

程式人《十分鐘系列》



用十分鐘搞懂

《電腦如何解方程式》

陳鍾誠

2016 年 8 月 31 日

話說

- 電腦很會計算！

舉例而言

- 假如我們要計算下列算式！

$$x^2 + 2x + 3$$

那麼只要寫個小程式

- 就可以輕易算完了！

```
function f(x) {  
    return x*x+2*x+3;  
}
```

=

$$x^2 + 2x + 3$$

就算裡面有複雜的函數

- 只要呼叫函式庫，通常也可以輕易解決

$$x^t + \sqrt{x} \sin(xt)$$

||

```
function f(x,t) {  
    return Math.pow(x,t)+Math.sqrt(x)*Math.sin(x*t);  
}
```

但是

- 你知道怎麼求解方程式的根嗎？

甚麼樣的方程式呢？

像是多項式

$$x^2 - 4x + 1 = 0$$

多變數方程式

$$x^2y - 3xy + 5 = 0$$

還有微分方程

$$\frac{\partial^2 z}{\partial x^2} + \frac{\partial^2 z}{\partial y^2} = 0$$

現在

- 就讓我們以程式人的直覺，先來想想到底怎麼解這些方程式。

$$x^2 - 4x + 1 = 0$$

$$x^2y - 3xy + 5 = 0$$

$$\frac{\partial^2 z}{\partial x^2} + \frac{\partial^2 z}{\partial y^2} = 0$$

就我能想到的方法中

- 第一個最簡單的方法是

暴力法

怎樣暴力呢？

- 就是把方程式裡每個變數，都從《最小到最大》算一遍。
- 然後看看是否有符合解答的結果！

舉例而言

- 假如我們要求解
- 而且假如我們知道《解答》在 ± 100 之間
- 那麼我們可以從 -100 到 $+100$ ，每隔 0.01 計算一次，如果有非常接近 0 的結果，那就是解答了。

以下程式碼就會印出解答

$$f(x) = x^2 - 4x + 1$$

```
function f(x) {  
    return x*x-4*x+1;  
}  
  
for (var x=-100; x<=100; x+=0.001) {  
    if (Math.abs(f(x)) < 0.00001)  
        console.log("x=", x, " f(x)=", f(x));  
}
```



而且這種方式非常強大

- 你只要將 $f(x)$ 寫成副程式，就可以列出任何的 $f(x)=0$ 的解答。


像是要求解下列方程式

$$\frac{\sin(x^2+2x)}{x^3} = 0$$

只要把 $f(x)$ 换掉


$$f(x) = \frac{\sin(x^2 + 2x)}{x^3}$$

```
function f(x) {  
    return x*x-4*x+1;  
}
```



```
function f(x) {  
    return sin(x*x+2*x)/x*x*x;  
}
```

```
for (var x=-100; x<=100; x+=0.001) {  
    if (Math.abs(f(x)) < 0.00001)  
        console.log("x=", x, " f(x)=", f(x));  
}
```

還是可以

- 列出相當符合條件的答案！

但是這個方法

- 有個重大的缺點！

這個重大的缺點就是

- 暴力法的速度比較慢！
- 當變數很多時，會非常的慢！

- 像是求解 $x^2 + y^2 - z = 0$

就會需要執行八千兆次 $f(x) = x^2 + y^2 - z$ 函數


$$((100 - (-100))/0.001)^3 = 200000^3$$

如果有六個變數，就需要算 (八千兆 * 八千兆) 次

所以

- 通常很少人用《暴力法》解決問題！

我們可以想出更好的方法

- 來求解方程式的根！

$$x^2 - 4x + 1 = 0$$

$$x^2y - 3xy + 5 = 0$$

$$\frac{\partial^2 z}{\partial x^2} + \frac{\partial^2 z}{\partial y^2} = 0$$

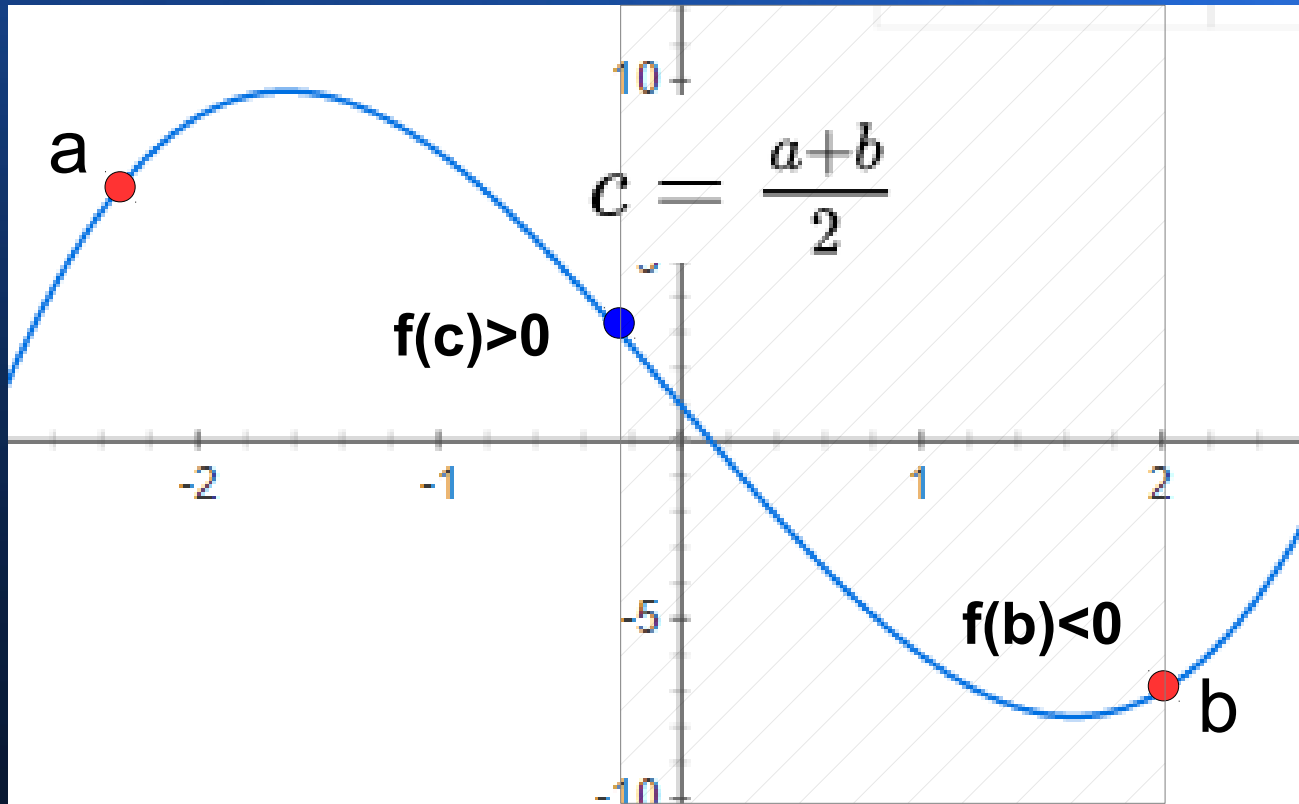
如果你曾經學過《演算法》

- 應該曾經使用過《二分搜尋法》

對於一個《連續函數》而言

- 假如我們知道兩個點 (a, b) ，其值 $f(a) > 0$ 且 $f(b) < 0$ ，這樣的話勢必有一個介於 (a, b) 之間的 c 值使得 $f(c) = 0$
- 假如我們每次都取 $c = \frac{a+b}{2}$ ，然後判斷要繼續搜尋哪一半的話，這樣我們就得到了一個《二分搜尋法》，可以較快速的找出 $f(x) = 0$ 的解答！

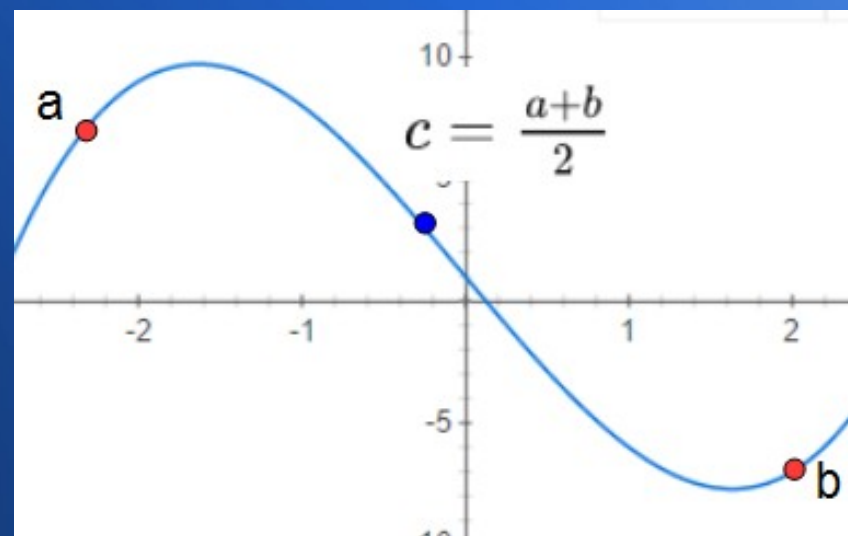
其想法圖示如下



計算完 $f(c)$ 之後，我們可以確定解答應該在 (c, b) 之間
所以接著用二分搜尋法繼續搜尋 (c, b) 區域。

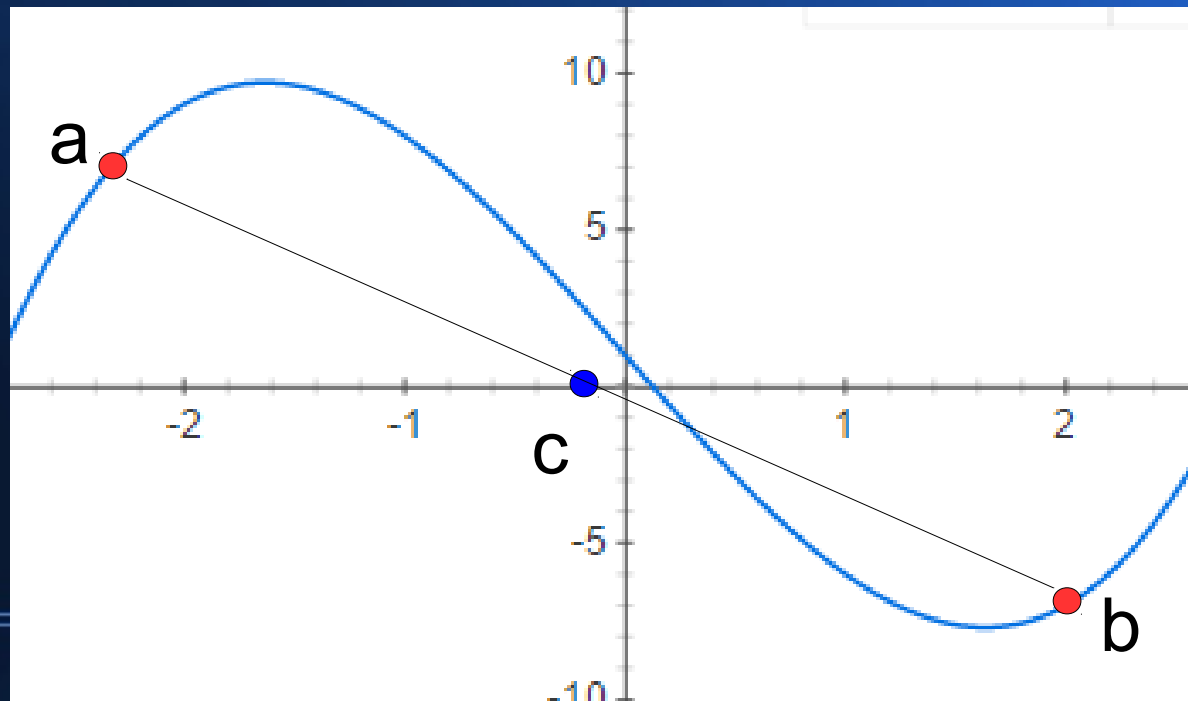
二分搜尋法求根的程式如下

```
function bsolve(f,a,b) {  
    var c = (a+b)/2;  
    if (Math.abs(a-b) < 0.00001)  
        return c;  
    if (f(c)*f(a)>=0)  
        return bsolve(f, c, b);  
    else  
        return bsolve(f, a, c);  
}
```



當然

- 我們也將 $c = \frac{a+b}{2}$ 改用另一種中間值
- 像是用《線性內插法》有時會更好！



以上的這種搜尋法

- 不管是二分搜尋法
- 或者是線性內插法
- 速度通常都不會太慢！

如果您學過演算法中的 Big O 複雜度概念

- 就會知道二分搜尋法的複雜度為 $O(\log n)$
- 但是在此問題中 n 應該改為兩個邊界值之間的差，也就是 $(b-a)$ ，所以複雜度是 $O(\log b-a)$

但是、二分搜尋法求根的一個小問題是

- 必須要先找出一組 (a, b) ，滿足 $f(a)$ 和 $f(b)$ 兩者正負號相反。

而且這種方法

- 並不是找出所有的根，而是只找出一個根
- 這和暴力法找範圍內全部的根有所不同！

現在、我們已經學過兩個方法了

- 而且這兩個方法都要先鎖定一個範圍
- 這種鎖定範圍的方法，稱為《界定法》
(Bracketing Method)。

接下來

- 讓我們看看另外一類的方法
- 這種方法不需要鎖定範圍
- 因此稱為《開放式方法》！

首先

- 讓我們看一個最簡單的開放式方法
- 這個方法稱為《爬山演算法》！

爬山演算法

- 是通用的《優化演算法》，也就是用來尋找最佳解的，並不只是用來解方程的。
- 假如尋找的是《極大值》，那麼就是《爬山演算法》，如果尋找的是《極小值》，那麼就變成了《下山演算法》。

而且爬山演算法這類的優化算法

- 很容易就可以用來找方程式的解。
- 因為我們只要最小化絕對值

$|f(x)-0|$ 就可以了！

爬山演算法的想法很簡單

- 就是先隨便選一個起點（例如 $x=0$ ）
- 然後每次都比較 $f(x)$ 和左邊的 $f(x-dx)$ 與右邊 $f(x+dx)$ 的值，假如左邊比較好，就往左邊走。
如果右邊比較好，就走右邊。
- 如果左邊右邊都比現在的 $f(x)$ 差，那麼現在的 x 就是個《區域最佳解》。

假如到區域最佳解時

- 還沒有找到 $|f(x)-0|$ 很接近零的解，那麼這次尋找就失敗了。
- 我們可以另選個起點繼續找，或者直接傳回尋找失敗。

以下是爬山演算法的程式碼

- 該程式碼求解下列方程式的根

$$x^2 - 4x + 1 = 0$$

```
var dx = 0.01;

function hillClimbing(f, x) {
  while (true) {
    if (f(x+dx) >= f(x))
      x = x+dx;
    else if (f(x-dx) >= f(x))
      x = x-dx;
    else
      return x;
  }
}

function f(x) {
  return -1*Math.abs(x*x-4*x+1);
}

hillClimbing(f, 0.0);
```

但是這個方法

- 速度並沒有很快，雖然還可以接受。
- 而且會常常落在《區域最佳解》出不來，因而沒有找到《方程式的解》。

所以

- 爬山演算法很少用來《解方程式》

接著、讓我們介紹另一個

- 用來解方程式的好方法！
- 這也是一個開放性方法。
- 而且不需要事先設定範圍。

這個方法稱為《迭代法》

話說《迭代法》

- 感覺非常神奇
- 但是說穿了很簡單！

迭代法的關鍵

- 可以說是一種《函數不動點》的尋找！

$$x=f(x)$$

$$x_2=f(x_1)$$

$$x_3=f(x_2)$$

...

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

所謂的不動點

- 就是 $x=f(x)$ 這樣一個方程式。
- 我們從 $k=0$ 開始反覆用 $x_{k+1}=f(x_k)$ 去找下一個 x_{k+1}
- 只要找到符合 $x_{k+1}=f(x_k)$ 的 x 時， x 基本上就定住了
- 這時我們找到的 x 就是 $x=f(x)$ 的一個解答！

問題是

- 如果我們並非想找 $f(x)=x$ 的解，而是 $f(x)=0$ 的解呢？
- 那該怎麼辦？

其實答案很簡單

- 只要修改方程式，想辦法讓 x 出現在其中一邊就行了。

舉例而言

- 假如我們想要找 $f(x)=0$ 的解
- 那麼我們可以對兩邊各加一個 x ，變成

$$f(x)+x = x$$

該等式仍然會成立。

- 這樣就可以進行迭代了！

當然、迭代的形式不只一種

- 對於 $f(x)=0$ ，以下都是可以用的迭代形式。

$$x = f(x) + x$$

$$x^2 = f(x) + x^2 \Rightarrow x = \frac{f(x) + x^2}{x}$$

$$3x^3 = f(x) + 3x^3 \Rightarrow x = \frac{f(x) + 3x^3}{3x^2}$$

於是、您只要選擇一個起點

- 像是 $x=3$ ，然後開始反複套用迭代公式，看看是否會收斂就行了！

假如我們的迭代公式是 $x=g(x)$

- 那麼只要隨便選一個起點，例如 $x_1=3$
- 然後用 $x_2=g(x_1)$ ， $x_3=g(x_2)$ ， \dots 一直算下去，直到收斂為止。

以下是一個迭代法的程式範例

- 用來尋找

$$x^2 - 4x + 1 = 0$$

的解！

```
function isolve(g, x) {  
  for (var i=0; i<100000; i++) {  
    if (Math.abs(x-g(x)) < 0.000001)  
      return x;  
    x = g(x);  
  }  
}  
  
function f(x) { return x*x-4*x+1; }  
function g(x) { return (f(x)+x*x)/x; }  
  
isolve(g, 3);
```

這種迭代法

- 其實幾乎可以用來解所有的方程式
- 最大的問題是《可能不會收斂》！
- 而且不同的迭代方法，收斂速度也常常有差異

在此我們舉一個簡單的例子

- 假如您想求某個數的平方根。

那麼可以用下列三種的迭代算式

範例：計算平方根

問題：請用迭代算法求 3 的平方根 ($x^2 = 3$)

迭代式：

$$1. x_{k+1} = \frac{3}{x_k}$$

$$2. x_{k+1} = x_k - \frac{1}{4}(x_k^2 - 3)$$

$$3. x_{k+1} = \frac{1}{2}\left(x_k + \frac{3}{x_k}\right)$$

結果：1 不收斂, 2, 3 收斂，3 收斂最快。

然後實作這三種方法

檔案：iterative.js

```
var f1=(x)=>3/x;  
var f2=(x)=>x-1/4*(x*x-3);  
var f3=(x)=>1/2*(x+3/x);  
  
x1=x2=x3=1;  
for (var i=0; i<20; i++) {  
    x1=f1(x1);    x2=f2(x2);    x3=f3(x3);  
    console.log("x1:", x1, "x2", x2, "x3", x3);  
}
```

這三種方法的收斂情形如下

```
D:\Dropbox\cccw\db\scl\code>node iterative
```

```
x1: 3 x2 1.5 x3 2
```

```
x1: 1 x2 1.6875 x3 1.75
```

```
x1: 3 x2 1.7255859375 x3 1.7321428571428572
```

```
x1: 1 x2 1.7311742305755615 x3 1.7320508100147274
```

```
x1: 3 x2 1.7319331764233397 x3 1.7320508075688772
```

```
x1: 1 x2 1.73203504452438 x3 1.7320508075688772
```

```
x1: 3 x2 1.7320486956592371 x3 1.7320508075688772
```

```
x1: 1 x2 1.732050524625521 x3 1.7320508075688772
```

```
x1: 3 x2 1.7320507696616354 x3 1.7320508075688772
```

```
x1: 1 x2 1.7320508024902694 x3 1.7320508075688772
```

```
x1: 3 x2 1.732050806888473 x3 1.7320508075688772
```

```
x1: 1 x2 1.7320508074777203 x3 1.7320508075688772
```

```
x1: 3 x2 1.7320508075566647 x3 1.7320508075688772
```

震盪

不收斂

收斂稍慢

收斂最快

因此

- 好的迭代算式可以让你上天堂！
- 不好的迭代算式会让你住牢房！

如果想要確定迭代法會收斂

- 必須要好好設計《迭代函數》
與《初始值》才行！

當然、有人可能會問

- 假如我想解的不是方程式，而是《方程組》的話，那該怎麼辦呢？

$$\begin{cases} 2x + y = 8 \\ x + y = 6 \end{cases}$$

$$\begin{cases} 3x + 2y + z = 39 \\ 2x + 3y + z = 34 \\ x + 2y + 3z = 26 \end{cases}$$

解線性微分方程組

$$\begin{cases} x_1' &= x_1 + x_2 \\ x_2' &= 2x_2 + x_3 \\ x_3' &= 3x_3 \end{cases}$$

其實這個問題

- 只要稍微轉換一下，就可以讓
《方程組變成單一的方程式》

假如您想求解下列方程組

$$\begin{array}{l} f(x)=0 \\ g(x)=0 \end{array}$$

- 那麼只要改寫為

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

就可以《將方程組變成方程式》了

只是這樣一來

- 線性的方程組就有可能變成
《二次的非線性方程式》了

這就是

- 用解方程式的方法來解方程組，所需要付出的代價。

不過迭代法確實是一個

- 很好的《數值方法》

可以用來解很多方程式。

這就是我們今天的

- 十分鐘系列！

希望您會喜歡！

我們下回見！

Bye Bye!