Simulation in Materials Engineering

BLOCK 2: Fundamentals of numerical analysis

J. Segurado

Departamento de Ciencia de Materiales
Polytechnic University of Madrid

October 5, 2011

Outline

Outline

Introduction

A linear operator or function $\mathbf{L}(\mathbf{x})$, is a function that fulfills two conditions

1. additivity $\mathbf{L}(\mathbf{x}+\mathbf{y}) = \mathbf{L}(\mathbf{x}) + \mathbf{L}(\mathbf{y})$
2. homogenety $\mathbf{L}(\alpha\mathbf{x}) = \alpha\mathbf{L}(\mathbf{x})$

and a linear algebraical equation is a equation $\mathbf{L}(\mathbf{x}) = \mathbf{b}$.
If an operator or function $\mathbf{F}(\mathbf{x})$ does not fulfill (1) and (2), then the resulting equation (or system of equations)

$$\mathbf{F}(\mathbf{x}) = \mathbf{b}$$

is a non linear equation
Mathematical examples are

$$x = \tan(x)$$

Introduction

Non-linear equations and systems appear very often in physics and Engineering because real systems have in general non-linear response. Some physical examples can be

- The Van der Waals state equation is a non-linear equation relating p, V, T,

$$[p + a(n/V)^2](V - nb) = nRT$$

- An elasto-viscoplastic material (or any other non-linear response) has a non-linear response, and the discretization of its behavior leads to non-linear algebraical equations as

$$(\Delta\sigma = E\Delta t(\dot{\varepsilon} - \dot{\varepsilon}_{vp})$$

Introduction

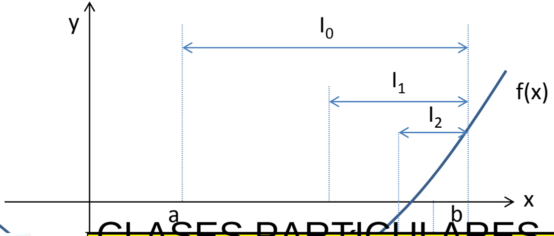
- The solution of non-linear equations or systems of equations in general cannot be accomplished by a finite number of operations.
- In addition, solutions are also in general not unique: i.e. the zeros of a 3er order polynomial function can have 3 different solutions
- Iterative methods are normally adopted: a sequence of vector $\mathbf{x}^{(k)}$ is searched that will converge to a solution $\mathbf{x}$ of the system
- Two methods will be covered within this course: *bisection* and *Newton-Raphson*

Outline

Bisection method

Let f be a continuous function in $[a, b]$ / $f(a)f(b) < 0$. Then f has at least one zero α in (a, b), $f(\alpha) = 0$

The bisection method iteratively halves the actual interval I_k, and from the two halves defines the new interval I_{k+1} as the interval where f still satisfies $f(a)f(b) < 0$



Bisection method

- The succession of points $\{x^{(k)}\}$ defined as the mid-points of the intervals I_k converges to the solution α, $f(\alpha) = 0$.
- As $\|I_k\| = (1/2)^k \|I_0\|$, then the error $|x^{(k)} - \alpha| = (1/2)\|I_k\| = (1/2)^{k+1}(b - a)$
- The number of iterations k_ϵ needed to obtain α with an error ϵ is

$$k_\epsilon \geq \operatorname{int}\left(\frac{\log[(b-a)/\epsilon]}{\log 2}\right)$$

- A general algorithm should have the function $f(x)$ as input, as well as a, b, ϵ

Bisection method

A function in MATLAB/Octave can be stored in several ways. In order to pass the function itself as a variable to other program or function it is useful to define it as `fun=inline('1/(1+x^2)')`
The function can be plotted directly using `fplot(fun,[a,b])`
In order to evaluate the function for a specific value of x, `feval(fun,x)`

OCTAVE/MATLAB exercise

Define a non-linear function $f(x)$ that has a zero and represent that function within an interval [a,b] that includes the x where $f = 0$.

Bisection method

Algorithm of bisection

```
function [solution,niter]=bisection(fun,a,b,epsilon)
niter=0;
if feval(fun,a)*feval(fun,b) >0
  disp('error, same sign in a and b');
  return
elseif abs(feval(fun,a))<=epsilon
  disp('The solution is a');
  solution=a;
  return
elseif abs(feval(fun,b))<=epsilon
  disp('The solution is b');
  solution=b;
  return
endif

if (b-a)>0
  error=(b-a)/2;
  interval(1)=a;
  interval(2)=(a+b)/2;
  interval(3)=b;
else
  error=(a-b)/2;
```

```
while error>=epsilon
  niter=niter+1;
  for i=1:3
    f(i)=feval(fun,interval(i));
    if(abs(f(i))<=epsilon)
      solution=interval(i);
      return
    endif
  endfor
  if f(1)*f(2)<0
    interval(3)=interval(2);
    interval(2)=(interval(1)+interval(2))/2;
  elseif f(1)*f(3)<0
    interval(1)=interval(2);
    interval(2)=(interval(1)+interval(3))/2;
  endif
  error=(interval(3)-interval(1))/2;
endwhile
solution=interval(2);
```

Bisection method

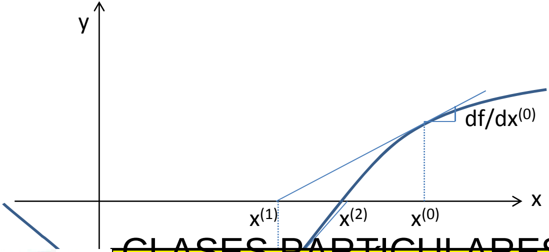
OCTAVE/MATLAB exercise

- Use the non-linear function $f(x)$ defined in previous exercise and obtain its solution in $[a, b]$ using the bisection algorithm, obtain also the number of iterations needed.
- Do the same with to solve the equation $\cos(x) = x$ in $[.5, 1.]$
- For $P = P_{atm}$ and T=273K calculate the volume of 1 mol of a Van der Waals gas with a=0.364 Pa m^6/mol^2, b=4.267 10^{-5} m^3/mol, help use `fun=inline('(P+a/V^2)*(V-b)-R*T','V')` to define V as variable and the rest as parameters. Compare with an ideal gas.

Outline

Newton-Raphson method

- In the bisection method, the sign of f at the endpoints of the subintervals is the only information exploited
- More efficiency can be obtained in the case of a differentiable function by exploiting the values of f and its derivative



Newton-Raphson method

Given a point $x^{(k)}$, then the equation of a line passing through $x^{(k)}$ and being tangent to f is obtained by

$$y(x) = f(x^{(k)}) + \left.\frac{\mathrm{d}f}{\mathrm{d}x}\right|_{x=x^{(k)}} (x - x^{(k)})$$

The line cuts the x axis when $y = 0$ and that point will be taken as the next iteration of x,$x^{(k+1)}$

$$y(x^{(k+1)}) = 0 = f(x^{(k)}) + \left.\frac{\mathrm{d}f}{\mathrm{d}x}\right|_{x=x^{(k)}} (x^{(k+1)} - x^{(k)})$$

And operating the new approach corresponds to

Newton-Raphson method

- The method finds a zero of f starting from $x^{(0)}$.
- In general does not converge for any$x^{(0)}$, but only for values sufficiently close to α.!! A good *predictor* is needed!!
- If converges, the method is much faster than bisection. Let f be derivable up to 2nd order, it can be prooved that

$$\lim_{k\to\infty} \frac{x^{(k+1)} - \alpha}{(x^{(k)} - \alpha)^2} = \frac{f''(\alpha)}{2f'(\alpha)} = cte$$

This mean that the method has *quadratic convergency*, the error at step $k+1$ is the square of the error in k multiolied by a constant

Newton-Raphson

OCTAVE/MATLAB exercise

- Complete the function to define a Newton-Raphson algorithm

```
function  [solution,niter]=newton_raphson(fun,dfun,x0,epsilon)
error=10*epsilon;
x=x0;
niter=0;
while (error>epsilon)
  niter=niter+1
  value=feval(fun,x);
  deriv=feval(dfun,x);
  !! here define new x, error and new x0
endwhile
solution=x;
return
end
```

- Include an *exit* in case the number of iterations is bigger than 100
- Use the algorithm to obtain the zeros of the functions defined previously. Compare the number of iterations needed.
- Check if the problem achieves quadratic convergency

Outline

Systems of non-linear equations

Lets consider a system of n non-linear equations of the form

$$\begin{cases} f_1(x_1, x_2, \cdots, x_n) = 0 \\ f_2(x_1, x_2, \cdots, x_n) = 0 \\ \vdots \\ f_n(x_1, x_2, \cdots, x_n) = 0 \end{cases}$$

where the functions f_i are non linear functions depending on variables $x_1, \cdots, x_n$. The system can be written in vectorial form by setting $\mathbf{f} = (f_1, f_2, \cdots f_n)^T$ and $\mathbf{x} = (x_1, x_2, \cdots, x_n)^T$,

$$\mathbf{f}(\mathbf{x}) = \mathbf{0}$$

As example

Systems of non-linear equations

- The Newton- Raphson method adapted to systems can be used to solve $\mathbf{f}(\mathbf{x}) = \mathbf{0}$
- Let $\mathbf{J_f}$ be the Jacobian matrix of the system

$$\mathbf{J_f} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \end{bmatrix},$$

  it plays the same role as $\frac{\mathrm{d}f}{\mathrm{d}x}$ in a scalar non-linear equation.
- Given an initial $\mathbf{x}^{(0)}$, the Newton-Raphson iteration $k+1$ is defined as

Systems of non-linear equations

- To obtain a new prediction for the solution $\mathbf{x}^{(k+1)}$, a linear system of equations have to be solved:

$$\mathbf{J_f}(\mathbf{x}^{(k)})\delta\mathbf{x} = -\mathbf{f}(\mathbf{x}^{(k)})$$

, where the matrix $\mathbf{J_f}(\mathbf{x}^{(k)})$ is the coefficient matrix (**A**), the unknown vector is $\delta\mathbf{x}$, and the independent term **b** corresponds to $-\mathbf{f}(\mathbf{x}^{(k)})$

- The non-linear system will have a solution provided that Jacobian is non-singular, $\det(\mathbf{J_f}(\mathbf{x}^{(k)})) \neq 0$ for every k. In this case *LU* decomposition with pivoting can be used as a general method to

Systems of non-linear equations

- A vectorial/matricial function in MATLAB/OCTAVE cannot be defined using `inline` definition.
- A vectorial function must be defined as a MATLAB/OCTAVE object name.m. Example, to define the function in the last example

```
function F=funvect(x)
F(1,1)=x(1)^2+x(2)^2-1;
F(2,1)=sin(x(1))^2+cos(x(2))^2-1;
return
end
```

- In the case of a function defined as *name.m*, it can be passed as an input of other function by using `@` before the name. Example to pass the fun...

Systems of non-linear equations

OCTAVE/MATLAB exercise

- Obtain the Jacobian matrix of the example
- Define two MATLAB/OCTAVE functions defining the function and the Jacobian matrix
- Create a matlab object called *evaluate.m* that uses a vectorial function (n=2) as input and writes as output the value of the function for $x_1 = \pi/4$ and $x_2 = [-1 : 1]$. An additional input should be the number of points to be evaluated
- Run *evaluate.m* on the function of the example and then represent $y = \mathbf{f}(\pi/4, x)$

Systems of non-linear equations

- The Newton-Raphson algorithm in the case of a system of equations is almost the same than for scalar equations.

```
function [solution,niter]=
newton_raphson_system(fun,dfun,x0,epsilon)
error=10*epsilon;
x=x0;
niter=0;
while
(error>epsilon)&(niter<100)
niter=niter+1
value=feval(fun,x);
deriv=feval(dfun,x);
x=x0-deriv\value;
error=norm(x-x0)
x0=x
endwhile
if(niter<100)
solution=x;
else
disp('error')
endif
return
end
```

- A linear system that has to be solved at each iteration. Here is done by the built-in command to solve a linear system: `A\b`.
- The method stops when the *norm* of the diference between

Systems of non-linear equations

OCTAVE/MATLAB exercise

- Write the Newton-Raphson algorithm for systems of equations
- Solve the example

$$\begin{cases} f_1(x_1, x_2) & = x_1^2 + x_2^2 - 1 = & 0 \\ f_2(x_1, x_2) & = \sin^2(x_1) + \cos^2(x_2) - 1 = & 0 \end{cases}$$

  for a given initial value of $\mathbf{x}^{(0)} = [1, 1]$
- Solve the same system for an initial value of $\mathbf{x}^{(0)} = [1, 0]$. What happens?
- Correct the code…

Systems of non-linear equations

As usual, MATLAB/OCTAVE provides built-in procedure to find the roots of a non-linear equation or system of equations, in its easiest form the the procedure is called by `fsolve(fun,x0)` and provides a numerical solution of the problem , if converge!

OCTAVE/MATLAB exercise

Solve the problem of Van der Waals using command *fsolve*