Numerical Methods

Roots of Nonlinear Equations

- Iterative Procedures

2022-2023

[DΣIM]

Example

- The behavior of normal gasses is often described by means of the ideal gas equation:

$$P = \frac{RT}{V}$$

- R is the universal constant for the gasses. Knowing the temperature and pressure its easy to obtain the volume:

$$V = \frac{RT}{P}$$

Example

- But real gasses are not ideal. For real gasses we have more precise rules like the Beattie-Bridgeman equation:

$$P = \frac{RT}{V} + \frac{\beta}{V^2} + \frac{\gamma}{V^3} + \frac{\delta}{V^4}$$

- This is a better approximation than the ideal gas equation and recovers the ideal law when β, γ, and δ are zero
- In this case, it is not easy to obtain V if we know P and T

Example

- Even if we know the values of P and T, we can not make an algebraic manipulation of the equation to obtain the value of V.
- In this chapter we will explore algorithms that can give a ***numerical solution*** of this kind of problems.
- Many of these algorithms were known much before the use of computers

Statement of the Problem

- We want to solve equations of the form:

$$f(x) = 0$$

- Where f, is a real valued and continuous function. This means that we want to find a value of the independent variable $x = \alpha$ for which we have

$$f(\alpha) = 0$$

- The value α is know as the ***solution*** of the equation

Statement of the Problem

- The function f, could be any function. Some of these equations can be solved algebraically. For instance, the equation

$$x^2 - x - 2 = 0$$

- Could be solved algebraically, but we are interested in the most general cases. Normally we need to deal with ***nonlinear equations*** that can not be solved in closed form.

Statement of the Problem

- The following equations are all nonlinear:

$$x^2 - 2xe^{-x} + e^{-2x} = 0$$

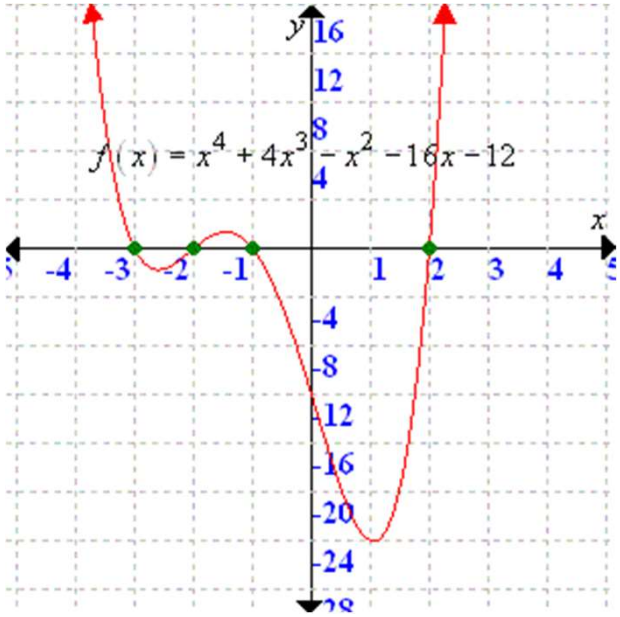
$$\sin(x) + e^{-x} = 0$$

$$\cos(3x) = 0$$

- How could we solve this kind of problems?

Statement of the Problem

- In a geometrical sense, solving this kind of equations means to find the point where these functions intersect the *x* axis



[DΣIM]

Iterative Methods

- The methods used normally to solve these kind of problems are ***iterative methods***. This means that we want to generate a sequence of values $x^{(k)}$, such that

$$\lim_{k\to\infty} x^{(k)} = \alpha$$

- And verifying that:

$$f(\alpha) = 0$$

- If the method can generate such a sequence, we say that the method is ***convergent***.

Convergence Rate

- We will need to compare also the ***speed of convergence*** of different methods. We say that the sequence $\{x^{(k)}\}$ converges to α with ***order*** $p \geq 1$ if:

$$\exists C > 0 \ : \ \frac{\left|x^{(k+1)} - \alpha\right|}{\left|x^{(k)} - \alpha\right|^{p}} \leq C, \ \forall k \geq k_0$$

- The constant C is called the ***convergence factor*** of the method.

Starting Point

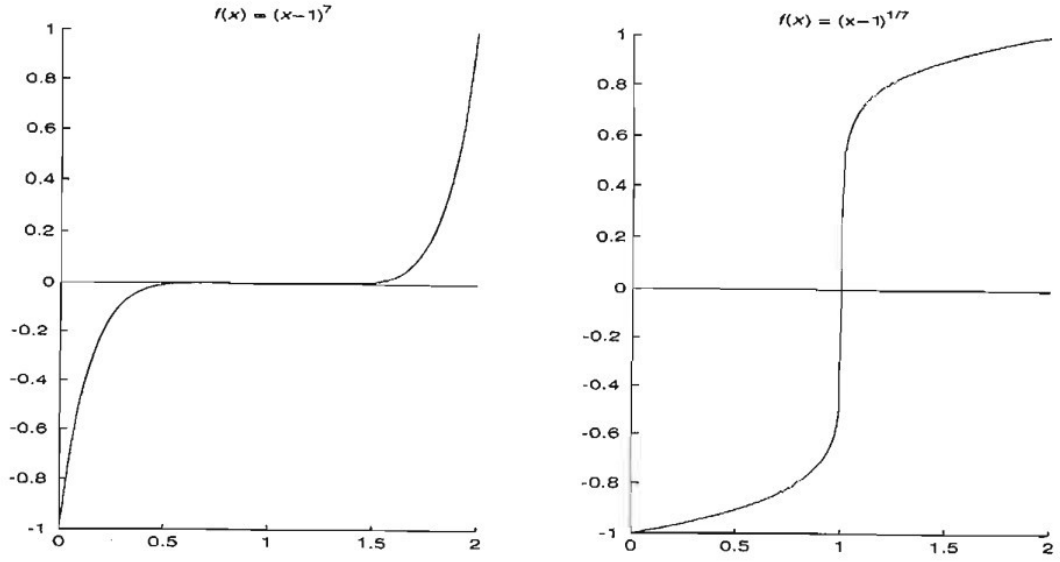
- To start the iterations, we will need a first approximation x_0. This first value can be critical for the convergence. In this case we say the method is ***locally convergent***.
- Some methods, however, are more robust and can find a solution under very general conditions, irrespective of the initial condition used. These are ***globally convergent methods***.

Stopping Criteria

- Numerically, however, we will not be able to compute the infinite elements of the sequence $\{x_k\}$
- At some point, we must use some criteria to ***stop the iterations***. Note that the value of α will be unknown.
- We can use the condition $|f(x_k)| < \epsilon$ for some predetermined small value ϵ
- Another possible criteria is $|x_{k+1} - x_k| < \epsilon$

Stopping Criteria

- But both criteria can show its own pitfalls:

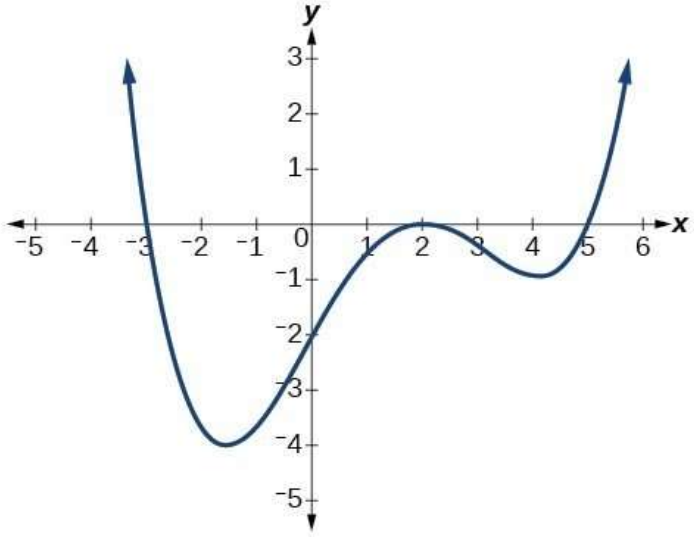


- And we will stop if ***any*** of the criteria is satisfied

[DΣIM]

Bolzano's Theorem

- If f is a continuous real function defined in a closed interval of the line $[a, b]$ the following theorem is satisfied:
- ***Bolzano's Theorem***. If f is continuous within $[a, b]$ and f alternates sign in the extremes of the interval, i.e., if $f(a)f(b) < 0$, then there is a point α within (a, b) for which $f(\alpha) = 0$.



[DΣIM]

Bisection Method

- This theorem allows to locate a region of the abscissa axis were the solution of our equation lies.
- The initial interval, is divided into two halves using the midpoint $c = (a + b)/2$
- We keep the subinterval where Bolzano's Theorem still holds and generate a sequence of decreasing intervals:

$$[a_0, b_0] \Rightarrow [a_1, b_1] \Rightarrow \cdots \Rightarrow [a_n, b_n]$$

Bisection Method

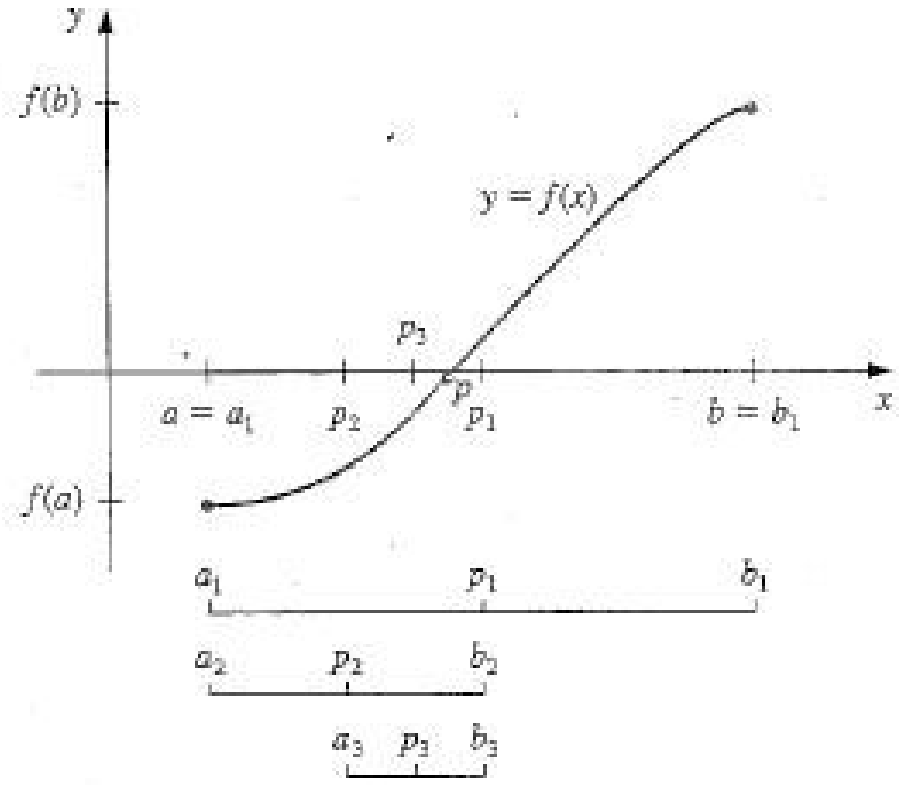
- An ***algorithm*** is an ordered set of logical rules that can solve a determinate problem.
- For the bisection method we have.
  - 1) Given the interval $[a_k, b_k]$, with $k \geq 0$ compute the value

$$c_k = \frac{1}{2}(a_k + b_k)$$

  - 2) If $f(c_k) = 0 \rightarrow \alpha = 0$
  - 3) If $f(a_k)f(c_k) < 0 \rightarrow [a_{k+1}, b_{k+1}] = [a_k, ck]$
  - 4) If $f(a_k)f(c_k) > 0 \rightarrow [a_{k+1}, b_{k+1}] = [c_k, bk]$
  - 5) If $0 < |a_{k+1} - b_{k+1}| < \varepsilon$ stop. Otherwise, go back to step 1)

Numerical Methods

Bisection Method



2022-2023

[DΣIM]

Convergence of the Bisection

- If we denote the successive intervals by $[a_0, b_0], [a_1, b_1]$, ... we have:

$$a_0 \le a_1 \le a_2 \le \cdots \le b_0$$

$$b_0 \ge b_1 \ge b_2 \ge \cdots \ge a_0$$

$$b_{n+1} - a_{n+1} = \frac{1}{2}(b_n - a_n)\ , n > 0$$

- Now the sequence $\{a_k\}$ is increasing and bounded above, while $\{b_k\}$ is decreasing and bounded below. Both sequences converge.

Convergence of the Bisection

- The iterations give:

$$b_n - a_n = 2^{-n}(b_0 - a_0)$$

- And so:

$$\lim_{n\to\infty} b_n - \lim_{n\to\infty} a_n = \lim_{n\to\infty} 2^{-n}(b_0 - a_0) = 0$$

- Putting

$$\alpha = \lim_{n\to\infty} a_n = \lim_{n\to\infty} b_n$$

- We obtain:

$$0 \geq \lim_{n\to\infty} f(a_n) f(b_n) = \left[f(\alpha)\right]^2 \Rightarrow f(\alpha) = 0$$

Error Analysis of the Bisection

- Suppose that a certain stage in the process we obtain the interval $[a_n, b_n]$ and we stop the process. The best approximation of the root is then

$$c_n = \frac{a_n + b_n}{2} \approx \alpha$$

- And the error is:

$$\varepsilon_n = |c_n - \alpha| \leq \frac{1}{2}|b_n - a_n| \leq \frac{1}{2^{n+1}}|b_0 - a_0|$$

Error Control

- Then, after k iterations the upper limit for the error in the bisection method is then:

$$E_k = \frac{b-a}{2^{k+1}}$$

- And if we want a solution with a precision with an error of at most E, we will be forced to make at least the following number of iterations:

$$k > \frac{\ln(b-a) - \ln(2E)}{\ln 2}$$

Convergence Rate of the Bisection

- For the bisection method the convergence rate can be easily computed as:

$$\lim_{k\to\infty}\frac{|\varepsilon_{k+1}|}{|\varepsilon_k|}=\lim_{k\to\infty}\frac{(b-a)2^{k+1}}{(b-a)2^{k+2}}=\frac{1}{2}$$

- As in this case, we have $p=1$ we say that the convergence rate of the bisection method is ***linear***

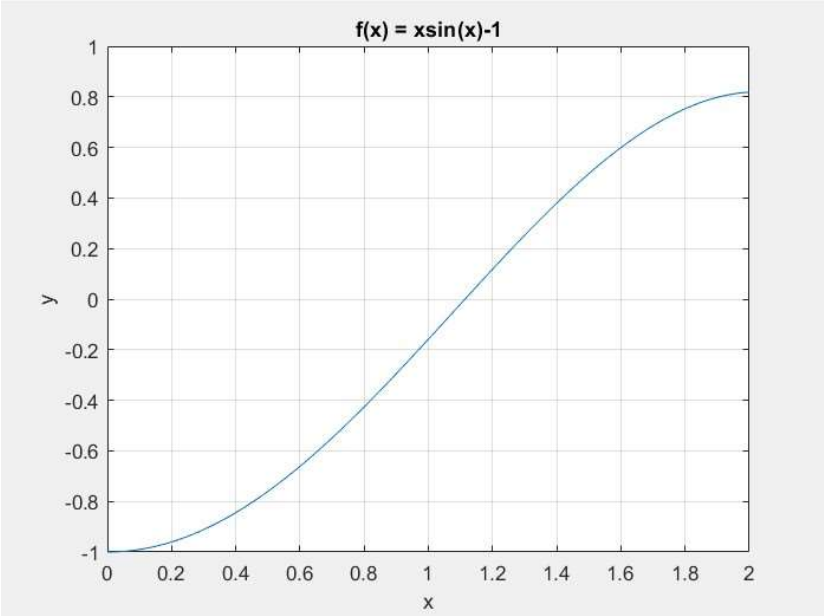
Bisection Example

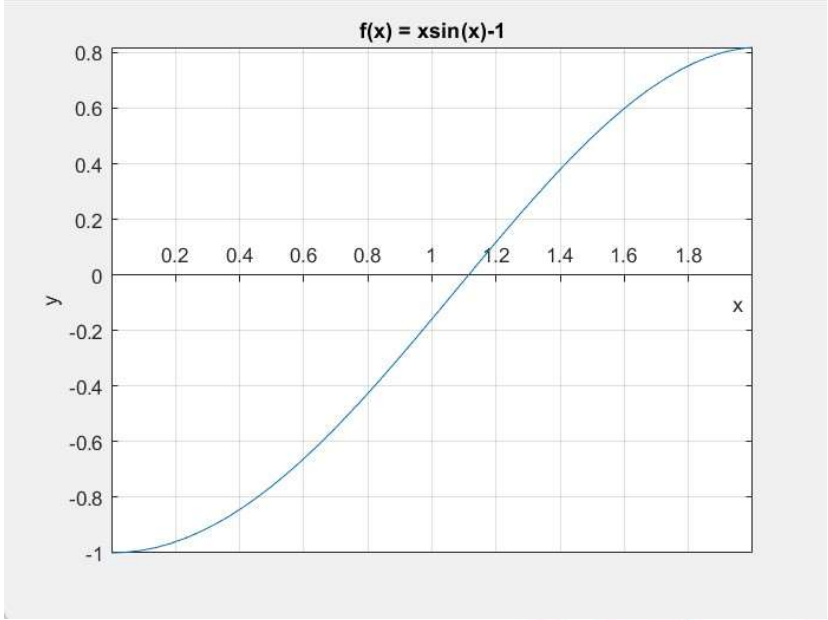
```
% Example Bisection
clear

% Basic Parameters
TolX = 10^-6;

% Define the basic functions
fm = @(x) x.*sin(x) - 1;

% Plot the function

fplot( fm, [0,2])
xlabel('x')
ylabel('y')
ax = gca;
ax.XAxisLocation = 'origin';
ax.YAxisLocation = 'origin';
title (' f(x) = xsin(x)-1 ')
grid on

% Solve the nonlinear equation for x = -2

a = 1;
b = 2;
[xs, error, xx, iter] = Bisection(fm, a, b, TolX);

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 (' xs : %10.8f, error =  %10.8f \n\n', xs, error)

```

Example Bisection

- Let us solve the nonlinear equation

$$f(x) = x\sin(x) - 1 = 0$$





[DΣIM]

Example Bisection

```
function [xroot,err,xx,iter]= Bisection(f,a,b,TolX, varargin)

% Bisection: Bisection method for solving f(x)=0.
% [x,err,xx,iter]= Bisection(f,a,b,TolX,varargin)
% Uses the bisection 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)|))

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

% Basic Parameters

% Variable Check and Setting of default constants

if nargin<3, error('at least 3 input arguments required'), end
if nargin<4 || isempty(TolX), TolX=0.0001; end

% Check that the interval contains a root.
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.
% In the bisection method we know beforehand the number of iterations

MaxIter = ceil(log(abs(b-a)/TolX)/log(2.0));

xx = zeros(1,MaxIter);

% Main Loop
for iter=1:MaxIter
    xx(iter)= a + 0.5*(b-a);
    fx= f(xx(iter),varargin{:});
    err=(b-a)/2;
    if abs(err)<TolX
        break;
    elseif fx*fa>0
        a = xx(iter);
    else
        b = xx(iter);
    end
end
xroot=xx(iter);

```

2022-2023

[DΣIM]

Example Bisection

```
>> Ex_001_Bisection
 k =  1, x = 1.50000000,   f(x) =  0.49624248
 k =  2, x = 1.25000000,   f(x) =  0.18623077
 k =  3, x = 1.12500000,   f(x) =  0.01505104
 k =  4, x = 1.06250000,   f(x) = -0.07182663
 k =  5, x = 1.09375000,   f(x) = -0.02836172
 k =  6, x = 1.10937500,   f(x) = -0.00664277
 k =  7, x = 1.11718750,   f(x) =  0.00420803
 k =  8, x = 1.11328125,   f(x) = -0.00121649
 k =  9, x = 1.11523438,   f(x) =  0.00149600
 k = 10, x = 1.11425781,   f(x) =  0.00013981
 k = 11, x = 1.11376953,   f(x) = -0.00053832
 k = 12, x = 1.11401367,   f(x) = -0.00019925
 k = 13, x = 1.11413574,   f(x) = -0.00002972
 k = 14, x = 1.11419678,   f(x) =  0.00005505
 k = 15, x = 1.11416626,   f(x) =  0.00001266
 k = 16, x = 1.11415100,   f(x) = -0.00000853
 k = 17, x = 1.11415863,   f(x) =  0.00000207
 k = 18, x = 1.11415482,   f(x) = -0.00000323
 k = 19, x = 1.11415672,   f(x) = -0.00000058
 xs : 1.11415672, er =  0.00000191
```

Newton-Raphson Method

- Let f be a function for which f'' exists and is continuous in some interval $[a, b]$, and let α be a zero of this function.
- Using Taylor's theorem, we have:

$$0 = f(\alpha) = f(x+h) = f(x) + hf'(x) + \frac{h^2}{2} f''(\xi),$$

- Where $x < \xi < \alpha, \quad h = \alpha - x$, and x approximates α.

Newton-Raphson Method

- If h is small, it is reasonable to ignore the $O(h^2)$ term and solve the remaining equation for $h = -f(x)/f'(x)$
- With this value, we can obtain a better approximation to the root α:

$$\alpha \approx x - \frac{f(x)}{f'(x)}$$

Newton-Raphson Method

- This procedure suggests the ***Newton-Raphson iteration method***.
- Select a first approximation x_0 to the root and then build the succession of values $\{x_k\}$ such that:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, \quad k \geq 0.$$

Newton-Raphson Method

- This iterate admits a geometrical interpretation. The slope of the tangent to the function f in the point x_0 is the value of the derivative of the function in this point, $f'(x_0)$, and the tangent line through $(x_0, f(x_0))$ is given by

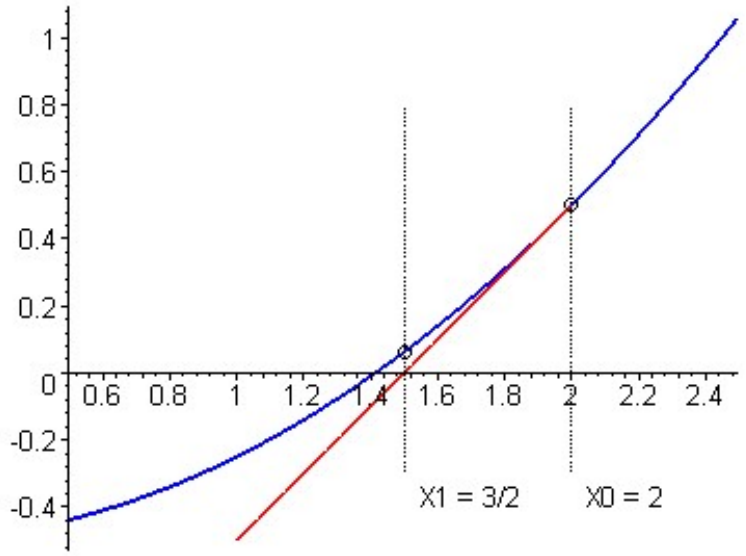
$$r(x) = f(x_0) + f'(x_0)(x - x_0)$$

- When $r(x)=0$ we can find were this straight line cuts the x axis

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

Newton-Raphson Method

- This new value x_1, is a better approximation to the root α and can be used as starting point for the new iteration



[DΣIM]

Newton-Raphson Method

- Thus, starting from the point x_0 we obtain a new point

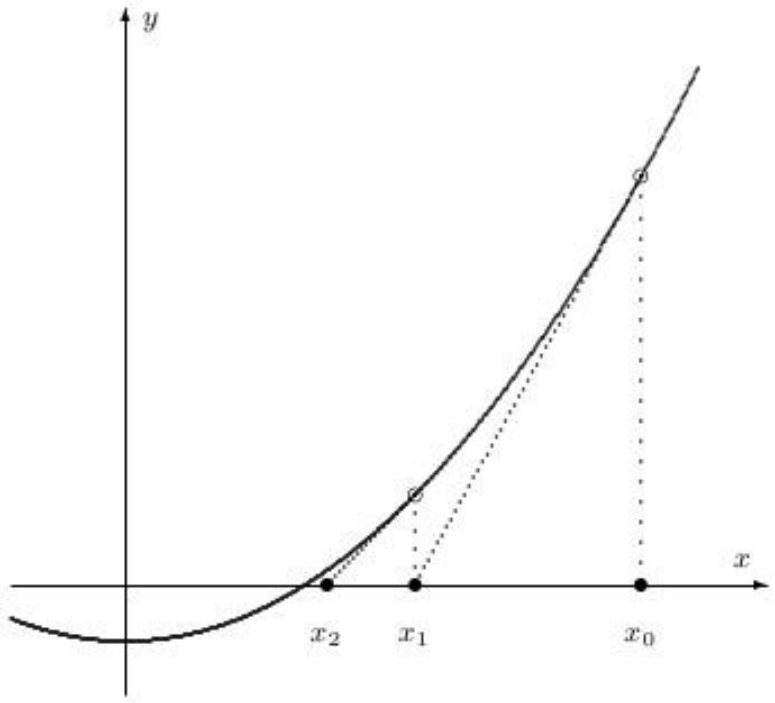
$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

- Repeating this process, we obtain a sequence of iterates which are the crossings of the tangent lines with the x axes:

$$\boxed{x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, \quad k \geq 0}$$

Numerical Methods

Newton-Raphson Method



2022-2023

[DΣIM]

Newton-Raphson Method

- Newton's algorithm can be written as follows:
  - Given x_0 calculate:

$$x = x_0 - \frac{f(x_0)}{f'(x_0)}$$

  - If $|x - x_0| < \varepsilon \rightarrow \alpha = x$
  - Otherwise make $x = x_0$ and start again.
- We could use alternatively the relative error in the second step

$$\frac{|x - x_0|}{|x|} < \varepsilon$$

Error Analysis

- We need to study the convergence of the Newton-Raphson method. Define the error of an iterate as:

$$e_n = x_n - \alpha$$

- And assume that f'' is continuous and α is a ***simple zero*** of f. This is $f'(\alpha) \neq 0$.

Error Analysis

- From the definition of the Newton-Raphson iteration, we have:

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

$$= e_n - \frac{f(x_n)}{f'(x_n)} = \frac{e_n f'(x_n) - f(x_n)}{f'(x_n)}$$

Error Analysis

- Now, using Taylor's theorem we have

$$0 = f(\alpha) = f(x_n - e_n) = f(x_n) - e_n f'(x_n) + \frac{1}{2} e_n^{\,2} f''(\xi_n)$$

- Where ξ_n is a number between x_n and α. From this equation, we have

$$e_n f'(x_n) - f(x_n) = \frac{1}{2} f''(\xi_n) e_n^{\,2}$$

Error Analysis

- Combining these two equations yields:

$$e_{n+1} = \frac{1}{2}\frac{f''(\xi_n)}{f'(x_n)}e_n^2 \approx \frac{1}{2}\frac{f''(\alpha)}{f'(\alpha)}e_n^2 = Ce_n^2$$

- Thus, e_{n+1} is roughly a constant times e_n^2. This indicates that the Newton-Raphson method has ***quadratic convergence***.
- Note that this means that each iterate will double the number of precise digits

Convergence

- We have yet to establish the convergence of the method. Note that if e_n is small and the factor

$$\frac{1}{2}\frac{f''(\xi)}{f'(x_n)}$$

- is not too large, then e_{n+1} will be smaller than e_n.
- Define a quantity $c(\delta)$ dependent on δ by the relation

$$c(\delta) = \frac{1}{2}\frac{\max\limits_{|x-\alpha|\le\delta}|f''(x)|}{\min\limits_{|x-\alpha|\le\delta}|f'(x)|}, \quad (\delta > 0)$$

Convergence

- Suppose that we start the iterations with a point x_0 satisfying $|x_0 - \alpha| \leq \delta$. Then $|e_0| \leq \delta$ and $|\xi_0 - \alpha| \leq \delta$. Hence, by the definition of $c(\delta)$ we have:

$$\frac{1}{2}\frac{|f''(\xi_0)|}{|f'(x_0)|} \leq c(\delta)$$

- Note that as δ converges to zero we have

$$\lim_{\delta \to \infty} c(\delta) = \lim_{\delta \to \infty} \frac{1}{2}\frac{\max_{|x-\alpha| \leq \delta} |f''(x)|}{\min_{|x-\alpha| \leq \delta} |f'(x)|} = \frac{1}{2}\frac{|f''(\alpha)|}{|f'(\alpha)|}$$

Convergence

- Then, we can choose δ so that $\rho = \delta c(\delta) < 1$ because as $\delta \to 0$, we have also that the factor $\rho = \delta c(\delta) \to 0$.
- Therefore:

$$\begin{aligned} |x_1 - \alpha| = |e_1| \le e_0{}^2 c(\delta) = |e_0||e_0| c(\delta) \\ \le |e_0| \delta c(\delta) = |e_0| \rho < |e_0| \le \delta \end{aligned}$$

- And the next point x_1 also lies within δ units of the root α.

Convergence

- This argument can be repeated with each iterate obtaining

$$|e_1| \le \rho |e_0|$$
$$|e_2| \le \rho |e_1| \le \rho^2 |e_0|$$
$$|e_3| \le \rho |e_2| \le \rho^3 |e_0|$$
$$\vdots$$

- And in general $|e_n| \le \rho^n |e_0|$
- Since $0 \le \rho < 1$, we have $\lim_{n\to\infty} \rho^n = 0$ and so:

$$\lim_{n\to\infty} e_n = 0$$

Newton-Raphson Method

- **Theorem:** ***Convergence of Newton's Method***. Let f be a function in $C^2[a,b]$. If α is a point within $[a,b]$ such that $f(\alpha) = 0$ and $f'(\alpha) \neq 0$, then, there exists some $\delta > 0$ for which the Newton-Raphson method generates a convergent succession $\{x_n\}_{n=1}^{\infty}$ to the value α for any initial value x_0 within $[\alpha - \delta, \alpha + \delta]$.

Newton-Raphson Method

- We have just obtained that Newton's method is of quadratic convergence ($p = 2$) while the bisection method had a linear convergence ($p = 1$). Is this really an advantage?
- Let us suppose that $C = 1/2$ and that for some step we have $\varepsilon_k \sim 10^{-n}$. If the convergence is linear, we obtain

$$\varepsilon_{k+1} \approx C\varepsilon_k \approx 0.5\varepsilon_k$$

- Halving the error in each new step

Newton-Raphson Method

- But in the case of a quadratic convergence, we have

$$\varepsilon_{k+1} \approx C\varepsilon_k^2 \approx 0.5 \cdot 10^{-2n}$$

- And we can ***double the number of correct digits*** in each step. Thus, the convergence is much faster.

Newton's Method

- Newton's method is much faster than the bisection method, but it can fail sometimes.
- If there is a point β with $f'(\beta) = 0$ within $[a, b]$ while $f(a)f(b) < 0$ i.e., there is a maximum or a minimum within the interval, or when the derivative values are very small, the slope of the tangent can be very shallow, and this line will cut the x axis outside the search interval

Newton's Method

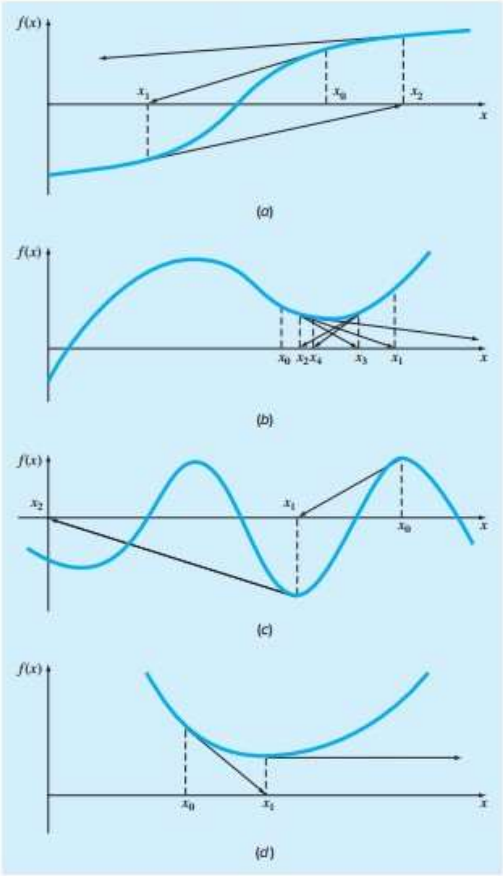


FIGURE 6.6
Four cases where the Newton-Raphson method exhibits poor convergence.

[DΣIM]

Example Newton

```
% Exmaple Newton-Raphson
clear
% Basic Parameters
TolX = 10^-6;
TolF = 10^-6;
MaxIter = 50;
% Define the basic functions
fm = @(x) x.*sin(x) - 1;
dfm = @(x) x.*cos(x) + sin(x);

% Plot the function
x = linspace(0,2,300);
y = fm(x);
plot (x,y)
xlabel('x')
ylabel('y')
ax = gca;
ax.XAxisLocation = 'origin';
ax.YAxisLocation = 'origin';
title (' f(x) = xsin(x)-1 ')
grid on

% Solve the nonlinear equation for x0 = 1
x0 = 1;
[xs, er, xx, iter] = Newton(fm, dfm, x0, TolX, TolF, MaxIter);

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

Numerical Methods

Example Newton

```
function [xroot,err,xx,iter] = Newton(f,df,x0,TolX,TolF,MaxIter,varargin)

% Newton: Newton-Raphson root location zeroes
% [root,ea,xx,iter]=newtraph(func,dfunc,xr,es,maxit,p1,p2,...):
% uses Newton-Raphson method to find the root of f(x)=0

% Input:
%   func = name of function
%   dfunc = name of derivative of function
%   x0 = initial guess
%   TolX is the tolerance for | x(n+1) - x(n) |
%   TolF is the tolerance for the function values | f(x) |
%   MaxIter = maximum allowable iterations (default = 50)

% Output:
%   root = real root
%   err = approximate absolute error (%)
%   xx = array containit the iterations
%   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

% Preallocate Memory for loop array.

xx = zeros(1,MaxIter);
```

```
% Main Array

iter = 1;
err = 100;

xx(iter) = x0;
for iter = 2:MaxIter
    fval = f(xx(iter-1),varargin{:});
    dfval = df(xx(iter-1),varargin{:});
    xx(iter) = xx(iter-1) - fval/dfval;
    err = abs(xx(iter)-xx(iter-1));
    if err <= TolX || abs(fval) < TolF
        break;
    end
end
if (iter >= MaxIter)
    xroot = NaN;
else
    xroot = xx(iter);
end
```

2022-2023

[DΣIM]

Example Newton

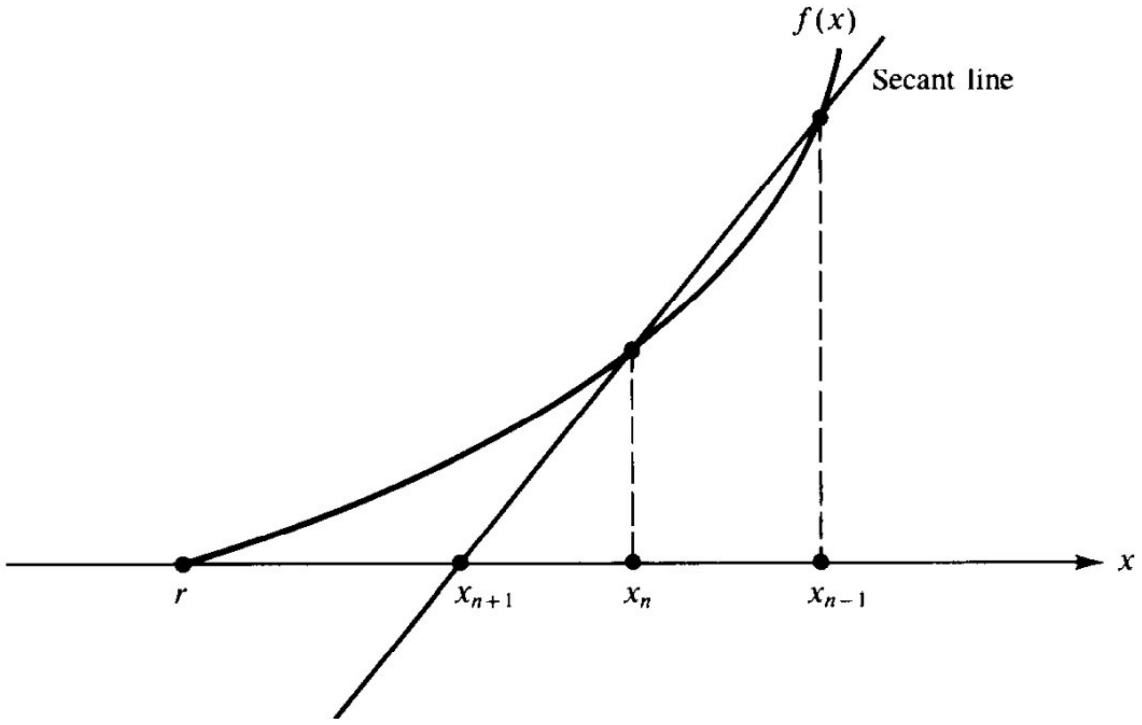
```
>> Ex_002_Newton
 k =  1, x = 1.00000000,  f(x) = -0.15852902
 k =  2, x = 1.11472867,  f(x) =  0.00079373
 k =  3, x = 1.11415713,  f(x) = -0.00000002
 k =  4, x = 1.11415714,  f(x) = -0.00000000
 xs : 1.11415714, er =  0.00000001
```

Secant's Method

- If f is such that f" is a complicated function to calculate we can lose the advantage in speed of the Newton's method
- In this case, it is simpler to use a good approximation of the derivative. We could use the secant line, which cuts the curve instead of having a single point in common

Numerical Methods

Secant's Method



2022-2023

[DΣIM]

Secant's Method

- The derivative in a point x_0 of the function f is defined as:

$$f'(x_0) = \lim_{x \to x_0} \frac{f(x) - f(x_0)}{x - x_0}$$

- But we could use just an approximation using two close values, x_1 and x_0

$$f'(x_0) \approx \frac{f(x_1) - f(x_0)}{x_1 - x_0}$$

Secant's Method

- Now if we change the value of $f'(x_n)$ in Newton's iteration by the former approximation, we obtain a new recurrence without derivatives

$$x_{k+1} = x_k - f(x_k)\frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})}$$

- We will need two points for starting the iterative process, but these can be any two points within the search interval

Error Analysis

- If we define again $e_n = x_n - \alpha$, using the recursive formula for the secant method we have:

$$e_{n+1} = x_{n+1} - \alpha = x_n - f(x_n)\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} - \alpha$$

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

$$= \frac{f(x_n)e_{n-1} - f(x_{n-1})e_n}{f(x_n) - f(x_{n-1})}$$

Error Analysis

- Factoring out $e_n e_{n-1}$ and inserting the factor $(x_n - x_{n-1})/(x_n - x_{n-1})$ we obtain:

$$e_{n+1} = \left[\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}\right]\left[\frac{f(x_n)/e_n - f(x_{n-1})/e_{n-1}}{x_n - x_{n-1}}\right] e_n e_{n-1}$$

- By Taylor's theorem:

$$f(x_n) = f(\alpha + e_n) = f(\alpha) + e_n f'(\alpha) + \frac{1}{2} e_n^2 f''(\alpha) + O(e_n^2).$$

Error Analysis

- And since $f(\alpha) = 0$, we obtain:

$$\frac{f(x_n)}{e_n} = f'(\alpha) + \frac{1}{2} e_n f''(\alpha) + O(e_n^2)$$

$$\frac{f(x_{n-1})}{e_{n-1}} = f'(\alpha) + \frac{1}{2} e_{n-1} f''(\alpha) + O(e_{n-1}^2)$$

- Subtracting these equations, we arrive at:

$$f(x_n)/e_n - f(x_{n-1})/e_{n-1} = \frac{1}{2}(e_n - e_{n-1}) f''(\alpha) + O(e_{n-1}^2)$$

Error Analysis

- Since $x_n - x_{n-1} = e_n - e_{n-1}$, we arrive at

$$\frac{f(x_n)/e_n - f(x_{n-1})/e_{n-1}}{x_n - x_{n-1}} \approx \frac{1}{2} f''(\alpha)$$

- On the other hand:

$$\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} \approx \frac{1}{f'(\alpha)}$$

- And finally

$$e_{n+1} \approx \frac{1}{2}\frac{f''(\alpha)}{f'(\alpha)} e_n e_{n-1} = C e_n e_{n-1}$$

Convergence Order

- To obtain the order of convergence of the secant method, we postulate the following asymptotic relationship:

$$|e_{n+1}| \sim A|e_n|^p$$

- Hence, we can write

$$|e_n| \sim A|e_{n-1}|^p \Leftrightarrow |e_{n-1}| \sim \left(A^{-1}|e_n|\right)^{1/p}$$

Convergence Order

- Now:

$$|e_{n+1}| \sim A|e_n|^p \sim C|e_n| A^{-1/p} |e_n|^{1/p}$$

- Or:

$$A^{1+1/p} C^{-1} \sim |e_n|^{1-p+1/p}$$

- Since the left side is a non-zero constant while $e_n \to 0$ we conclude that

$$1 - p + \frac{1}{p} = 0 \Leftrightarrow p = \frac{1+\sqrt{5}}{2} \approx 1.62$$

Convergence Order

- We can determine the value of A since the right side of the relation is 1.

$$A = C^{1/(1+1/p)} = C^{1/p} = C^{p-1} = C^{0.62} = \left[\frac{f''(\alpha)}{2f'(\alpha)}\right]^{0.62}$$

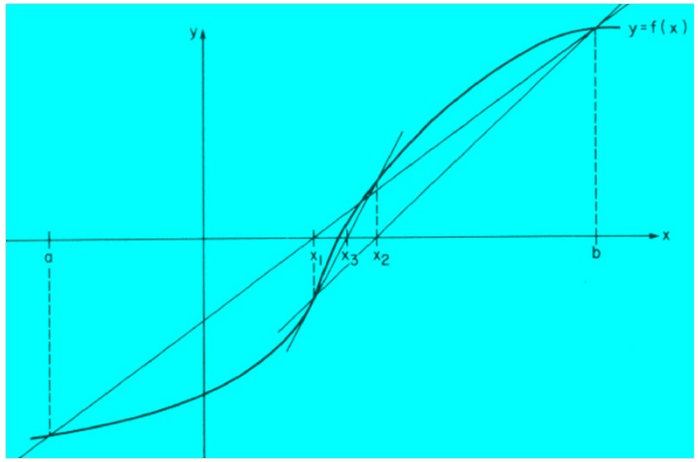
- Finally, we obtain for the secant method:

$$|e_{n+1}| \approx A|e_n|^{(1+\sqrt{5})/2}$$

- Since $\frac{1+\sqrt{5}}{2} \approx 1.62 < 2$, the secant method is ***superlinear*** (better than linear).

Secant's Method

- As $p<2$, the method is not quadratic and is a bit slower than Newton's, but anyway is much faster than the bisection method
- It will have the same problems than Newton's method, though



[DΣIM]

Example Secant

```
% Exmaple Secant
clear
% Basic Parameters
TolX = 10^-6;
TolF = 10^-6;
MaxIter = 50;

% Define the basic functions
fm = @(x) x.*sin(x) - 1;

% Plot the function
x = linspace(0,2,300);
y = fm(x);
plot (x,y)
xlabel('x')
ylabel('y')
ax = gca;
ax.XAxisLocation = 'origin';
ax.YAxisLocation = 'origin';
title (' f(x) = xsin(x)-1 ')
grid on

% Solve the nonlinear equation for x = -2
x0 = 1;
x1 = 2;
[xs, error, xx, iter] = Secant(fm, x0, x1, TolX, TolF, MaxIter);

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 (' xs : %10.8f, error =  %10.8f \n\n', xs, error)
```

Example Secant

```
function [xroot,err,xx,iter]=Secant(f,x0,x1,TolX,TolF,MaxIter,varargin)

% Secant: secant method for solving f(x)=0.
% [xroot,ea,iter,xx] = Secant (f,x0,x1,delta,epsilon,MaxIter):
% uses secant method to find the root of f(x)=0

%Input:
% f = Function to be given as a function handle or an M-file name
% x0 and x1 are the initial approximations to a zero
% TolX is the tolerance for | x(n+1)-x(n) |
% TolF is the tolerance for the function values | f(x(n)) |
% MaxIter is the maximum number of iterations

%Output:
% xroot is the secant method approximation to the zero
% err is the error estimate for xroot
% xx History of x
% k is the 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

% Preallocate Memory for loop array.

xx = zeros(1,MaxIter);

xx(1) = x0;
xx(2) = x1;

for iter = 3:MaxIter
    p0 = xx(iter-2);
    p1 = xx(iter-1);
    fp0 = f(p0,varargin{:});
    fp1 = f(p1,varargin{:});
    p2 = p1-fp1*(p1-p0)/(fp1-fp0);
    err=abs(p2-p1);
    xx(iter) = p2;
    y = f(p2,varargin{:});
    if abs(y)< TolF || err < TolX
        break;
    end
end
if (iter >= MaxIter)
    xroot = NaN;
else
    xroot = xx(iter);
end
```


[DΣIM]

Example Secant

```
>> Ex_003_Secant
 k =  1, x = 1.00000000,   f(x) = -0.15852902
 k =  2, x = 2.00000000,   f(x) =  0.81859485
 k =  3, x = 1.16224045,   f(x) =  0.06658284
 k =  4, x = 1.08806556,   f(x) = -0.03626703
 k =  5, x = 1.11422119,   f(x) =  0.00008895
 k =  6, x = 1.11415720,   f(x) =  0.00000008
 xs : 1.11415720, er =  0.00006399
```


[DΣIM]

Convergence Criteria

- The bisection method is a global convergent method. When we locate an interval containing the root, the method will eventually converge to a solution
- On the other hand, Newton-Rahpson and Secant methods can be sensitive to the initial condition and are only locally convergent. For these methods is essential a good initial approximation to the root

Convergence Criteria

- Using graphic facilities, we can be able to give an approximate location for the root of a given function.
- We need a good criteria to ensure that the succession of values $\{x_n\}$ stops as close as possible to the root.
- As we want to solve the equation $f(x) = 0$, it seems reasonable to think of a criterion like $|f(x_n)| < \varepsilon$

Convergence Criteria

- Using this criterion, the algorithm will generate points on the plane $(x_k, f(x_k))$ until the last point is within a band of width 2ε centered around the x axis.

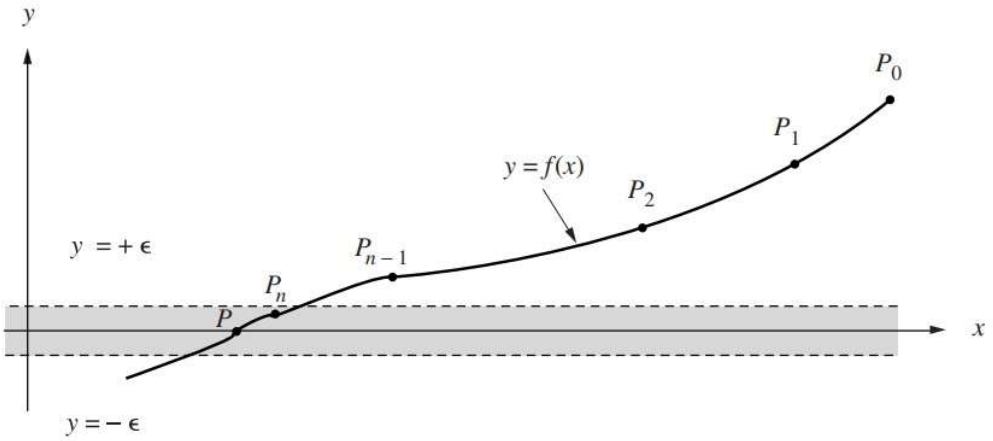


Figure 2.11 (a) The horizontal convergence band for locating a solution to $f(x) = 0$.

Convergence Criteria

- As we want to generate convergent sequences, other criteria could be $|x_k - \alpha| < \varepsilon$, but this criteria can not be used as we do not know the value of the root α.

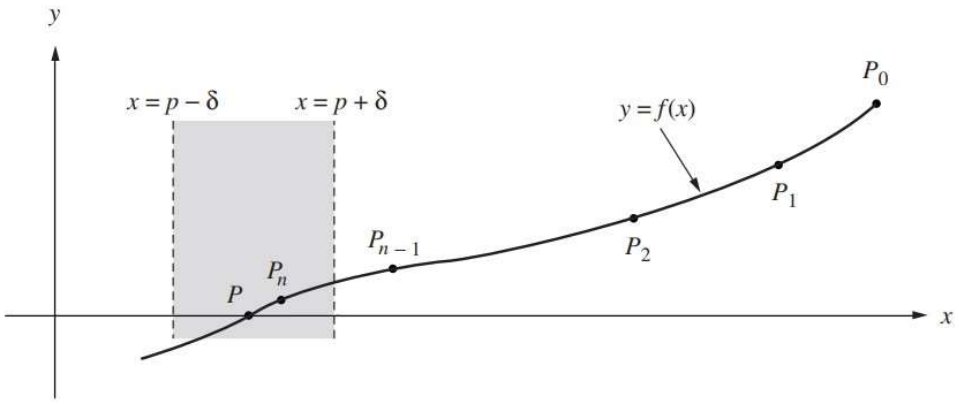


Figure 2.11 (b) The vertical convergence band for locating a solution to $f(x) = 0$.

[DΣIM]

Convergence Criteria

- This criterion is normally substituted by the similar condition $|x_k - x_{k+1}| < \varepsilon$.
- Since the sequence $\{x_n\}$ is a Cauchy sequence, and the distance between terms should tend to zero when we approach the limit, this criterion has a sound basis.
- But, unless the consecutive values x_k, x_{k+1} bracket the root, this criterion can stop far from the solution

Convergence Criteria

- A better criterion to stop the iterations is to combine both criteria. We can choose that at least one of the criteria be satisfied:

$$\left|x_{k+1}-x_k\right|<\delta \quad \text{OR} \quad \left|f(x_k)\right|<\varepsilon$$

- Note however, that in this case the region of the plane selected for this criterion is an unbounded region and we may stop when we are far from the root.

Convergence Criteria

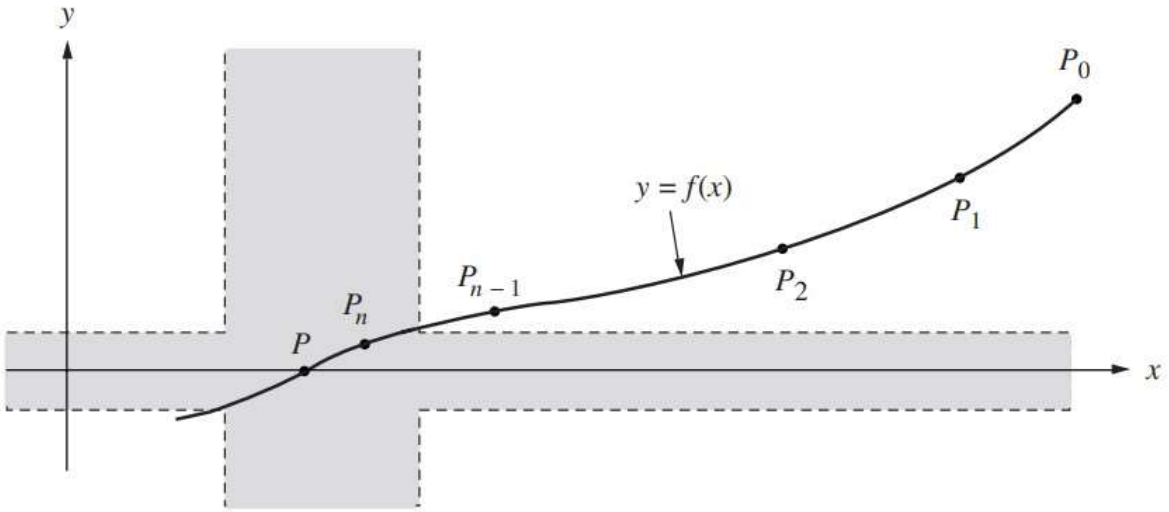


Figure 2.12 (b) The unbounded region defined by $|x - p| < \delta$ OR $|y| < \epsilon$.

[DΣIM]

Convergence Criteria

- Asking for both criteria to be satisfied simultaneously we obtain a bounded region in the plane.

$$\left|x_{k+1} - x_k\right| < \delta \quad \text{AND} \quad \left|f(x_k)\right| < \varepsilon$$

- The danger is here on the shape of the function f. If this function is flat near the solution, it can be very difficult to satisfy both criteria simultaneously and we can enter a nearly infinite loop.

Convergence Criteria

- There is no definitive, always working criterion. We should be aware of the limitations of each criteria and adapt to each problem.

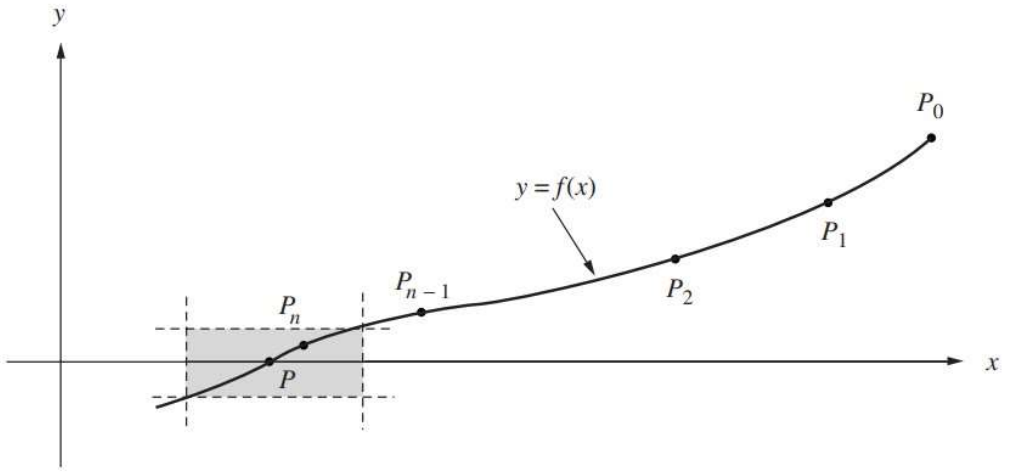


Figure 2.12 (a) The rectangular region defined by $|x - p| < \delta$ AND $|y| < \epsilon$.

[DΣIM]

Fixed Points and Functional Iteration

- Newton's method, the secant method and many others can be written in the form:

$$x_{n+1} = g(x_n) \quad (n \geq 0)$$

- The algorithms defined by such an equation are called ***functional iterations***. Thus, in Newton method we have

$$g(x) = x - \frac{f(x)}{f'(x)}$$

Fixed Points and Functional Iteration

- We want to generate sequences that converge to the solution, this means that:

$$\lim_{n\to\infty} x_n = \alpha$$

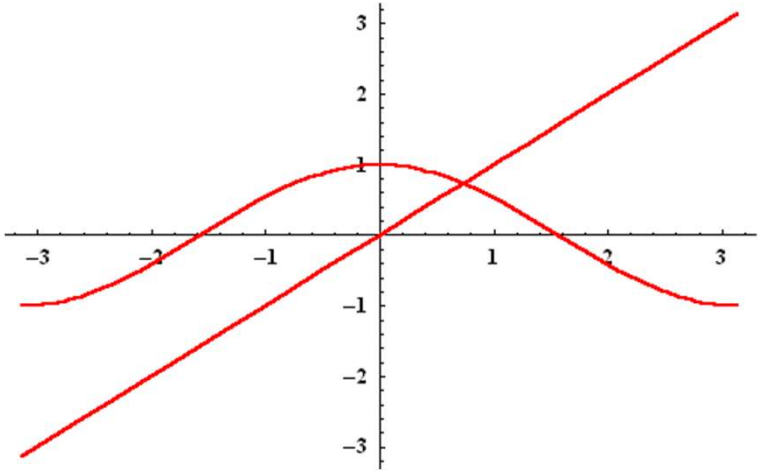
- Note that if g is a continuous function, then:

$$g(\alpha) = g\left(\lim_{n\to\infty} x_n\right) = \lim_{n\to\infty} g(x_n) = \lim_{n\to\infty} x_{n+1} = \alpha$$

- We call α a ***fixed point*** of the function g

Fixed Point Iterations

- Many problems can be reduced to the problem of finding a fixed point of a function. Geometrically, we look for the point where the function g cuts the straight-line $y=x$



[DΣIM]

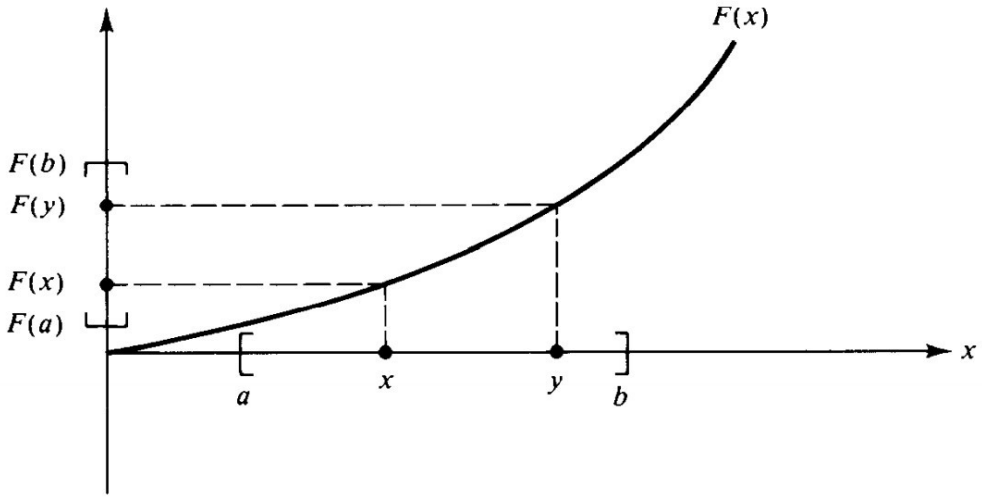
Fixed Point Iterations

- The function g whose fixed points are sought will be usually, a mapping from one vector space into another. We will analyze the simplest case when g maps some closed set $C \subset \mathbb{R}$ into itself.
- We are interested in ***contractive mappings***, defined by a number $\lambda < 1$ such that:

$$\left|g(x) - g(y)\right| \leq \lambda \left|x - y\right|$$

Fixed Point Iterations

- In contractive mappings the distance between x and y is mapped into a smaller distance between $g(x)$ and $g(y)$.



[DΣIM]

Contractive Mapping Theorem

- ***Theorem***. Let g a contractive mapping of a closed set $C \subset \mathbb{R}$ into C. Then g has a unique fixed point. This fixed point is the limit of every sequence obtained from the equation

$$x_{n+1} = g(x_n)$$

- With any starting point $x_0 \in C$.
- It is also said that a contractive mapping satisfies the ***Lipschitz condition***.

Contractive Mapping Theorem

- Using the contractive property, we have:

$$|x_n - x_{n-1}| = |g(x_{n-1}) - g(x_{n-2})| \le \lambda |x_{n-1} - x_{n-2}|$$

- This argument can be repeated to obtain:

$$|x_n - x_{n-1}| \le \lambda |x_{n-1} - x_{n-2}| \le \lambda^2 |x_{n-2} - x_{n-3}|$$
$$\le \cdots \le \lambda^{n-1} |x_1 - x_0|$$

- Now, since

$$x_n = x_0 + (x_1 - x_0) + (x_2 - x_1) + \cdots + (x_n - x_{n-1})$$

Contractive Mapping Theorem

- The sequence $\{x_n\}$ will converge if and only if the series

$$\sum_{n=1}^{\infty}\left(x_n - x_{n-1}\right)$$

- Is convergent. It suffices to prove that the series

$$\sum_{n=1}^{\infty}\left|x_n - x_{n-1}\right|$$

- converges

Contractive Mapping Theorem

- Using the comparison test we have

$$\sum_{n=1}^{\infty}\left|x_n - x_{n-1}\right| \le \sum_{n=1}^{\infty}\lambda^{n-1}\left|x_1 - x_0\right| = \frac{1}{1-\lambda}\left|x_1 - x_0\right|$$

- And the sequence converges with

$$\alpha = \lim_{n\to\infty} x_n$$

- Thus, α is the fixed point $\alpha = g(\alpha)$.

[DΣIM]

Contractive Mapping Theorem

- An alternative proof of this theorem, is as follows:
- We define the continuous function $G(x) = g(x) - x$. As we have:

$$\begin{cases} G(a) = g(a) - a \geq 0, & \text{as } g(a) \in [a,b] \\ G(b) = g(b) - b \leq 0, & \text{as } g(b) \in [a,b] \end{cases}$$

- The conditions for the Bolzano theorem are fulfilled and there must be a point α such that $G(\alpha) = g(\alpha) - \alpha = 0$ within $[a,b]$ that is, a fixed point with $g(\alpha) = \alpha$

Contractive Mapping Theorem

- We still must prove that is the unique solution. Suppose that there are two fixed points: α_1 and α_2. Then

$$g(\alpha_1) = \alpha_1 \quad \text{and} \quad g(\alpha_2) = \alpha_2 \quad \text{on} \quad \alpha_1 \neq \alpha_2$$

- But the Lipsitchz condition implies that

$$|\alpha_1 - \alpha_2| = |g(\alpha_1) - g(\alpha_2)| \leq \lambda |\alpha_1 - \alpha_2| < |\alpha_1 - \alpha_2|$$

- Which is absurd unless $\alpha_1 = \alpha_2$

Fixed Point Iterations

- Lipsitchz condition of contractive mappings is intimately connected with values of the derivative of the function g.
- $$|g(x_1) - g(x_2)| \le L|x_1 - x_2| \Leftrightarrow$$
$$g'(x) \approx \frac{|g(x_1) - g(x_2)|}{|x_1 - x_2|} \le L < 1$$
- We need a function g such that the values of the derivative in a selected interval are less than the unity

Fixed Point Iterations

- If we have a function g such that

$$\left|g'(x)\right| \le \lambda < 1$$

- In some interval $|x - \alpha| < \rho$. The iterations will converge as $n \to \infty$ as we have:

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

- And then

$$\left|\alpha - x_{n+k}\right| \approx \left|g'(\alpha)\right|^k \left|\alpha - x_n\right|$$

- The quantity $\hat{\lambda} = |g'(\alpha)|$ is called the ***asymptotic convergence factor***.

Error Analysis

- Suppose that g has a fixed point α and the sequence $\{x_n\}$ is computed using the formula $x_{n+1} = g(x_n)$. Let the error of the nth iterate be:

$$e_n = x_n - \alpha$$

- If g' exists and is continuous, by the mean value theorem:

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

$$e_{n+1} = g'(\xi_n)e_n$$

- Where $x_n < \xi_n < \alpha$.

Error Analysis

- The condition $|g'(x)| < 1$ for all x ensures that the errors decrease in magnitude. If e_n is small, then ξ_n is near α, and $g'(\xi_n) \approx g'(\alpha)$. One would expect rapid convergence if $g'(\alpha)$ is small. An ideal situation would be that $g'(\alpha)$=0. In this case we will need additional terms in the Taylor series. Suppose that q is an integer such that:

$$g^{(k)}(\alpha) = 0 \quad \text{for} \quad 1 \le k < q \quad \text{but} \quad g^{(q)}(\alpha) \neq 0$$

Error Analysis

- Expanding the Taylor series of $g(x_n)$ around α, we have:

$$e_{n+1} = x_{n+1} - \alpha = g(x_n) - g(\alpha) = g(\alpha + e_n) - g(\alpha)$$

$$= \left[g(\alpha) + e_n g'(\alpha) + \frac{1}{2} e_n{}^2 g''(\alpha) + \cdots \right] - g(\alpha)$$

$$= e_n g'(\alpha) + \frac{1}{2} e_n{}^2 g''(\alpha) + \cdots + \frac{1}{(q-1)!} e_n{}^{q-1} g^{(q-1)}(\alpha) + \frac{1}{q!} e_n{}^q g^{(q)}(\xi_n)$$

- Therefore:

$$e_{n+1} = \frac{1}{q!} e_n{}^q g^{(q)}(\xi_n)$$

Error Analysis

- If we know that $\lim_{n\to\infty} x_n = \alpha$ we obtain

$$\lim_{n\to\infty} \frac{|e_{n+1}|}{|e_n|^q} = \frac{1}{q!} g^{(q)}(\xi_n)$$

- And the order of convergence will be q

Fixed Point Iterations

- Any equation of the form $f(x) = 0$ can be rewritten in the form $x = g(x)$. For instance, we could rewrite the equation:

$$x + \ln x = 0$$

- in many ways. Thus, we have the equivalent nonlinear equations

$$x = -\ln x, \quad x = e^{-x}, \quad x = \frac{(x + e^{-x})}{2}$$

- To solve our nonlinear equation, we will need to ensure that we have a contractive mapping.

Fixed Point Iterations

- When we have completed the former transformation, the algorithm is simple as we just follows the simple recurrence

$$\boxed{x_{k+1} = g(x_k)}$$

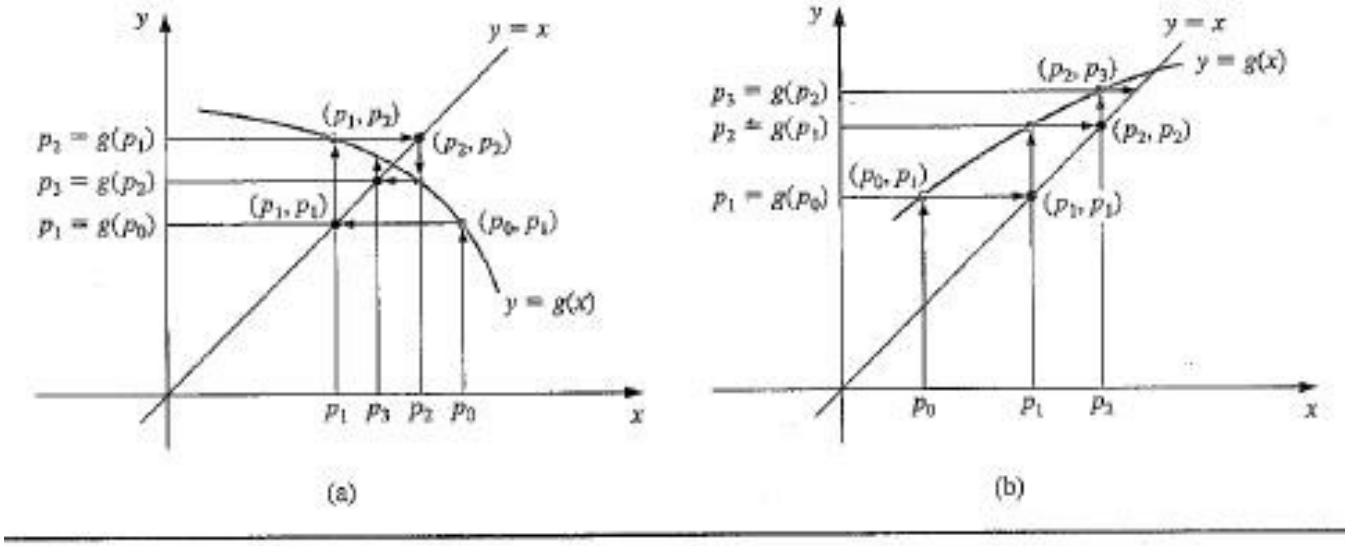
- The solution $x=\alpha$ is reached when

$$\alpha = g(\alpha)$$

- Therefore, this method is called a ***fixed-point iteration***

Numerical Methods

Fixed Point Iteration



2022-2023

[DEIM]

Fixed Point Iteration

- There can be many ways of writing the equation $g(x)=x$ and they are not equivalent. We are interested in the equations that make the iterative process convergent:

$$\lim_{k\to\infty} x_{k+1} = \lim_{k\to\infty} g(x_k) = \alpha$$

- Obviously, we can not make infinite iterations and we will need to stop when

$$\left|x_{k+1} - x_k\right| < \varepsilon$$

Fixed Point Iteration

- Consider the nonlinear equation:

$$x^3 + 4x^2 + 10 = 0$$

- This equation can be rewritten in many ways to obtain a functional iteration.

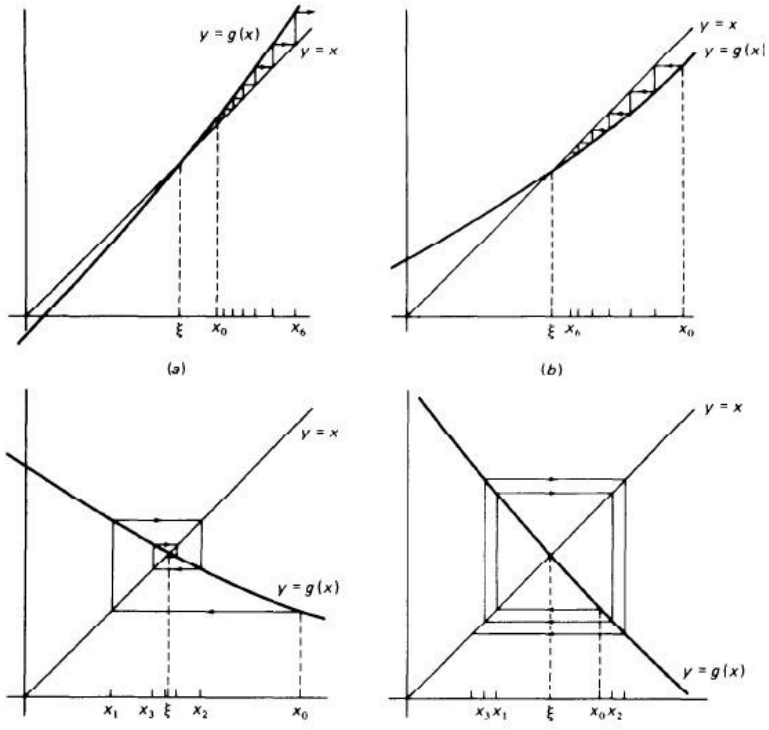
$$x = g_1(x) = x - x^3 - 4x^2 + 10 \qquad x = g_3(x) = \left(\frac{10}{4+x}\right)^{1/2}$$

$$x = g_2(x) = \frac{1}{2}(10 - x^3)^{1/2} \qquad x = g_4(x) = x - \frac{x^3 + 4x^2 - 10}{3x^2 + 8x}$$

- But not all of them will be useful for fixed-point iteration.

Numerical Methods

Fixed Point Iterations



2022-2023

[DΣIM]

Numerical Methods

Fixed Point Iterations

```
% Exmaple Fixed Point Iterations

clear

% Basic Parameters
x0 = 1.5;
delta = 10^-6;
max = 10;
MaxIter = 50;

% Define the basic functions
g1 = @(x) x - x.^3 - 4.*x.^2 + 10;
g2 = @(x) 0.5.*sqrt(10 - x.^3);
g3 = @(x) sqrt(10 ./ (4 + x));
g4 = @(x) x - (x.^3 + 4.*x.^2 - 10) / (3.*x.^2 + 8.*x);

% Preallocate Memory for loop array.

xx = zeros(1,MaxIter);
err = zeros(1,MaxIter);

xx(1) = x0;
err(1) = 0.0;
iter = 1;
```

```
% First Iterate

for k = 2:MaxIter
    xx(k) = g1(xx(k-1));
    err(k) = abs(xx(k) - xx(k-1));
    if err(k) < delta || err(k) > max
        break;
    end
    iter = iter + 1;
end

fprintf ( ' \n First Iterate \n')
for k = 1:iter
    fprintf ( ' k = %2d, x = %10.6f \n', k, xx(k))
end
```

2022-2023

[DΣIM]

Numerical Methods

Fixed Point Iterations

```
>> Ex_004_Fix

 First Iterate
 k =  1, x =   1.500000
 k =  2, x =  -0.875000
 k =  3, x =   6.732422
```

```
Second Iterate
k =  1, x =   1.500000 , err =   0.000000
k =  2, x =   1.286954 , err =   0.213046
k =  3, x =   1.402541 , err =   0.115587
k =  4, x =   1.345458 , err =   0.057082
k =  5, x =   1.375170 , err =   0.029712
k =  6, x =   1.360094 , err =   0.015076
k =  7, x =   1.367847 , err =   0.007753
k =  8, x =   1.363887 , err =   0.003960
k =  9, x =   1.365917 , err =   0.002030
k = 10, x =   1.364878 , err =   0.001039
k = 11, x =   1.365410 , err =   0.000532
k = 12, x =   1.365138 , err =   0.000272
k = 13, x =   1.365277 , err =   0.000139
k = 14, x =   1.365206 , err =   0.000071
k = 15, x =   1.365242 , err =   0.000037
k = 16, x =   1.365224 , err =   0.000019
k = 17, x =   1.365233 , err =   0.000010
k = 18, x =   1.365228 , err =   0.000005
k = 19, x =   1.365231 , err =   0.000003
k = 20, x =   1.365230 , err =   0.000001
```

```
Third Iterate
k =  1, x =   1.500000 , err =   0.000000
k =  2, x =   1.348400 , err =   0.151600
k =  3, x =   1.367376 , err =   0.018977
k =  4, x =   1.364957 , err =   0.002419
k =  5, x =   1.365265 , err =   0.000308
k =  6, x =   1.365226 , err =   0.000039
k =  7, x =   1.365231 , err =   0.000005

Fourth Iterate
k =  1, x =   1.500000 , err =   0.000000
k =  2, x =   1.373333 , err =   0.126667
k =  3, x =   1.365262 , err =   0.008071
k =  4, x =   1.365230 , err =   0.000032
>>
```

2022-2023

[DΣIM]