaller than epsilon.
  while(norm(dphi,inf)>epsilon)

    % Solve the linear equation system J*delta=-F.
    dphi=-Jacobian(phi,x,ni,dx,Nx)\G(phi,x,ni,dx,Nx);

    phi=phi+dphi; % Update the solution

    disp(['||dphi||=' num2str(norm(dphi,inf),'%0.10g')]);

    counter=counter+1;

    % Check if we iterate more than 20 times, then no convergence.
    if counter>20
      disp('No convergence!')
      return;
    end

  end
```

==========

The system of equations $\vec{G}$ is calculated in the the file *G.m*:

==========

```
% Definition of the nonlinear system: G.m
```

```
function G=G(phi,x,ni,dx,Nx)


% The system of nonlinearly coupled equations
G(1,1)=phi(1,1);
G(2:Nx,1)=(phi(1:Nx-1,1)-2*phi(2:Nx,1)+phi(3:Nx+1,1))/dx^2 ...
  -exp(phi(2:Nx,1))+ni(2:Nx,1); G(Nx+1,1)=phi(Nx+1,1);
```

==========

Finally, the Jacobian matrix is calculated in the file *Jacobian.m*

==========

```
% The Jacobian matrix dG/dphi: Jacobian.m

function Jacobian=Jacobian(phi,x,ni,dx,Nx)

  Jacobian=sparse(0); % Store only non-zero elements

  Jacobian(1,1)=1;
  for j=2:Nx
      Jacobian(j,j-1)=1/dx^2;
      Jacobian(j,j)=-2/dx^2-exp(phi(j,1));
      Jacobian(j,j+1)=1/dx^2;
  end
  Jacobian(Nx+1,Nx+1)=1;
```

==========

Now try to run the program by the command `main` in matlab. The printout on screen shows the norm of the correction term, which is a measure of the convergence of the method.

Printout:

```
||dphi||=0.3246604088

||dphi||=0.02132031524

||dphi||=0.0001206819599

||dphi||=4.499908369e-009
```

After 4 iterations, we have the solution with about 10 digits accuracy. The plot of the solution is