Review TUTORIAL 1 : non linear solvers & LU decomposition

Given a non-linear equation (i.e. $f(x) = \cos x \cdot e^{-x} - x^3$) it can be really hard (or impossible!) to find a zero for it.

I usually need an iterative algorithm in the form:

$$x^{(k)} = \phi(x^{(k-1)}) \rightarrow \text{recurrence iteration}$$

If x^* is the true zero for the function what I want is:

$$\lim_{k \to \infty} x^k = x^*$$

Iterative methods can converge with different "speed":
the 2 most common kind of convergence are:

LINEAR

$$\frac{|x^{(k+1)} - x^*|}{|x^{(k)} - x^*|} \leq C$$

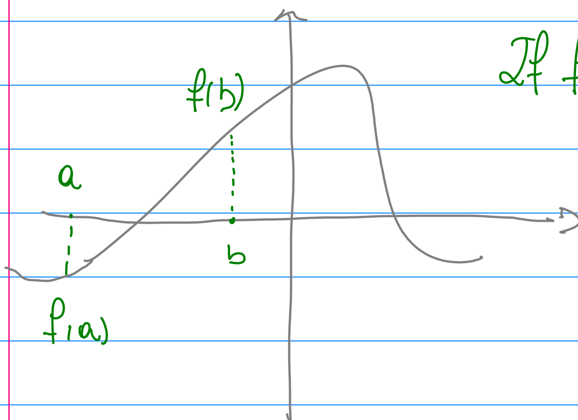
QUADRATIC

$$\frac{|x^{(k+1)} - x^*|}{|x^{(k)} - x^*|^2} \leq C$$

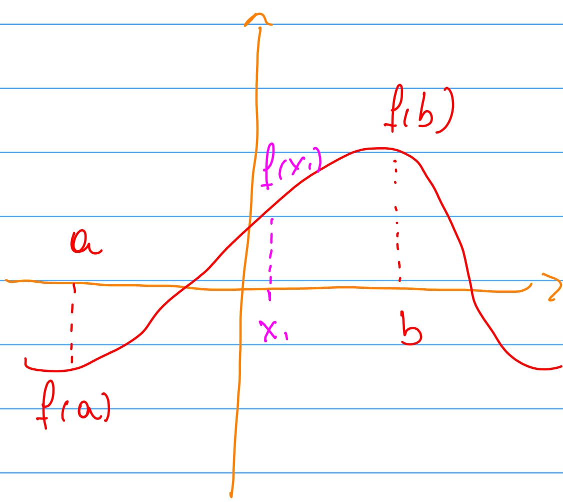
In matlab we solve non linear equation using the command: fzero which take as arguments the function $f(x)$ I want to find the zero of and some initial guess x_0. Alternatively, instead you can pass the interval in which you want to find the zero.

Bisection METHOD

Based on intermediate value theorem



If $f(x) \in C^0$ on $[a,b]$ and $f(a)f(b)<0 \rightarrow$
$\exists x^*$ in $[a,b] \mid f(x^*) = 0.$



pseudo-code:

$x_1 = \frac{(b-a)}{2}$

if $f(x_1) = 0$

break

else check for $f(a)f(x_1)<0$ or $f(b)f(x_1)<0$

if $f(a) \cdot f(x_1) < 0$

$x_2 = \frac{x_1 - a}{2}$

if $f(x_2) = 0$

break

else repeat until $f(x_k) = 0$

Bisection algorithm has linear convergence:

interval at step k $\leftarrow$

$$\frac{(b^{(k)} - a^{(k)})}{(b^{(0)} - a^{(0)})} \leq \frac{1}{2^k}$$

initial interval

Newton's Method

given a function $f(x)$ and its derivative $f'(x)$ the Newton's method is:

$$x^{k+1} = x^k - \underbrace{\frac{f(x^k)}{f'(x^k)}}_{\text{Newton step}}$$

$f(x) = 3x^3 - e^x$

$f'(x) = 9x^2 - e^x$

$x_0 = 3$

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = 3 - \frac{3\cdot 3^3 - e^3}{9\cdot 3^2 - e^3}$$

Newton's method for systems of equations:

Suppose I have a system of n equation, in n variables:

$$\bar{f}(x,y) = \begin{cases} f_1(x,y) \\ f_2(x,y) \end{cases}$$ → system of 2 equations in 2 variables

For an initial guess $\bar{x}_0 = \begin{pmatrix} \bar{x}_{01} \\ \bar{x}_{02} \end{pmatrix}$ the newton's method is:

$$\bar{X}_1 = \bar{X}_0 - \frac{\bar{f}(\bar{x}_0)}{\bar{J}f(\bar{x}_0)}$$

After k steps I can rewrite:

$$\bar{X}_k = \bar{X}_{k-1} - \frac{\bar{f}(\bar{x}_{k-1})}{J\bar{f}(\bar{x}_{k-1})}$$

where Jf is the Jacobian. The Jacobian is a matrix of partial derivatives with entries:

$$[Jf]_{ke} = \frac{\partial f_k}{\partial x_e} = \begin{bmatrix} \partial f_1/\partial x & \partial f_1/\partial y \\ \partial f_2/\partial x & \partial f_2/\partial y \end{bmatrix}$$

Example:

$$\bar{f}(x,y) = \begin{cases} x^3 - xy - 10 = f_1(x,y) \\ y^2 - 4x - 5 = f_2(x,y) \end{cases}$$

$$J\bar{f} = \begin{bmatrix} \partial f_1/\partial x & \partial f_1/\partial y \\ \partial f_2/\partial x & \partial f_2/\partial y \end{bmatrix} = \begin{bmatrix} 3x^2 - y & -x \\ -4 & 2y \end{bmatrix}$$

see multinewton.m to see how it works!

LU decomposition:

Given $Ax = b$, LU decomposition factorize the matrix A into 2 matrix L and U such that:

$$L \cdot U = A$$

In particular, L is lower triangular and U is upper triangular

$$L = \begin{bmatrix} l_{11} & & \emptyset \\ l_{21} & l_{22} & \\ l_{31} & l_{32} & l_{33} \\ \cdots & & \ddots \end{bmatrix} \qquad U = \begin{bmatrix} u_{11} & u_{12} & u_{13}\ldots \\ & u_{22} & \cdots \\ \emptyset & u_{32} & \cdots \end{bmatrix}$$

Now my system is: $L\overbrace{Ux}^{y} = b \rightarrow Ly = b$

I solve for y:

$$y = L^{-1}b$$

this is easy and not expensive to solve using backward Euler

Now, remember that

$$Ux = y$$

I solve for x: $x = U^{-1}y$

this is easy and not expensive to solve using forward Euler

Experiment in Matlab using $[L,U] = lu(A)$ and $[L,U,P] = lu(A)$