Lecture 3

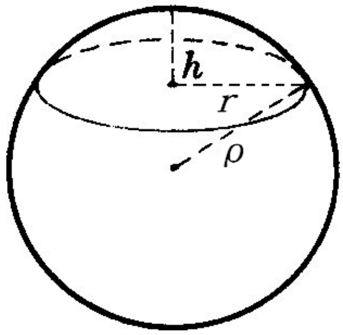
第二章 單變數函數

解題步驟

- 1.瞭解問題
- 2.建構問題之數學模式
- 3.解題
  - *真正解-數學方法*
  - *近似解-數值方法*

球體之部分體積

問題：求軟木球進水深度h之體積V



$$r^2 = \rho^2 - (\rho - x)^2$$

$$V = \int_0^h \pi r^2 dx = \int_0^h \pi[\rho^2 - (\rho - x)^2]dx$$

$$= \pi \int_0^h [\rho^2 - (\rho^2 - 2\rho x + x^2)]dx$$

$$= \pi \int_0^h (2\rho x - x^2)dx = \pi(\rho x^2 - \frac{x^3}{3})\Big|_0^h$$

$$= \pi\rho h^2 - \frac{h^3}{3}$$

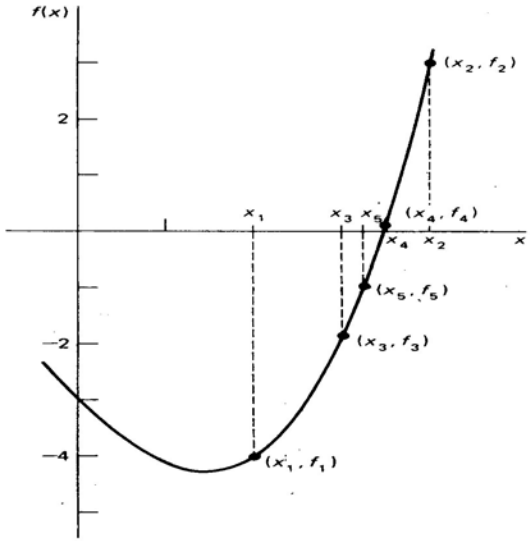
$$= \frac{\pi}{3} h^2 (3\rho - h)$$

單變數函數求根方法

- **二分法(Bisection Method)**
- **錯位法(Regula Falsi)**
  - *改良式錯位法*
- **正割法(Secant Method)**
- **牛頓法(Newton Method)**
- **Muller法**

二分法(Bisection Method)

- 找到一個確知包括有零點(函數為零)的區間，將此區間等分為兩區段，然後再找有包含零點的那個區段。
- $f(x_1)$與$f(x_2)$異號



Bisection Algorithm

Method of halving the interval (Bisection method)

To determine a root of $f(x)=0$, accurate within a specified tolerance value, given values of x_1 and x_2 such that $f(x_1)$ and $f(x_2)$ are of opposite sign,

```
DO  WHILE ½|x1 − x2| ≥ tolerance value,
      Set x3 = (x1 + x2)/2.
      IF    f(x3) of opposite sign to f(x1):
            Set x2 = x3.
      ELSE  Set x1 = x3.
      ENDIF.
ENDDO.
```

The final value of x_3 approximates the root.

Note. The method may give a false root if $f(x)$ is discontinuous on $[x_1, x_2]$.

二分法實例1

$$f(x) = x^3 + x^2 - 3x - 3 = 0$$

Method of halving the interval for $f(x) = x^3 + x^2 - 3x - 3 = 0$.

Iteration number	x_1	x_2	x_3	$f(x_1)$	$f(x_2)$	$f(x_3)$	Maximum error in x_3
1	1	2	1.5	−4.0	3.0	−1.875	0.5
2	1.5	2	1.75	−1.875	3.0	0.17187	0.25
3	1.5	1.75	1.625	−1.875	0.17187 . . .	−0.94335 . . .	0.125
4	1.625	1.75	1.6875	−0.94335 . . .	0.17187	0.40942	0.0625
5	1.6875	1.75	1.71875	−0.40942 . . .	0.17187 . . .	−0.12478	0.03125
6	1.71875	1.75	1.73437 . . .	−0.12478 . . .	0.17187 . . .	−0.02198	0.015625*
7	1.71875	1.73437 . . .	1.72656 . . .				0.0078125
	.	.	.				
	.	.	.				
	.	.	.				
∞			1.73205 . . .			−0.00000 . . .	

* Actual error in x_3 after 5 iterations is 0.01330.

二分法實例1

$$f(x) = x^3 + x^2 - 3x - 3 = 0$$

使用BisectionMethod.m

```
• f=inline('x.^3+x.^2-3*x-3');
• a=1; b=2; kmax=8; tol=0.00001;
• ya=f(a); yb=f(b);
• if sign(ya)==sign(yb), error('function has same sign at end points'), end;
• disp('    step      a       b       m       ym      bound')
• for k=1:kmax
•    m=(a+b)/2; ym=f(m); iter=k; bound=(b-a)/2;
•    out=[iter, a, b, m, ym, bound];
•    disp(out);
•    if abs(ym)<tol, disp('bisection has converged');break; end
•    if sign(ym)~=sign(ya)
•       b=m; yb=ym;
•    else
•       a=m; ya=ym;
•    end
•    if (iter>=kmax),disp('zero not found to desired tolerance'), end
• end;
```

二分法實例1

step	a	b	m	ym	bound
1.0000	1.0000	2.0000	1.5000	-1.8750	0.5000
2.0000	1.5000	2.0000	1.7500	0.1719	0.2500
3.0000	1.5000	1.7500	1.6250	-0.9434	0.1250
4.0000	1.6250	1.7500	1.6875	-0.4094	0.0625
5.0000	1.6875	1.7500	1.7188	-0.1248	0.0313
6.0000	1.7188	1.7500	1.7344	0.0220	0.0156
7.0000	1.7188	1.7344	1.7266	-0.0518	0.0078
8.0000	1.7266	1.7344	1.7305	-0.0150	0.0039

二分法實例2

問題：求軟木球進水深度h之體積V

若球半徑 $\rho = 1, r^2 = \rho^2 - (\rho - x)^2 =$

$$V = \int_0^h \pi r^2 dx = \int_0^h \pi [\rho^2 - (\rho - x)^2] dx$$

$$= \pi \int_0^h [\rho^2 - (\rho^2 - 2\rho x + x^2)] dx$$

$$= \pi \int_0^h (2\rho x - x^2) dx = \pi (\rho x^2 - \frac{x^3}{3}) \Big|_0^h$$

$$= \pi \rho h^2 - \frac{h^3}{3}$$

$$= \frac{\pi}{3} h^2 (3\rho - h)$$

二分法實例2

$$f(x) = x^3 - 3x^2 + 1 = 0$$（求軟木球吃水深度之體積）

```
f=inline('x.^3-3*x.^2+1');
a=0; b=1; kmax=6; tol=0.00001;
ya=f(a); yb=f(b);
if sign(ya)==sign(yb), error('function has same sign at end points'), end;
disp('    step      a       b       m       ym      bound')
for k=1:kmax
   m=(a+b)/2; ym=f(m); iter=k; bound=(b-a)/2;
   out=[iter, a, b, m, ym, bound];
   disp(out);
   if abs(ym)<tol, disp('bisection has converged');break; end
   if sign(ym)~=sign(ya)
      b=m; yb=ym;
   else
      a=m; ya=ym;
   end
   if (iter>=kmax),disp('zero not found to desired tolerance'), end
end;
```

二分法實例2

$$f(x) = x^3 - 3x^2 + 1 = 0$$

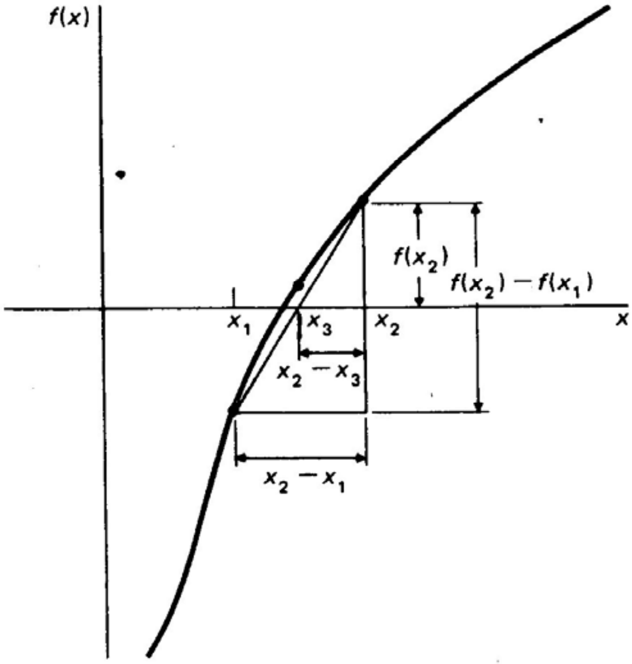
step	a	b	m	ym	bound
1.0000	0	1.0000	0.5000	0.3750	0.5000
2.0000	0.5000	1.0000	0.7500	-0.2656	0.2500
3.0000	0.5000	0.7500	0.6250	0.0723	0.1250
4.0000	0.6250	0.7500	0.6875	-0.0930	0.0625
5.0000	0.6250	0.6875	0.6563	-0.0094	0.0313
6.0000	0.6250	0.6563	0.6406	0.0317	0.0156

錯位法(Regula Falsi)

$$\frac{x_2 - x_3}{x_2 - x_1} = \frac{f(x_2)}{f(x_2) - f(x_1)},$$

$$x_3 = x_2 - \frac{f(x_2)}{f(x_2) - f(x_1)}(x_2 - x_1).$$

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



錯位法(Regula Falsi)

Method of linear interpolation (*regula falsi* method)

To determine a root of $f(x) = 0$, given values of x_1 and x_2 such that $f(x_1)$ and $f(x_2)$ are of opposite sign,

DO WHILE $|x_2 - x_1| \geqslant$ tolerance value 1, or
$|f(x_3)| \geqslant$ tolerance value 2,
 Set $x_3 = x_2 - f(x_2)\dfrac{x_2 - x_1}{f(x_2) - f(x_1)}$.
 IF $f(x_3)$ of opposite sign to $f(x_1)$:
  Set $x_2 = x_3$.
 ELSE Set $x_1 = x_3$.
 ENDIF.
ENDDO.

Note. The method may give a false root if $f(x)$ is discontinuous on $[x_1, x_2]$.

錯位法實例

- 收斂速度較二分法快。

使用RegulaFalsi.m

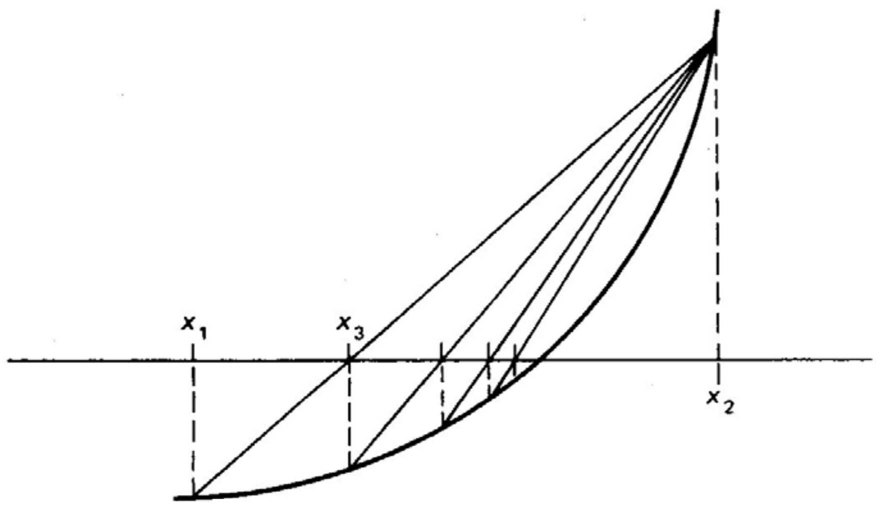
Method of linear interpolation for $f(x) = x^3 + x^2 - 3x - 3 = 0$.

Iteration number	x_1	x_2	x_3	$f(x_1)$	$f(x_2)$	$f(x_3)$
1	1.0	2.0	1.57142	−4.0	3.0	−1.36449
2	1.57142	2.0	1.70540	−1.36449	3.0	−0.24784
3	1.70540	2.0	1.72788	−0.24784	3.0	−0.03936
4	1.72788	2.0	1.73140	−0.03936	3.0	−0.00615
5	1.73140	2.0	1.73194*			

* Error in x_3 after 5 iterations is 0.00011.

錯位法(Regula Falsi)

- 單側搜尋方式：若x_1與x_2間函數之曲率很大時，其收斂速度將非常慢。



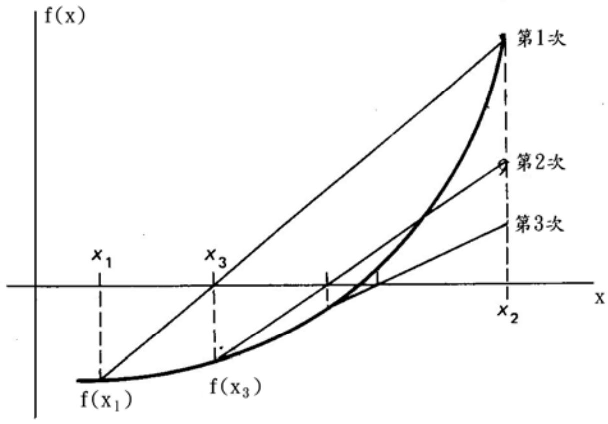
改良式錯位法

- 方法改良：$f(x_2)^*=f(x_2)/2$

Modified linear interpolation method

To determine a root of $f(x) = 0$, given values of x_1 and x_2 such that $f(x_1)$ and $f(x_2)$ are of opposite sign,

Set SAVE = $f(x_1)$; set F1 = $f(x_1)$; set F2 = $f(x_2)$.
DO WHILE $|x_1 - x_2| \geq$ tolerance value 1, or
$|f(x_3)| \geq$ tolerance value 2,
  Set $x_3 = x_2 - \text{F2}\dfrac{x_2 - x_1}{\text{F2} - \text{F1}}$.
  IF $f(x_3)$ of opposite sign to F1:
    Set $x_2 = x_3$.
    Set F2 = $f(x_3)$.
    IF $f(x_3)$ of same sign as SAVE:
      Set F1 = F1/2.
    ENDIF.
  ELSE Set $x_1 = x_3$.
    Set F1 = $f(x_3)$.
    IF $f(x_3)$ of same sign as SAVE:
      Set F2 = F2/2.
    ENDIF.
  ENDIF.
  Set SAVE = $f(x_3)$.
ENDDO.



改良式錯位法實例

Modified linear interpolation for $f(x) = x^3 + x^2 - 3x - 3 = 0$.

Iteration number	x_1	x_2	x_3	F1	F2	$f(x_3)$	SAVE
1	1.0	2.0	1.57142	−4.0	3.0	−1.3449	−4.0
2	1.57142	2.0	1.77557	−1.36449	1.5*	0.42369	−1.36449
3	1.57142	1.77557	1.72720	−1.36449	0.42369	−0.04576	0.42369
4	1.72720	1.77557	1.73191	−0.04576	0.42369	-1.332×10^{-3}	−0.04576
5	1.73191	1.77557	1.732183†	-1.332×10^{-3}	0.21184*		

* These function values are old F2/2.
† Error in x_3 after 5 iterations is −0.00013.

正割法(Secant Method)

- **To determine a root of f(x)=0, given values of x1 and x2 such that f(x1) and f(x2) are their values of function**

$$\text{DO WHILE} \left| f(x_{k+1}) \right| > \text{tolerance value}$$

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

$$\text{ENDDO}$$

正割法實例

Secant method for $f(x) = x^3 + x^2 - 3x - 3 = 0$.

Iteration number	x_1	x_2	x_3	$f(x_1)$	$f(x_2)$	$f(x_3)$
1	1.0	2.0	1.57142	−4.0	3.0	−1.36449
2	2.0	1.57142	1.70540	3.0	−1.36449	−0.24784
3	1.57142	1.70540	1.73513	−1.36449	−0.24784	0.02920
4	1.70540	1.73513	1.73199	−0.24784	0.02920	−0.0005755
5	1.73513	1.73199	1.73205*			

* Error in x_3 after 5 iterations is $<10^{-6}$.

使用SecantMethod.m

```
-------------- Secant Method -----------
step  x(k-1)   x(k)    x(k+1)      y(k+1)        Dx(k+1)
  1   1.0000  2.0000  1.5714  -1.3644e+000  -4.2857e-001
  2   2.0000  1.5714  1.7054  -2.4775e-001   1.3398e-001
  3   1.5714  1.7054  1.7351   2.9255e-002   2.9725e-002
  4   1.7054  1.7351  1.7320  -5.1518e-004  -3.1394e-003
  5   1.7351  1.7320  1.7321  -1.0390e-006   5.4327e-005
  6   1.7320  1.7321  1.7321   3.7030e-011   1.0979e-007

Run Time for Secant Method =0.9370 (sec)
```

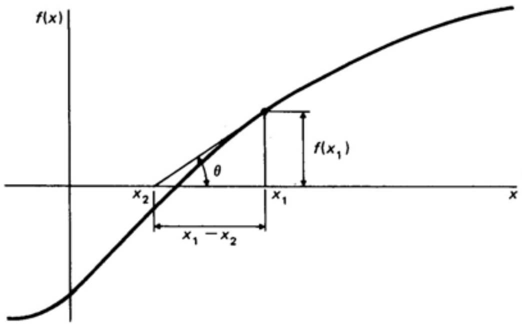
牛頓法(Newton Method)

Newton's method

To determine a root of $f(x) = 0$, given a value x_1, reasonably close to the root,

DO WHILE $|x_2 - x_1| \geqslant$ tolerance value 1, or
$|f(x_2)| \geqslant$ tolerance value 2, or
$f'(x_1) \neq 0$,
Set $x_2 = x_1 - f(x_1)/f'(x_1)$.
Set $x_1 = x_2$
ENDDO.

Note. The method may converge to a root different from the expected one or diverge if the starting value is not close enough to the root.



牛頓法實例

$$求 f(x) = x^3 + x^2 - 3x - 3 之根$$

使用NewtonMethod.m

-------------- Newton Method -----------
Newton method has converged

step	x	y
1	0.5000	-4.1250e+000
2	-2.8000	-8.7120e+000
3	-2.2161	-2.3240e+000
4	-1.8978	-5.4005e-001
5	-1.7631	-8.2719e-002
6	-1.7335	-3.7201e-003
7	-1.7321	-8.9497e-006
8	-1.7321	-5.2262e-011
9	-1.7321	-4.4409e-016

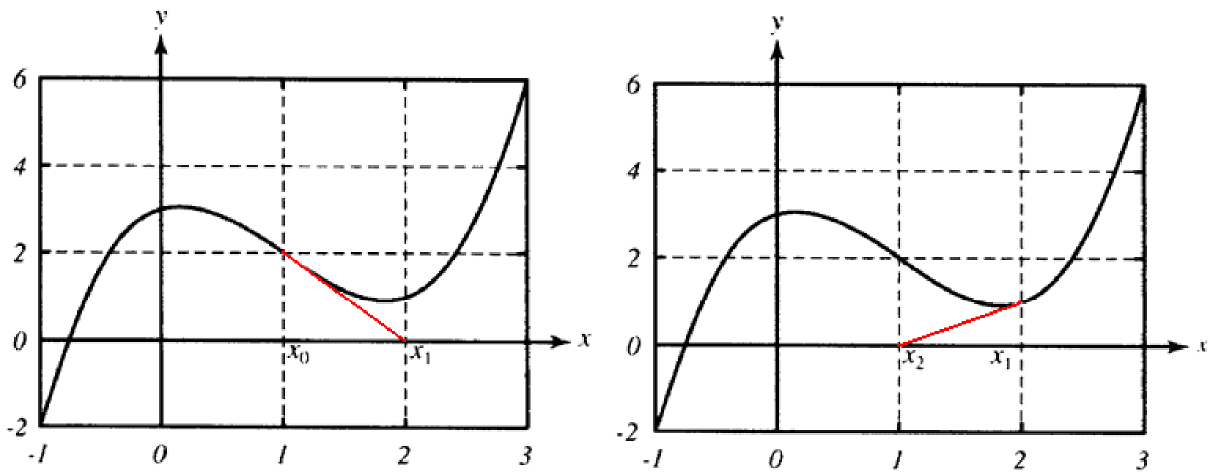
Run Time for Newton Method =1.1720 (sec)

牛頓法

- 牛頓法的收斂階數高，過程簡單，因此它經常是人們的第一選擇。但是此方法也會有出問題的時候。
- 在每一個零點的近似值處，其導數都不得為零，不然牛頓法就無法使用。
- 停止條件應結合最大迭代次數，以及每次近似值變化量的最小許可誤差。因為有發散的可能，最好也檢查計算所得之零點的變量是否過大，那可能意謂出了問題。

牛頓法

- 對某些問題及某些初始值，牛頓法可能出現震盪的結果。



牛頓法的震盪行為

牛頓法對重根之處理

- 在遇到重根時，牛頓法就沒有二階收斂了。但是，只要事先知道(或加以估計)重根的次數，我們可以修改牛頓法以保有收斂速度。

牛頓法對重根之處理實例

求 $f(x)=(x^2-5)^2(x^2-3)$ 之根

使用NewtonDoubleRoot.m

-------------- Newton Method for Double Roots-----------

step	x(k)	y(k)	Dx	C
0	2.0000	1.0000	5.0000e-001	0.0000e+000
1	2.5000	5.0781	-2.0968e-001	-8.3871e-001
2	2.2903	0.1354	-5.0832e-002	-1.1562e+000
3	2.2395	0.0005	-3.4066e-003	-1.3184e+000
4	2.2361	0.0000	-1.5559e-005	-1.3407e+000
5	2.2361	0.0000	-3.2479e-010	-1.3416e+000

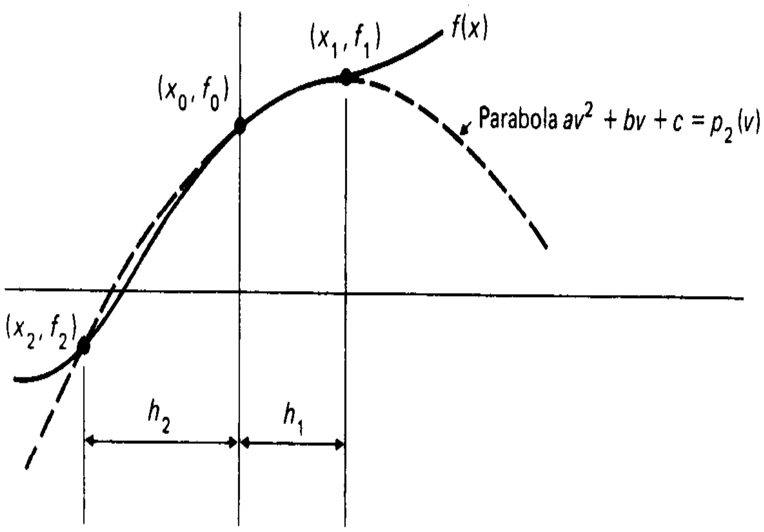
Newton method has converged

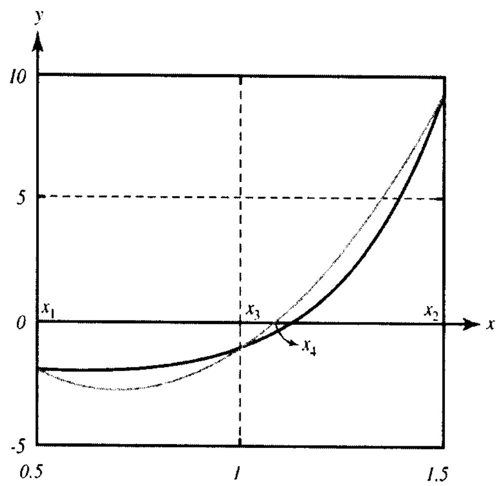
Run Time for Newton Method =1.4840 (sec)

Muller法

- 所謂Muller法就是用二次式來近似所要求解的函數。它的優點是即使初始估計值是實數，它也能產生複數零點的近似值。雖然它的公式比前面的各種方法都要複雜，但它的觀念很簡單。使用函數上的三點，我們可以找到一個通過此三點的二次式，然後求這個二次式的零點。開始時也可先找到兩個已知包夾住零點的點，再以這兩個點的中點當做第三點。當然，導出新近似值的公式之後，無須在每迭次代時實際求出此二次方程式。

Muller法





$y = x^6 - 2$ (黑色) 及 Muller 法所用的拋物線 (灰色)

Muller法

$$c_1 = \frac{f(x_2) - f(x_1)}{x_2 - x_1}, c_2 = \frac{f(x_3) - f(x_2)}{x_3 - x_2},$$

$$d_1 = \frac{c_2 - c_1}{x_3 - x_1};$$

令$s = c_2 + d_1(x_3 - x_2);$

$$x_4 = x_3 - \frac{2f(x_3)}{s + sign(s)\sqrt{s^2 - 4f(x_3)d_1}}$$

Muller法

求$f(x) = x^6 - 2$之根

使用MullerMethod.m

-------------- Muller Method -----------

Muller method has converged

step	x	y
1	0.5000	-1.9844e+000
2	1.5000	9.3906e+000
3	1.0000	-1.0000e+000
4	1.0779	-4.3172e-001
5	1.1170	-5.7635e-002
6	1.1225	7.6162e-004
7	1.1225	-4.7432e-007
8	1.1225	-4.8628e-013
9	1.1225	0.0000e+000

Run Time for Muller Method =1.1560 (sec)

Muller法

- 一般而言，Muller法對初始值不如牛頓法那麼敏感。
- 相對於牛頓法，Muller法只需要用到函數值，無須計算其導數。
- Muller法可以求得複數及實數零點。
- 但x若為多重零點時，則不能使用Muller法。

Homework #1

- 1.寫出改良式錯位法之程式
  - *畫出誤差收斂圖及所需之計算時間*
- 2.針對五種單函數求根之數值方法：Bisection、 Regula Falsi、Secant、Newton、Muller
  - *自行使用不同函數討論比較五種方法之收斂性。*
  - *比較五種方法之收斂速度。*
  - *改善程式內容使其執行速度更快(不要畫圖)*

求最小化之方法

- 求一個單變數函數最小值的問題。函數最小化的方法可以區分為兩類
  - *只需要用到函數值的*
  - *同時要用到函數值和函數之導數值的。*
- 方法
  - *黃金分割搜尋法(Golden-section Search)*
  - *Brent法*

黃金分割法

- 不需要用導數的求函數最小值的方法中，最簡單的是黃金分割搜尋法(Golden-section Search)。它可類比於求相問題中的二分法。

黃金分割法

- 我們希望以一序列的近似值趨近最小值，使得最小值被括在一個區間內，而每次迭代均將此區間縮小，同時讓計算函數值的次數盡可能的少。
- 在此一程序的每一階段，都會有一個括住最小值的三數組(a,t1,b)，而新的點t2是由t1向[a,t1]和[t1,b]中較長的一段伸入原長度的r<1/2倍。經由比較函數值f(t1)和f(t2)，可選出一組新的三數組並繼續整個程序，而搜尋區間的寬度，每次迭代減為原來的(1-r)≒0.618。r值的選取，是要能使得，不論新的區間是(a,t1,t2)或(t1,t2,b)，新搜尋區間的寬度都是一樣的。搜尋區間縮小的比值(1-r)是黃金此例r*=(1+√5)/2的倒數。(t2=t1+r*(b-t1))

黃金分割法

求$f(x)=(x-0.7)^2$之最小值

使用GlodSection.m

------------- Golden-section Search Method -----------

step	a	t1	t2	b	y1	y2
1	0.4000	0.6292	0.7708	1.0000	5.0155e-003	5.0155e-003
2	0.6292	0.7708	0.8584	1.0000	5.0155e-003	2.5078e-002
3	0.6292	0.7167	0.7708	0.8584	2.7951e-004	5.0155e-003
4	0.6292	0.6833	0.7167	0.7708	2.7951e-004	2.7951e-004
5	0.6833	0.7167	0.7374	0.7708	2.7951e-004	1.3975e-003
6	0.6833	0.7039	0.7167	0.7374	1.5576e-005	2.7951e-004
7	0.6833	0.6961	0.7039	0.7167	1.5576e-005	1.5576e-005
8	0.6833	0.6912	0.6961	0.7039	7.7882e-005	1.5576e-005
9	0.6912	0.6961	0.6991	0.7039	1.5576e-005	8.6804e-007
10	0.6961	0.6991	0.7009	0.7039	8.6804e-007	8.6804e-007
11	0.6961	0.6979	0.6991	0.7009	4.3402e-006	8.6804e-007
12	0.6979	0.6991	0.6998	0.7009	8.6804e-007	4.8374e-008
13	0.6991	0.6998	0.7002	0.7009	4.8374e-008	4.8374e-008
14	0.6991	0.6995	0.6998	0.7002	2.4187e-007	4.8374e-008
15	0.6995	0.6998	0.6999	0.7002	4.8374e-008	2.6958e-009
16	0.6998	0.6999	0.7001	0.7002	2.6958e-009	2.6958e-009
17	0.6998	0.6999	0.6999	0.7001	1.3479e-008	2.6958e-009
18	0.6999	0.6999	0.7000	0.7001	2.6958e-009	1.5023e-010
19	0.6999	0.7000	0.7000	0.7001	1.5023e-010	1.5023e-010
20	0.7000	0.7000	0.7000	0.7001	1.5023e-010	7.5116e-010
21	0.7000	0.7000	0.7000	0.7000	8.3721e-012	1.5023e-010

Golden-section search has converged

Run Time for Gold Section Search Method =3.6090 (sec)

Brent法

- Brent法是一種功能更強的求最小值的方法，它結合拋物線內插與黃金分割 搜尋。
- 它的基本觀念是，求通過函數三個點之拋物線的最小值，同時確保函數的最小值被括在一已知區間內，並且保持合理的步距大小。(例如：步距太大可能意味著拋物線在逐漸發散)

Brent法

- 通過(a,f(a))、 (b,f(b)) 及(c,f(c))三點之拋物線，其最小值之座標值為

$$x = b - \frac{(b-a)^2[f(b)-f(c)]-(b-c)^2[f(b)-f(a)]}{2[(b-a)(f(b)-f(c))-(b-c)(f(b)-f(a))]}$$

- Matlab的內建函數fminbnd就是使用Brent法。[x=fminbnd(myfun,x1,x2)]

Brent法

- x=fminbnd(inline('x.^3-2*x-5'),0,2)
- 得x=0.8165

使用MATLAB內建函數 fminbnd

```
>> x=fminbnd(inline('x.^3-2*x-5'),0,2)

x =

    0.8165
```


MATLAB求根之內建函數

- 求x0附近之零點
  - *[x,fval]=fzero(myfun,x0);*
  - *x0可為純量或向量*
- 求多項式之零點
  - *roots(p); % p:多項式之係數向量*
  - *p可為純量或向量*

其他方法

- Laguerre法
- Ridders法
- Brent-Dekker法
- Bairstow法

Laguerre法

- 在各種能夠保證收斂到多項式所有的根的方法中，Laguerre法是最簡單的一種，其能求得實數根、複數根、以及重根。
- Laguerre法假設所求的多項式根是與其他的根是分離的(然後將其它的根視為相等)。雖然此一假設看似輕率，但此方法對單一實根為三階收斂，對重根為線性收斂。對複數根的收斂特性則不確定。
- Lagurre法會用到根的估計值所對應的多項式值，及多項式的一階和二階導數值。此方法必須使用複數運算，而且即使所給的初始估計值為實數，此方法仍可能得到複數根。

Laguerre法

$x_1 = x_0 - a$

其中$a = \dfrac{n}{A(x_0) \pm \sqrt{(n-1)[nB(x_0) - A(x_0)^2]}}$

$$A(x) \equiv \frac{P'(x)}{P(x)} = \frac{1}{x-x_1} + \frac{1}{x-x_2} + \cdots + \frac{1}{x-x_n}$$

$$B(x) \equiv \frac{P'(x)^2}{P(x)^2} - \frac{P''(x)}{P(x)} = \frac{1}{(x-x_1)^2} + \frac{1}{(x-x_2)^2} + \cdots + \frac{1}{(x-x_n)^2}$$

假設其餘的根$x_2, x_3, \cdots, x_n$與目前估測值的距離都是b,則

$$A(x_0) = \frac{1}{a} + \frac{n-1}{b}, \qquad B(x_0) = \frac{1}{a^2} + \frac{n-1}{b^2}$$

Laguerre法

使用LagurreMethod.m

```
初始值X0=1
-------------- Laguerre Method -----------
 step     x0       x          y
   1    1.0000    1.1963    -1.9345e-002
   2    1.1963    1.2000    -1.8658e-007
   3    1.2000    1.2000    0.0000e+000

Laguerre method has converged

Run Time for Lagurre Method =1.0630 (sec)
```

Laguerre法

使用LagurreMethod.m

初始值X0=2

-------------- Laguerre Method -----------

step	x0	x	y
1	2.0000	2.0713	6.6897e-003
2	2.0713	2.0920	5.3565e-004
3	2.0920	2.0978	4.1203e-005
4	2.0978	2.0994	3.1348e-006
5	2.0994	2.0998	2.3778e-007
6	2.0998	2.1000	1.8021e-008
7	2.1000	2.1000	1.3655e-009
8	2.1000	2.1000	1.0346e-010
9	2.1000	2.1000	7.8337e-012

Laguerre method has converged

Run Time for Lagurre Method =1.9070 (sec)

Ridders法

- Ridders法是衍生自試位法，對所要求的根，它使用三個初始估計值。前兩個估計值必須括住根，第三個則是前面兩個值的中點。

Ridders法

已知初始估計值$x_1 < x_2$，而$x_m = (x_1 + x_2)/2$，

並獲得其函數值$y_1 = f(x_1)$、$y_2 = f(x_2)$、$y_m = f(x_m)$

則

$$x_n = x_m + (x_m - x_1)\frac{sign(y_1 - y_2)y_m}{s}$$

其中$s = \sqrt{y_m^2 - y_1 y_2}$

Ridders法

- Ridders法的一項優點是，每一階段產生的估計值，都一定在該步驟的區間(x_1,x_2)之內。
- 另一項優點是二階收斂。此方法每一步可使近似解的有效位數加倍。因為每一步需要計算兩次函數值，整個方法的階數為有√2。(牛頓法也是一樣，它每一步須計算*f(x)*與*f'(x)*。)
- 在實用上Ridders法非常強健，就可靠性與速度而言，它可相比於其它更複雜的方法(例如Brent-Dekker法)。

Brent-Dekker法

- Brent-Dekker法，發表於1973年，是對於1960年代在阿姆斯特丹數學中心所發展出的通用求根方法的改良版。
- 此方法的基本觀念是，由括住所要的根的一個區間開始，然後結合了保證收斂(但經常很慢)的二分法，和一種收斂快速(但不一定收斂)的方法。由括住根的區間(x_a,x_c)開始，用二分法或正割法求得第三個值x_b。
- 通常此方法使用反二次內插(Inverse Quadratic Interpolation)。也就是，此方法利用所要求根之函數上的三個點，將x表示為y的二次函數(即為反二次內插)。然後設y=0，就得到一個新的近似值。

Brent-Dekker法

已知用來內插的三個點(x_a, y_a)、(x_b, y_b)、(x_c, y_c)

$$x = \frac{(y-y_b)(y-y_c)}{(y_a-y_b)(y_a-y_c)}x_a + \frac{(y-y_a)(y-y_c)}{(y_b-y_a)(y_b-y_c)}x_b + \frac{(y-y_a)(y-y_b)}{(y_c-y_a)(y_c-y_b)}x_c$$

設$y=0$可得新的近似根x_n

$$x_n = \frac{y_b y_c}{(y_a-y_b)(y_a-y_c)}x_a + \frac{y_a y_c}{(y_b-y_a)(y_b-y_c)}x_b + \frac{y_a y_b}{(y_c-y_a)(y_c-y_b)}x_c$$

若

$$R = \frac{y_b}{y_c}，S = \frac{y_b}{y_a}，T = \frac{y_a}{y_c}$$

及

$$P = S[T(R-T)(x_c-x_b)-(1-R)(x_b-x_a)]$$

$$Q = (T-1)(R-1)(S-1)$$

$$\therefore x_n = x_b + P/Q$$

Bairstow法

- Bairstow法是找多項式的實數二次因式。它的基礎是，用於包含兩個非線性方程之方程組的牛頓法(見第7章)。就和用於單變數函數的牛頓法一樣，必須要有一個夠好的初始值。
- 若$a(x)=(x^2+px+q)Q(x)+Rx+S$，我們可將R和S視為是p和q的函數。我們希望求得可使R=0且S=0的p和q。Bairstow法使用綜合除法(Synthetic Division)計算R和S，以及它們對p和q的偏導數，以組成矩陣A和右側項向量r，使得Ax=r的解可在每次迭代更新p和q。

Bairstow法

求$f(x)=x^3-x-3$之根

使用BairstowMethod.m

-------------- Bairstow Method -----------

step	p	q
0	1.0000	1.0000
1	2.0000	2.0000
2	1.7000	1.8000
3	1.6719	1.7945
4	1.6717	1.7946
5	1.6717	1.7946

Bairstow method has converged

Run Time for Baitstow Method =1.5160 (sec)

得到根：
1.6717
-0.8358 + 1.0469i
-0.8358 - 1.0469i