Numerical Methods

Roots of Nonlinear Equations

- Advanced Material

Course 2022-2023

[DΣIM]

The Regula-Falsi Method

- The bisection method has a low convergence order, due to the use only of partial information on the function as we use only the signs of $f(x_n)$ and not their values.
- If we have, for instance that $f(a) = -1 < 0 < 1000 = f(b)$ we can expect that the root is near the value $x = a$, but we take, however, the value $x_n = (a + b)/2$ as the next iterate.
- In the regula-falsi method we change the way that we select the next iterate, using also the function values.

The Regula-Falsi Method

- We use the same hypothesis as in the bisection method, namely that f is a continuous function and that $f(a)f(b) < 1$. Then, at least one root is to be found in the interval (a, b).
- Instead of using the midpoint, however, we use the straight line through the points $(a, f(a))$ and $\big(b, f(b)\big)$, written as:

$$\frac{y - f(b)}{f(b) - f(a)} = \frac{x - b}{x - a}$$

The Regula-Falsi Method

- Now, solving for $y = 0$, we obtain the solution:

$$w = b - f(b)\frac{b-a}{f(b)-f(a)}$$

- Then, using this new value, we select the interval (a, w) or (w, b) containing the root and repeat the iteration.
- This method is also kwon as the method of False Position.
- We cannot say with certainty that this method will outperform always the bisection method. This method, however, has the advantage of having a computable error estimate.

The Regula-Falsi Method

- Before discussing the convergence of the method, we will need to prove the following result.
- Let $f \in C^2[a,b]$ with $f(a)f(b) < 0$. If f'' has no roots within the interval (a,b), then one of the sequences $\{a_n\}$ or $\{b_n\}$ indicating the extremes of the iteration intervals will remain constant.
- If f'' is continuous and has no roots, then it must have the same sign within the interval (a,b).

The Regula-Falsi Method

- This means that the iteration will eventually end in one of the following four configurations:
- 1. $f'' > 0$ in (a, b); $f(a) < 0 < f(b)$
- 2. $f'' > 0$ in (a, b); $f(a) > 0 > f(b)$
- 3. $f'' < 0$ in (a, b); $f(a) > 0 > f(b)$
- 4. $f'' < 0$ in (a, b); $f(a) < 0 < f(b)$
- We are going to prove the result for the first case, as the rest are similar.

The Regula-Falsi Method

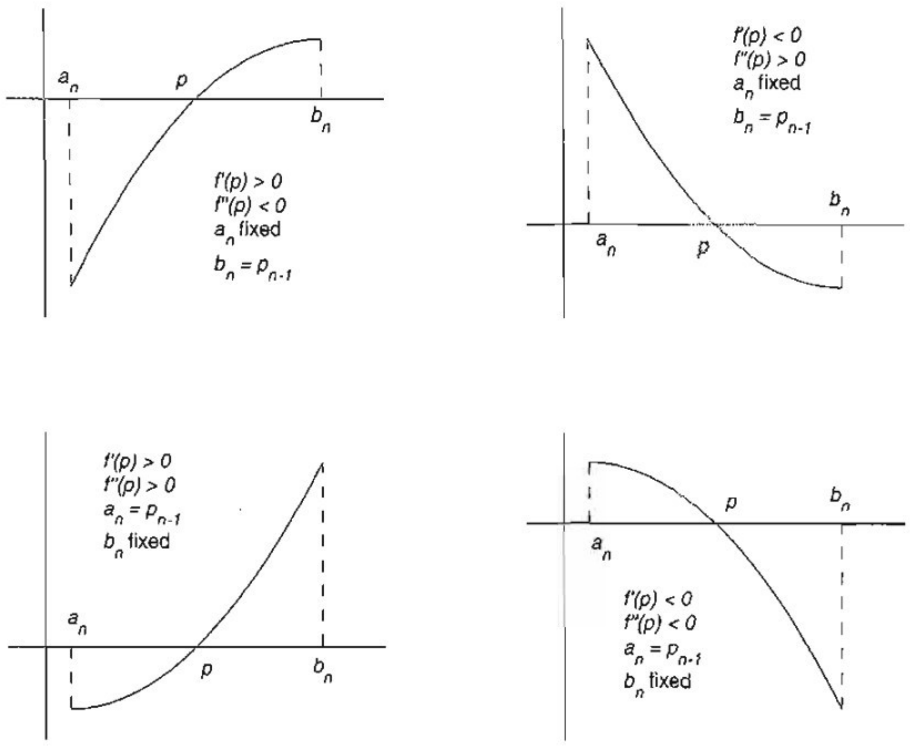


Figure 2.5 Eventual configurations for the method of false position.

Numerical Methods

The Regula-Falsi Method

- In the first case, the curve f, will lie entirely below the straight line between the points $(a, f(a))$ and $\big(b, f(b)\big)$, and we have:

$$f(x) < f(b) + (x-b)\frac{f(b)-f(a)}{b-a}, \quad x \in (a,b)$$

- If the next iteration point is $w \in (a, b)$, then we have $f(w) < 0$ or $f(a) < 0$, from the hypothesis. The next interval will be (w, b), but this gives a situation entirely identical as the initial one, and we can conclude that b will not change.

The Regula-Falsi Method

- We can prove now that the method is convergent. The error at step n, $e_n = x_n - \alpha$ can be written as:

$$x_n - \alpha = b_n - \alpha - f(b_n)\frac{b_n - a_n}{f(b_n) - f(a_n)}$$

- If we approximate the values $f(a_n)$ and $f(b_n)$ using a second-degree Taylor series using the fact $f(\alpha) = 0$:

$$f(a_n) \approx f'(\alpha)(a_n - \alpha) + \frac{f''(\alpha)}{2}(a_n - \alpha)^2$$

$$f(b_n) \approx f'(\alpha)(b_n - \alpha) + \frac{f''(\alpha)}{2}(b_n - \alpha)^2$$

The Regula-Falsi Method

- Then, the term $f(b_n) - f(a_n)$ can be approximated as:

$$f(b_n) - f(a_n) \approx f'(\alpha)(b_n - a_n) + \frac{f''(\alpha)}{2}\left[(b_n - \alpha)^2 - (a_n - \alpha)^2\right]$$
$$= (b_n - a_n)\left[f'(\alpha) + \frac{f''(\alpha)}{2}(b_n + a_n - 2\alpha)\right].$$

- Using these relations in the error term we obtain:

$$x_n - \alpha = \left(b_n - \alpha\right)\left[1 - \frac{f'(\alpha) + \frac{f''(\alpha)}{2}(b_n - \alpha)}{f'(\alpha) + \frac{f''(\alpha)}{2}\left(b_n + a_n - 2\alpha\right)}\right]$$
$$= (b_n - \alpha)(a_n - \alpha)\frac{f''(\alpha)}{2f'(\alpha) + f''(\alpha)\left(b_n + a_n - 2\alpha\right)}$$

The Regula-Falsi Method

- This means that the error term e_n can be written as:

$$e_n \approx \lambda e_{n-1},$$

- Where:

$$\lambda = \frac{lf''(\alpha)}{2f'(\alpha) + lf''(\alpha)}$$

- And

$$l = \begin{cases} a_n - \alpha, & \text{when } a_n \text{ is fixed} \\ b_n - \alpha, & \text{when } b_n \text{ is fixed} \end{cases}$$

The Regula-Falsi Method

- It remains only to show that $|\lambda| < 1$ to prove that the method is convergent. We can prove this for the first case, being the rest similar.
- Suppose that a_n is fixed. Then $a_n - \alpha < 0$. If we are in the first configuration $f''(\alpha) < 0$ and $(a_n - \alpha)f''(\alpha) > 0$. Since $f'(\alpha) > 0$, it follows that:

$$2f'(\alpha) + (a_n - \alpha)f''(\alpha) > (a_n - \alpha)f''(\alpha)$$

- And

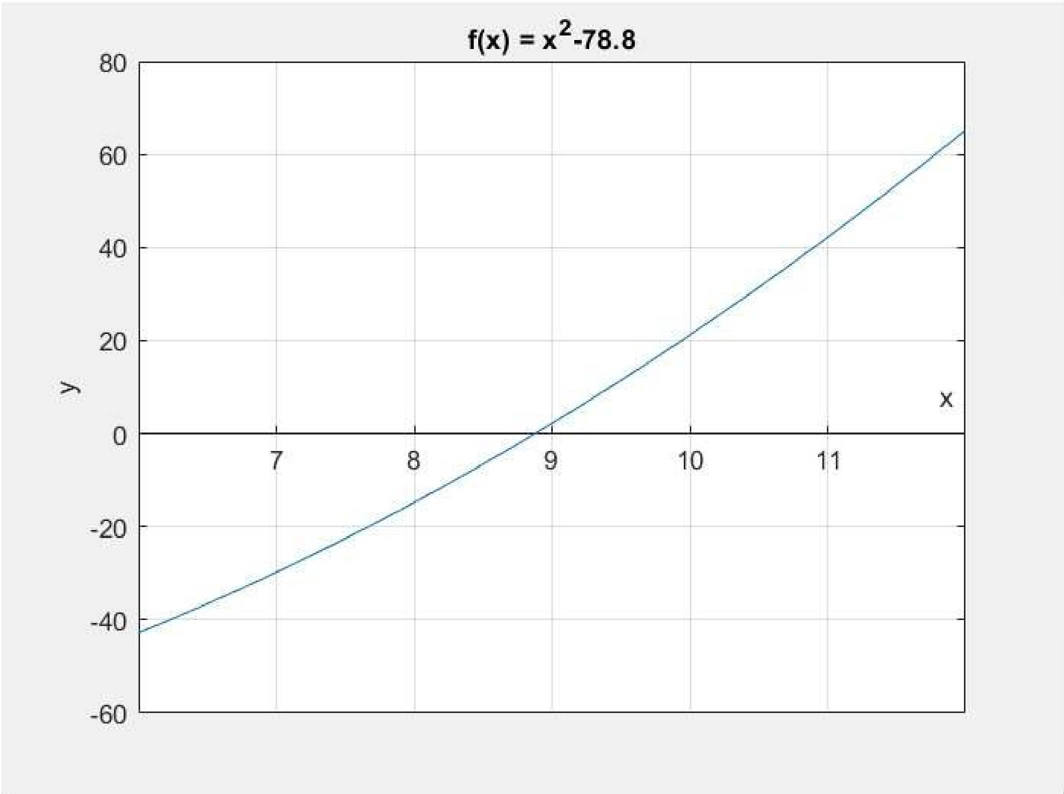
$$0 < \frac{(a_n - \alpha)f''(\alpha)}{2f'(\alpha) + (a_n - \alpha)f''(\alpha)} = \lambda < 1.$$

The Regula-Falsi Method

```matlab
% Solve the equation and find the positive root of
%   x^2 - 78.8 = 0
% Using the Regula Falsi and the Bisection Methods

clear

% Basic Parameters
delta = 10^-6;
tol = 10^-6;
MaxIter = 50;

% Define the basic functions
fm = @(x) x.^2 - 78.8;

% Plot the function
xp = linspace(6,12,400);
yp = fm(xp);
plot (xp,yp)
xlabel('x')
ylabel('y')
ax = gca;
ax.XAxisLocation = 'origin';
ax.YAxisLocation = 'origin';
title (' f(x) = x^2-78.8 ')
grid on

% Solve the nonlinear equation using the Regula Falsi x=sqrt(78.8)

a= 6;
b= 12;

[xroot,er,xx,iter] = Falsep(fm,a,b,delta,tol,MaxIter);

fprintf ('Results for Regula Falsi \n')
for k = 1:iter
    yx = fm(xx(k));
    fprintf ( ' k = %2d, x = %10.8f,  f(x) = % 10.8f \n', k, xx(k), yx)
end

fprintf (' xroot : %10.8f, er =  %10.8f \n\n', xroot, er)

% Solve the nonlinear equation using the Bisection x=sqrt(78.8)

a= 6;
b= 12;

[xroot,er,xx,iter] = Bisection(fm,a,b,tol);

fprintf ('Results for Bisection \n')
for k = 1:iter
    yx = fm(xx(k));
    fprintf ( ' k = %2d, x = %10.8f,  f(x) = % 10.8f \n', k, xx(k), yx)
end

fprintf (' xroot : %10.8f, er =  %10.8f \n\n', xroot, er)
```

Numerical Methods

The Regula-Falsi Method



Course 2022-2023

[DΣIM]

The Regula-Falsi Method

```matlab
function [xroot,err,xx,iter] = Falsep(f,a,b,TolX,TolF,MaxIter,varargin)
% Falsep: False Postion Method for solving f(x)=0.
% [xroot,err,xx,iter] = Falsep (f,a,b,TolX,TolF,MaxIter,varargin)
% Uses the False Position method to find the root of f(x)=0
%

% Input :
%   f = Function to be given as a function handle or an M-file name
%   a/b = Initial left/right point of the solution interval
%   TolX = Upperbound of error(max(|x(k)-a|,|b-x(k)|))
%   TolF = Upperbound of abs(f(x))
%   MaxIter = Maximum # of iterations%

% Output:
%   xroot = Point which the algorithm has reached
%   err = Max(x(last)-a|,|b-x(last)|)
%   xx = History of x
%   iter = Number of iterations%

% Variable Check
if nargin<3, error('at least 3 input arguments required'), end
if nargin<4 || isempty(TolX), TolX=0.0001; end
if nargin<5 || isempty(TolF), TolF=0.0001; end
if nargin<6 || isempty(MaxIter), MaxIter = 50; end

% Check that the Interval contains a Solution
fa = f(a,varargin{:});
fb = f(b,varargin{:});
if fa*fb>0, error('We must have f(a)f(b)<0!'); end

% Preallocate Memory for loop array.
xx = zeros(1,MaxIter);

% We need the first iteration to compute the error
xx(1) = (a*fb-b*fa)/(fb-fa);
fx = f(xx(1),varargin{:});
if fx*fa > 0
    a = xx(1);
    fa = fx;
else
    b = xx(1);
    fb = fx;
end

for iter = 2:MaxIter
    xx(iter)= (a*fb-b*fa)/(fb-fa);
    fx= f(xx(iter), varargin{:});
    err= max(abs(xx(iter)-xx(iter-1)));
    if abs(fx)< TolF || err < TolX, break;
    elseif fx*fa>0
        a=xx(iter);
        fa=fx;
    else
        b=xx(iter);
        fb=fx;
    end
end
if (iter >= MaxIter)
    xroot = NaN;
else
    xroot = xx(iter);
end
```

Numerical Methods

Course 2022-2023

Acceleration of Convergence

- It is rare to have quadratic convergence. However, the convergence of a linear sequence can be accelerated using ***Aitken's method***.
- Suppose $\{p_n\}_{n=0}^{\infty}$ is a linearly convergent sequence with limit p. Assume that the signs of the differences $p_n - p, p_{n+1} - p$, and $p_{n+2} - p$ agree and that n is sufficiently large that

$$\frac{p_{n+1} - p}{p_n - p} \approx \frac{p_{n+2} - p}{p_{n+1} - p}$$

Acceleration of Convergence

- Then

$$\left(p_{n+1}-p\right)^2 \approx \left(p_{n+2}-p\right)\left(p_n-p\right)$$

- So

$$p_{n+1}^2-2p_{n+1}p+p^2 \approx p_{n+2}p_n-\left(p_n+p_{n+2}\right)p+p^2$$

- And

$$\left(p_{n+2}+p_n-2p_{n+1}\right)p \approx p_{n+2}p_n-p_{n+1}^2$$

- Finally, solving for *p*

$$p \approx \frac{p_{n+2}p_n-p_{n+1}^2}{p_{n+2}-2p_{n+1}+p_n}$$

Acceleration of Convergence

- This equation can be rewritten as follows:

$$p \approx \frac{p_n^2 + p_n p_{n+2} - 2p_n p_{n+1} + 2p_n p_{n+1} - p_n^2 - p_{n+1}^2}{p_{n+2} - 2p_{n+1} + p_n}$$

$$= \frac{\left(p_n^2 + p_n p_{n+2} - 2p_n p_{n+1}\right) - \left(p_n^2 - 2p_n p_{n+1} + p_{n+1}^2\right)}{p_{n+2} - 2p_{n+1} + p_n}$$

$$= p_n - \frac{\left(p_{n+1} - p_n\right)^2}{p_{n+2} - 2p_{n+1} + p_n}$$

Acceleration of Convergence

- If we introduce the forward difference Δp_n defined by:

$$\Delta p_n = p_{n+1} - p_n, \qquad \text{for } n \geq 0$$

- And use the recurrence relation

$$\Delta^k p_n = \Delta\left(\Delta^{k-1} p_n\right), \qquad \text{for k} \geq 2$$

- We can write:

$$\Delta^2 p_n = \Delta\left(p_{n+1} - p_n\right) = \Delta p_{n+1} - \Delta p_n = \left(p_{n+2} - p_{n+1}\right)\left(p_{n+1} - p_n\right)$$

- And so

$$\Delta^2 p_n = p_{n+2} - 2p_{n+1} + p_n$$

Acceleration of Convergence

- Using this expression, we can introduce the new sequence defined by:

$$\hat{p}_n = p_n - \frac{(\Delta p_n)^2}{\Delta^2 p_n} = p_n - \frac{(p_{n+1} - p_n)^2}{p_{n+2} - 2p_{n+1} + p_n}, \quad n \geq 0$$

- This expression is known as ***Aitken's extrapolation formula***.
- If $\{p_n\}_{n=0}^{\infty}$ is a linearly convergent sequence, then the new sequence $\{\hat{p}_n\}_{n=0}^{\infty}$ will converge faster in the sense that

$$\lim_{n\to\infty} \frac{\hat{p}_n - p}{p_n - p} = 0$$

Steffensen's Method

- By applying the Aitken's Δ^2 method to a linearly convergent sequence obtained from fixed-point iteration we can accelerate the convergence to quadratic. This procedure is known as ***Steffensen's method***.
- This is a slight modification of Aiken's method. In Aitken method we compute the values as

$$p_0, \quad p_1 = g(p_0), \quad p_2 = g(p_1), \quad \hat{p}_0 = \Delta^2 p_0,$$
$$p_3 = g(p_2), \quad \hat{p}_1 = \Delta^2 p_1, \quad \ldots$$

Steffensen's Method

- Steffensen's method constructs the same first four terms, p_0, p_1, p_2 and $\hat{p}_0$. However, at this step it assumes that $\hat{p}_0$ is a better approximation to p than is p_2 and applies fixed point iteration to $\hat{p}_0$ instead of p_2. Then, we generate the sequence

$$p_0, \quad p_1 = g(p_0), \quad p_2 = g(p_1), \quad \hat{p}_0 = \Delta^2 p_0,$$
$$\hat{p}_1 = g(\hat{p}_0), \quad \hat{p}_2 = g\left(\hat{p}_1\right), \quad \hat{p}_3 = \Delta^2 \hat{p}_0 \ldots$$

- Every third terms is generated using Aitken's formula, while the rest are computed using fixed-point iteration on the previous term.

Acceleration of Convergence

- Let us write explicit formulas for the Aitken's method and for the Steffenson's method in the case of fixed-point iteration.
- Using fixed point iteration, we can write:

$$\alpha - x_n = g(\alpha) - g(x_{n-1}) = g'(\xi_{n-1})(\alpha - x_{n-1}), \qquad \xi_{n-1} \in \left(\alpha, x_{n-1}\right)$$

- Now consider

$$\alpha - x_n = \left(\alpha - x_{n-1}\right) + \left(x_{n-1} - x_n\right) = \frac{1}{g'(\xi_{n-1})}\left(\alpha - x_n\right) + \left(x_{n-1} - x_n\right)$$

Acceleration of Convergence

- Solving for α we obtain:

$$\alpha = x_n + \frac{g'(\xi_{n-1})}{1 - g'(\xi_{n-1})}(x_n - x_{n-1})$$

- This expression gives α in terms of x_n, x_{n-1}, which are computable quantities and $g'(\xi_{n-1})$ which is not. This value, however, can be estimated as follows. Since we assume that $x_n \to \alpha$ and that $g'(\xi_n) \to g'(\alpha)$ we have

$$g'(\xi_{n-1}) \approx g'(\alpha)$$

Acceleration of Convergence

- On the other hand, consider the ratio:

$$\gamma_n = \frac{x_{n-1} - x_n}{x_{n-2} - x_{n-1}} = \frac{(\alpha - x_{n-1}) - (\alpha - x_n)}{(\alpha - x_{n-2}) - (\alpha - x_{n-1})}$$

$$= \frac{(\alpha - x_{n-1}) - g'(\xi_{n-1})(\alpha - x_{n-1})}{(\alpha - x_{n-1}) / g'(\xi_{n-2}) - (\alpha - x_{n-1})}$$

$$= \frac{1 - g'(\xi_{n-1})}{1/g'(\xi_{n-2}) - 1} = g'(\xi_{n-2}) \frac{1 - g'(\xi_{n-1})}{1 - g'(\xi_{n-2})} \to g'(\alpha)$$

- Thus, $\gamma_n \approx g'(\alpha)$ which is computable. We can use:

$$\gamma_n \approx g'(\xi_{n-1})$$

Acceleration of Convergence

- To obtain a computable estimation of α as:

$$\alpha \approx x_n + \frac{\gamma_n}{1-\gamma_n}(x_n - x_{n-1})$$

- Which gives Aitken's extrapolation formula for fixed point iteration:

$$\hat{x}_n = x_n + \frac{\gamma_n}{1-\gamma_n}(x_n - x_{n-1})$$

- Where:

$$\gamma_n = \frac{x_{n-1} - x_n}{x_{n-2} - x_{n-1}}$$

Acceleration of convergence

- For Steffenson's method we will use essentially the same expression, except that we compute the three first terms using fixed point iteration and then use the Aitken formula to compute the next point. From now on we continue to use the fixed-point iteration and use Aitken formula with the new values, that is we use the ratio:

$$\hat{\gamma}_n = \frac{\hat{x}_{n-1} - \hat{x}_n}{\hat{x}_{n-2} - \hat{x}_{n-1}}$$

- Note the potential problems with cancellation of terms in the computation of the ratio γ_n

Acceleration of Convergence

- Aitken’s Extrapolation

```
% Aitken's Extrapolation Example
% Iterate the function
%       g(x) = 1/2 * exp(-x)

% Function
g = @(x) 0.5.*exp(-x);

% Basic Constants.
MaxIter = 100;
error = 1e-8;

% Preallocate Memory for loop arrays.
xs = zeros(1,MaxIter);
xa = zeros(1,MaxIter);

x0 = 0.0;
xs(1) = g(x0);
xs(2) = g(xs(1));
xx(1) = xs(1);
xx(2) = xs(2);

```

```
% Main loop

iter = 2;
for k=3:MaxIter
    iter = iter + 1;
    % Compute the standard iteration
    xs(k) = g(xs(k-1));
    % Aitken extrapolation.
    if abs(xs(k-1) - xs(k-2)) >= 1e-20
        gamma = (xs(k) - xs(k-1) ) / ( xs(k-1) - xs(k-2));
    else
        gamma = 0.0d0;
    end
    xx(k) = xs(k-1) + gamma*(xs(k-1) - xs(k-2) ) / (1.0 - gamma);
    if(abs(xx(k) - xx(k-1)) < error)
        break;
    end
end

for k = 1:iter
    fprintf ( [' k = %2d, xs = %10.8f, gs = % 10.8f,  xx = %10.8f', ...
        'gxx = % 10.8f \n'], k, xs(k), g(xs(k)), xx(k), g(xx(k)) )
end

```

Acceleration of Convergence

```
>> AitkenExtrapolation
 k =  1, xs = 0.50000000, gs =  0.30326533,  xx = 0.50000000, gxx =  0.30326533
 k =  2, xs = 0.30326533, gs =  0.36920157,  xx = 0.30326533, gxx =  0.36920157
 k =  3, xs = 0.36920157, gs =  0.34564303,  xx = 0.35265011, gxx =  0.35141153
 k =  4, xs = 0.34564303, gs =  0.35388255,  xx = 0.35184456, gxx =  0.35169472
 k =  5, xs = 0.35388255, gs =  0.35097870,  xx = 0.35174752, gxx =  0.35172885
 k =  6, xs = 0.35097870, gs =  0.35199937,  xx = 0.35173542, gxx =  0.35173311
 k =  7, xs = 0.35199937, gs =  0.35164028,  xx = 0.35173392, gxx =  0.35173364
 k =  8, xs = 0.35164028, gs =  0.35176658,  xx = 0.35173374, gxx =  0.35173370
 k =  9, xs = 0.35176658, gs =  0.35172215,  xx = 0.35173371, gxx =  0.35173371
 k = 10, xs = 0.35172215, gs =  0.35173778,  xx = 0.35173371, gxx =  0.35173371
```

Acceleration of Convergence

- Steffensen's Extrapolation

```
% Steffensen's Extrapolation Example
% Iterate the function
%       g(x) = 1/2 * exp(-x)

% Function
g = @(x) 0.5.*exp(-x);

% Basic Constants.
MaxIter = 100;
error = 1e-8;
% Preallocate Memory for loop arrays.
xs = zeros(1,MaxIter);
xx = zeros(1,MaxIter);

x0 = 0.0;
xs(1) = x0;
```

```
% Main loop

xx(1) = x0;
for k=2:MaxIter
    xs(k) = g(xs(k-1));
    x1 = g(xx(k-1));
    x2 = g(x1);
    if(abs(x1-xx(k-1)) > 1e-20)
        gamma = (x2-x1) / (x1 - xx(k-1));
    else
        gamma = 0.0;
    end
    xnew = x2 + gamma*(x2 - x1) / ( 1- gamma);
    xx(k) = xnew;
    if (abs(xnew-x2) < error)
      break;
    end
end

for k = 1:k
    fprintf ( [' k = %2d, xs = %10.8f, gs = % 10.8f,  xx = %10.8f, ',  ...
        'gxx = % 10.8f \n'], k, xs(k), g(xs(k)), xx(k), g(xx(k)) )
end
```

Acceleration of Convergence

```
>> SteffensenExtrapolation
 k =  1, xs = 0.00000000, gs =  0.50000000,  xx = 0.00000000, gxx =  0.50000000
 k =  2, xs = 0.50000000, gs =  0.30326533,  xx = 0.35881665, gxx =  0.34925121
 k =  3, xs = 0.30326533, gs =  0.36920157,  xx = 0.35173600, gxx =  0.35173291
 k =  4, xs = 0.36920157, gs =  0.34564303,  xx = 0.35173371, gxx =  0.35173371
 k =  5, xs = 0.34564303, gs =  0.35388255,  xx = 0.35173371, gxx =  0.35173371
fx >>
```

Systems of Nonlinear Equations

- Newton method for nonlinear equations can be readily adapted for the case of systems on nonlinear equations of the form:

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

- The strategy is again to linearize and solve the simpler system

Systems of Nonlinear Equations

- Let us illustrate this procedure for a system of two nonlinear equations

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

- Supposing that (x_1, x_2) is an approximate solution, we want to compute corrections h_1 and h_2 so that $(x_1 + h_1, x_2 + h_2)$ is a better approximate solution.

Systems of Nonlinear Equations

- If we linearize the system of equation using the Taylor expansion of each function, we obtain:

$$\begin{cases} 0 = f_1(x_1 + h_1, x_2 + h_2) \approx f_1(x_1, x_2) + h_1 \dfrac{\partial f_1}{\partial x_1} + h_2 \dfrac{\partial f_1}{\partial x_2} \\ 0 = f_2(x_1 + h_1, x_2 + h_2) \approx f_2(x_1, x_2) + h_1 \dfrac{\partial f_2}{\partial x_1} + h_2 \dfrac{\partial f_2}{\partial x_2} \end{cases}$$

- Where the partial derivatives are evaluated at the point (x_1, x_2). This is a linear system of equations to determine h_1 and h_2

Systems of Nonlinear Equations

- These equations can be written in matrix form:

$$\begin{pmatrix} \dfrac{\partial f_1}{\partial x_1} & \dfrac{\partial f_2}{\partial x_1} \\ \dfrac{\partial f_2}{\partial x_1} & \dfrac{\partial f_2}{\partial x_2} \end{pmatrix} \begin{pmatrix} h_1 \\ h_2 \end{pmatrix} = -\begin{pmatrix} f_1 \\ f_2 \end{pmatrix}$$

- Solving this system of equation, we will obtain the vector of corrections that can be used to obtain a better approximation to the solution of the system.

Systems of Nonlinear Equations

- The coefficient matrix is the Jacobian matrix of f_1 and f_2:

$$J = \begin{pmatrix} \dfrac{\partial f_1}{\partial x_1} & \dfrac{\partial f_2}{\partial x_1} \\ \dfrac{\partial f_2}{\partial x_1} & \dfrac{\partial f_2}{\partial x_2} \end{pmatrix}$$

- To solve the system J must be nonsingular and then:

$$\begin{pmatrix} h_1 \\ h_2 \end{pmatrix} = -J^{-1} \begin{pmatrix} f_1(x_1, x_2) \\ f_2(x_1, x_2) \end{pmatrix}$$

Systems of Nonlinear Equations

- Hence, Newton's method for two nonlinear equations in two variables is:

$$\begin{pmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \end{pmatrix} = \begin{pmatrix} x_1^{(k)} \\ x_2^{(k)} \end{pmatrix} + \begin{pmatrix} h_1^{(k)} \\ h_2^{(k)} \end{pmatrix}$$

- Where we need to solve the linear system:

$$\mathbf{J}\begin{pmatrix} h_1^{(k)} \\ h_2^{(k)} \end{pmatrix} = -\begin{pmatrix} f_1\left(x_1^{(k)}, x_2^{(k)}\right) \\ f_2\left(x_1^{(k)}, x_2^{(k)}\right) \end{pmatrix}$$

- At each step.

Systems of Nonlinear Equations

- The generalization to more variables is straightforward. The system of equations

$$f_i(x_1, x_2, \ldots, x_n) = 0 \quad (1 \le i \le n)$$

- Can be expressed simply as

$$\mathbf{F(X) = 0}$$

- With $X = (x_1, x_2, \ldots, x_n)^T, F = (f_1, f_2, \ldots, f_n)^T$

Systems of Nonlinear Equations

- Then

$$\mathbf{0} = \mathbf{F}(\mathbf{X}+\mathbf{H}) \approx \mathbf{F}(\mathbf{X}) + \mathbf{F}'(\mathbf{X})\mathbf{H}$$

- Where $\boldsymbol{H} = (h_1, h_2, \dots, h_n)^T$ is the correction vector and $\boldsymbol{F}'(\boldsymbol{X})$ represents the $n \times n$ Jacobian matrix with elements $\partial f_i / \partial x_j$.
- Theoretically have

$$\mathbf{H} = -\mathbf{F}'(\mathbf{X})^{-1}\mathbf{F}(\mathbf{X})$$

Systems of Nonlinear Equations

- However, the correction vector $\boldsymbol{H}$ can be computed using gaussian elimination, which is not so computationally expensive as the computation of the inverse $\boldsymbol{F}'(\boldsymbol{X})^{-1}$. We can solve the linear system:

$$\mathbf{F}'(\mathbf{X}^{(k)})\mathbf{H}^{(k)} = -\mathbf{F}(\mathbf{X}^{(k)})$$

- And then add the correction vector

$$\mathbf{X}^{(k+1)} = \mathbf{X}^{(k)} + \mathbf{H}^{(k)}$$

- Until some desired precision is reached.

Systems of Nonlinear Equations

- If the number of equations in the system is high, it can be very difficult to compute and evaluate all the derivatives in Jacobian matrix. In this case it may be preferable to use a finite difference approximation for the derivatives:

$$\frac{\partial f_i}{\partial x_j} \approx \frac{f_i(x_1,\ldots,x_{j-1},x_j+h,x_{j+1},\ldots,x_n) - f_i(x_1,\ldots,x_{j-1},x_j,x_{j+1},\ldots,x_n)}{h}$$

- This will allow to compute faster the Jacobian matrix but can be unstable for small values of h due to cancellation of terms.

Systems of Nonlinear Equations

- Another variant to increase the computation speed can be to approximate all the derivatives in future iterations with the derivatives at a fixed iterate $\boldsymbol{X}_n$ when we already have the condition $\|\boldsymbol{X}_{n+1} - \boldsymbol{X}_n\| < \epsilon$ for some small value of the tolerance ϵ. In this case the iterations will follow as:

$$\mathbf{F}'(\mathbf{X}^{(N)})\mathbf{H}^{(k)} = -\mathbf{F}(\mathbf{X}^{(k)}), \quad k > N$$

- This can give a slower rate of convergence but can be a good trade off in the case of a big number of equations.

Systems of Nonlinear Equations

- As an example, we can solve the following nonlinear system

$$\begin{cases} x^3 + 3y^2 - 21 = 0 \\ x^2 + 2y + 2 = 0 \end{cases}$$

- The Jacobian is:

$$J = \begin{pmatrix} 3x^2 & 6y \\ 2x & 2 \end{pmatrix}$$

Systems of Nonlinear Equations

```
% Test newt_sys

% Basic Constants

tol = 10^-8;
MaxIter = 40;

x0 = [1, -1]';

[xf, xx, iter, err] = newton_sys('f1', 'df1', x0, tol, MaxIter);

for k = 1:iter
    fprintf ( ' k = %2d, x = %10.8f,  y = % 10.8f \n', k, xx(k,1), xx(k,2))
end

fprintf (' xf =  %10.8f, yf = %10.8f, err =  %10.8f \n\n', xf, err)
```

```
function f=f1(X)
    x = X(1);
    y = X(2);
    f = [x.^3 + 3*y.^2 - 21; x.^2+2*y+2];
end
```

```
function df = df1(X)
    x = X(1);
    y = X(2);
    df = [3*x.^2, 6*y; 2*x, 2];
end
```

Systems of Nonlinear Equations

```matlab
function [xf, xx, iter, err] = newton_sys(f,jf,x0,TolX,MaxIter)

% Newton_Sys : Newton method for solving systems of nonlinear equations f(x)=0.
% [xf,xx,iter,err] = newton_sys (f,jf,x0,x1,TolX,MaxIter)
% uses Newton method to find the root of the vector nonlinear equation f(x)=0

% Input:
%   f = Vector function as a function handle or an M-file name
%   jf = Jacobian Matrix as functon handle or an M-file name
%   x0 = Initial guess of the (vector) solution
%   TolX = Upper limit of the norm |x(k)-x(k-1)|
%   MaxIter = Maximum # of iteration

% Output:
%   xf = Vector solution which the algorithm has reached
%   xx = History of x
%   iter = Number of Iterations
%   error = final error of the iterations

% Variable Check

if nargin<3, error('at least 3 input arguments required'), end
if nargin<4 || isempty(TolX), TolX=0.00001; end
if nargin<5 || isempty(MaxIter), MaxIter = 50; end

% Preallocate Memory for loop array.
xx = zeros(MaxIter, 2);


iter=1;
xx(iter,:) = x0';
while(iter<=MaxIter)
    y=-feval(jf,x0)\feval(f,x0);
    xn=x0+y;
    err= max(abs(xn-x0));
    if (err <= TolX)
        break;
    else
        x0=xn;
    end
    iter=iter+1;
    xx(iter,:) = xn';
end
if (iter >= MaxIter)
    disp(' Newton’s method does not converge')
    xf = NaN;
else
    xf = xn';
end
```

Systems of Nonlinear Equations

- Using the Newton method, we obtain:

```
>> test_newton_sys
 k =  1, x = 1.00000000,  y = -1.00000000
 k =  2, x = 2.55555556,  y = -3.05555556
 k =  3, x = 1.86504913,  y = -2.50080458
 k =  4, x = 1.66133689,  y = -2.35927080
 k =  5, x = 1.64317336,  y = -2.34984440
 k =  6, x = 1.64303806,  y = -2.34978702
 xf =  1.64303805, yf = -2.34978702, err =  0.00000001
```