Homework No.02

Ruqi Feng

September 2022

1 Solve Cubic Equation

1.1 Description

We are solving the cubic equation

$$x^3 - 5x + 3 = 0 \tag{1}$$

In other words, we are searching for zeros of function $f(x) = x^3 - 5x + 3$.

We will be using different methods including the bisection method, Newton method and the hybrid method to different levels of precision.

It's difficult to find all zeros without knowing roughly where they are, so we sketch the function first. When $x > 5$ or $x < -5$, the term x^3 dominates, so the range of x axis is $(-5, 5)$, as is shown in Fig.1. In this way we can identify 3 zeros lying within $(-4, 0)$, $(0, 1)$, $(1, 2)$, respectively.

1.1.1 Bisection Method

In order to determine the 2 positive roots to 4 decimal precision, the initial brackets are chosen as $(0, 1)$ and $(1, 2)$ as mentioned above. The bisection method is used, where in each iteration, the value of f at the middle point of the bracket is calculated. Then the bracket is shrunk according to the value of f. The exit condition is set as: *the length of bracket is smaller than* 10^{-4}.

1.1.2 Newton-Raphson method

The roots found in 1.1.1 are "polished up" to 14 decimal precision using Newton method. The theory of Newton method is to expand the function $g(x)$ near its zero x_0 using Taylor series and discard terms higher than 1 order: $f(x_0) = f(x) + f'(x)(x_0 - x)$. Insert $f(x_0) = 0$ and perceive x_0 as the next iterative x to get:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \tag{2}$$

The exit conditions are set as $|x_{i+1} - x_i| < 10^{-14}$. Although the error is not rigorously under $|x_{i+1} - x_i|$, it is a convenient approximation of the solution

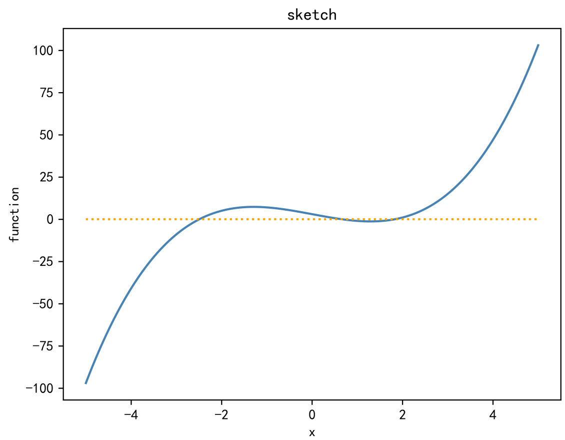


Figure 1: This is the sketch of f.

error. Additionally, the solution agrees with the results of other methods, which is to be shown in Sec.1.3.

There is a phenomenon worth mentioning: the underflow of $f(x)$. The reason is that Newton method converges quickly. When a step is made where f is so close to zero that it underflows, the difference in x_{i-1} and x_i may not have met the exit criteria. The program can be set so that but it's hard to estimate the error using the above approach.

1.1.3 Hybrid Method

The Newton method has some shortcomings including dividing by zero in ill conditions where $f'(x) = 0$, not converging due to oscillating solutions and so on. The hybrid method is used to avoid those problems while realizing better time complexity than the bisection method.

The hybrid methods takes the middle point of the bracket as the starting point in Newton method. The next x in Newton method is used to shrink the bracket. If the conditions are not valid like in the ill conditions mentioned above, a iteration of bisection method is used instead.

1.2 Pseudocode

The pseudocodes to solve the problems are presented below:

Algorithm 1 bisection method

Input: initial bracket $[a_0, b_0]$, function f
Output: root x_{final} and error δx
Requirement:$f(a_0) \cdot f(b_0) < 0$

if $f(a_0) < 0$ **then**
  $a_i, b_i \leftarrow b_0, a_0$
else
  $a_i, b_i \leftarrow a_0, b_0$
end if
while $|a_i - b_i| > 1 \times 10^{-14}$ **do**
  $mid \leftarrow \frac{a_i + b_i}{2}$
  if $f(mid) < 0$ **then**
    $b_i \leftarrow mid$
  else
    $a_i \leftarrow mid$
  end if
end while
$x_{root} \leftarrow \frac{a_i + b_i}{2}$, $error \leftarrow \frac{|a_i - b_i|}{2}$

Algorithm 2 Newton-Raphson method

Input: initial x coordinate x, function f
Output: root x_{root} and error δx
Requirement: $f'(x) \neq 0$ always

$count = 0$
while $count == 0$ or $|x - x_{last} > 10^{-14}|$ **do**
  $count \leftarrow count + 1$
  $x_{last} \leftarrow x$
  $x \leftarrow x - \frac{f(x)}{f'(x)}$
end while
$x_{root} \leftarrow x$, $error \leftarrow |x - x_{last}|$

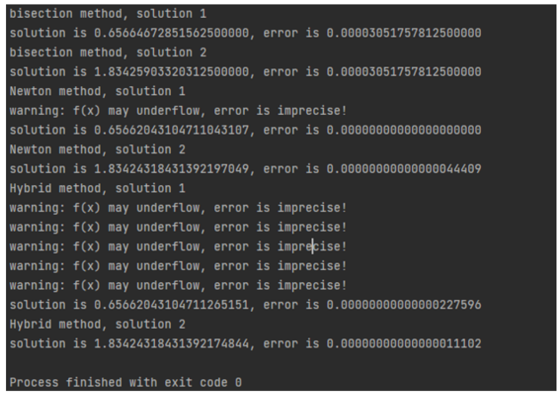
Algorithm 3 Hybrid method

Input: initial bracket $[a_0, b_0]$, function f
Output: root x_{final} and error δx
Requirement: $f(a_0) \cdot f(b_0) < 0$

if $f(a_0) < 0$ **then**
    $a_i, b_i \leftarrow b_0, a_0$
else
    $a_i, b_i \leftarrow a_0, b_0$
end if
while $|a_i - b_i| > 10^{-14}$ **do**
    $mid \leftarrow \frac{a_i+b_i}{2}$
    if $f'(mid) \neq 0$ **then**
        $x_{Newton} \leftarrow mid - \frac{f(mid)}{f'(mid)}$
        if $x_{Newton} \in [min(a_i, b_i), max(a_i, b_i)]$ **then**
            if $f(x_{Newton}) > 0$ **then**
                $a_i \leftarrow x_{Newton}$
            else
                $b_i \leftarrow x_{Newton}$
            end if
            continue
        end if
    end if
    if $f(mid) < 0$ **then**
        $b_i \leftarrow mid$
    else
        $a_i \leftarrow mid$
    end if
end while
$x_{root} \leftarrow \frac{a_i+b_i}{2}$, $error \leftarrow \frac{|a_i-b_i|}{2}$

1.3 Output Examples

Here are the results of the 3 problems. No input from the user is needed, because the conditions and parameters are fixed in the problem descriptions. In bisection method, the initial bracket is selected as in Sec.1.1.1, the two roots it produces is fed into the Newton method. The hybrid method uses the same initial bracket as the bisection method. The results are:



```
bisection method, solution 1
solution is 0.65664672851562500000, error is 0.00003051757812500000
bisection method, solution 2
solution is 1.83425903320312500000, error is 0.00003051757812500000
Newton method, solution 1
warning: f(x) may underflow, error is imprecise!
solution is 0.65662043104711043107, error is 0.00000000000000000000
Newton method, solution 2
solution is 1.83424318431392197049, error is 0.00000000000000044409
Hybrid method, solution 1
warning: f(x) may underflow, error is imprecise!
warning: f(x) may underflow, error is imprecise!
warning: f(x) may underflow, error is imprecise!
warning: f(x) may underflow, error is imprecise!
warning: f(x) may underflow, error is imprecise!
solution is 0.65662043104711265151, error is 0.00000000000000227596
Hybrid method, solution 2
solution is 1.83424318431392174844, error is 0.00000000000000011102

Process finished with exit code 0
```

Figure 2: A screenshot of the running program

Method Name	**Root Number**	**Root**	**Error**
bisection method	root 1	0.65664672851562500000	3×10^{-5}
	root 2	1.83425903320312500000	3×10^{-5}
Newton method	root 1	0.65662043104711043107	(underflow)
	root 2	1.83424318431392197049	4×10^{-16}
hybrid method	root 1	0.65662043104711487196	5×10^{-15}
	root 2	1.83424318431392174844	1×10^{-16}

It can be seen that bisection method gives roots to 4 decimals; Newton method "polished them up" to 14 decimals precision; and hybrid method gives the roots to 14 decimals precision. The underflow refers to the underflow of f as is discussed in Sec.1.1.2. The reason why some error are of 10^{-16} scale is similar: $f(x)$ approaches 0 quicker than δx does. Fig.1.3 is a screenshot of the program.

The program is run at Python 3.9.11, and packages numpy 1.21.5 and matplotlib 3.5.1 are used. The modules are packed in the executable file q1.exe, so there should be no need for additional installation of dependencies.

2 Global Minimum in Two Dimensions

2.1 Problem description

The problem is to find the global minimums of function

$$f = \sin(x + y) + \cos(x + 2y) \tag{3}$$

2.2 My approach: Genetic algorithm

First, we use the periodic property of the function to simplify the problem. Because $f(x, y) = f(x+2\pi, y)$ and $f(x, y) = f(x, y+2\pi)$, any (x', y') exceeding the range $(0, 2\pi)$ can be converted into points in this domain:

$$\{(\xi, \eta)|\xi \in [0, 2\pi] \text{ and } \eta \in [0, 2\pi]\} \tag{4}$$

Any minimums in the x-y plane must have an identical minimum in this domain. The optimization problem is thus restricted to a finite variable space. Plot f in this domain in Fig.3, and we can have a rough impression on the problem.

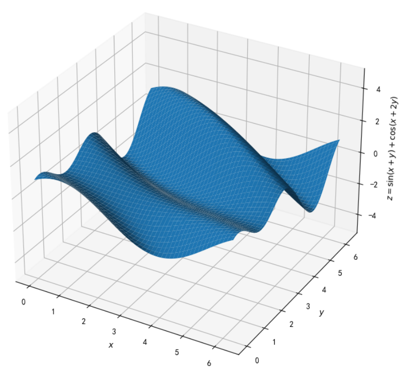


Figure 3: Sketch the target function in Q2

How do we choose the appropriate initial point so that the gradient descent method can find global minimum? Although in this case, all minimums are the same and therefore a gradient descent / Newton method would work well, my program use the genetic algorithm instead. The advantage is that no derivative is required and it is also suitable for more complex target functions.

The spirit of the genetic algorithm is to mimic evolution processes in the nature. Individuals that best fit the environment have the biggest chance to reproduce, passing their genes to the next generation. Therefore the genes associated to ”fitting the environment” become popularized in the herd. In our optimization scenario, the ”herd” is a batch of points, their ”genomes” being the coordinates (x, y), and the fitness is quantified using a merit function.

Before starting, f need some minor embellishment to be used as the merit function because f can be either positive or negative. We want the value of the merit function to be proportional to the probability of passing its gene, so the merit function is defined as:

$$merit(x, y) = \frac{1}{f(x, y) + 2.01} \tag{5}$$

so that the lower value of f, the better fitness is. Because $\sin\xi > -1$ and $\cos\xi > -1$, $f > -2$, the 2.01 in the denominator ensures the fitness to be positive.

There are 3 main steps in genetic algorithm, selection, crossing and mutation. Selection means the parents are picked from the current herd by probability proportional to their merits. The i-th individual has probability of

$$p_i = \frac{merit_i}{\sum_j merit_j} \tag{6}$$

to be picked. Cross means when 2 parents are selected, their genes merge and produce one individual of the next generation. This program simply take the average of the coordinates of the 2 parents.

Mutation means that the coordinates of the next generation receives a random perturbation so that the herd will not be stuck in local minimums. However in or case, because the local minimums are not ”attractive” enough, the algorithm works well even without mutations.

The select-cross-mutate process is repeated for herd_size times to generate the next generation. The first generation has HERDSIZE individuals randomly scattered across the allowed domain according to uniform distribution. After MAXGEN generations, the procedure is terminated.

The first generation is produced by randomly scattering points in the allowed region.

2.3 Pseudocode

The pseudocode of the genetic algorithm used in the program is shown below.

Algorithm 4 genetic algorithm

Input: function f
Output: coordinate of the minimum (x, y) and the minimum value f_{min}

MAXGEN$\leftarrow$ 10, HERDSIZE$\leftarrow$ 1000
$meritFunc \leftarrow$ lambda x, y: $\frac{1}{f(x,y)+2.01}$
$generation \leftarrow 0$
$herd \leftarrow$ random array of size(HERDSIZE, 2), each element $\in (0, 2\pi)$
while $generation$ <MAXGEN **do**
    $generation \leftarrow generation + 1$
    for i in range(HERDSIZE) **do**
        for j in range(HERDSIZE) **do**
            $probs[j] \leftarrow meritFunc(herd[j, 0:2])$
        end for
        $randnum \leftarrow$ random number in $(0, 1)$
        $probAccum \leftarrow 0$
        for j in range(HERDSIZE) **do**
            $probAccum \leftarrow probAccum + probs[j]$
            if $probAccum > randnum$ **then**
                $parent1 \leftarrow herd[j, :]$
                break
            end if
        end for
        $randnum \leftarrow$ random number in $(0, 1)$
        $probAccum \leftarrow 0$
        for j in range(HERDSIZE) **do**
            $probAccum \leftarrow probAccum + probs[j]$
            if $probAccum > randnum$ **then**
                $parent2 \leftarrow herd[j, :]$
                break
            end if
        end for
        $nextherd[i, :] \leftarrow \frac{parent1+parent2}{2}$
        $nextherd \leftarrow nextherd+$ random array
    end for
    $herd \leftarrow nextherd$
end while
$(x_{out}, y_{out}) \leftarrow$ average of last $herd$
$f_{min} = f(x_{out}, y_{out})$

2.4 Output Examples

The result after 9 generations is shown in the table below.

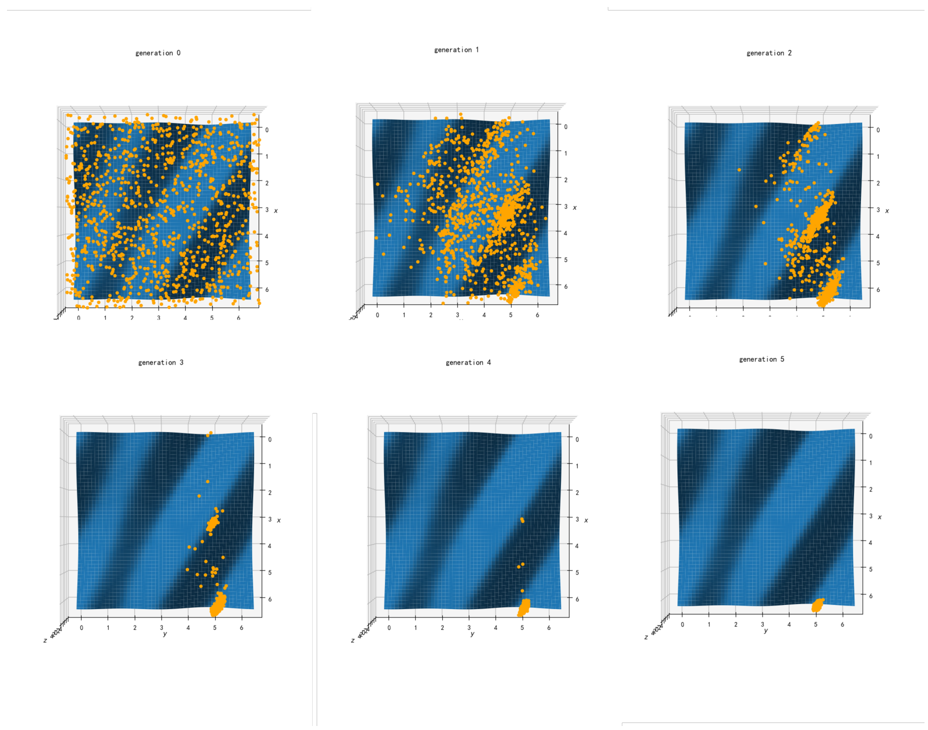


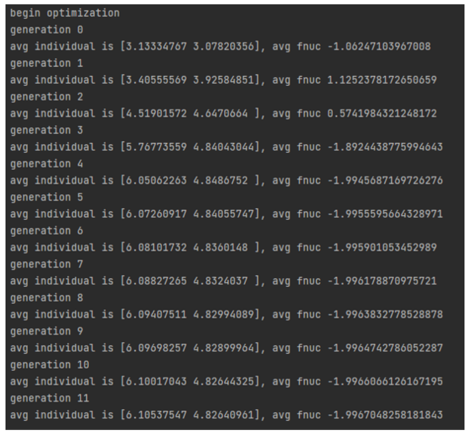
Figure 4: The first 6 generations in a run

mutation intensity	(x, y)	f_{min}
0.1	(6.09698257, 4.828994089)	-1.99647428
0.01	(6.02453092, 4.88309938)	-1.99271223
0	(0.19657633, 4.59602594)	-1.99613131

There are several global minimums, while the minimum of f is about -2. Due to random initialization and optimization process, results in different runs can be different even if the parameters are identical. For a more illustrative demo, different generations are plotted as shown in Fig.4. The mutation is 0.1 here.

The results is slightly deviated from the theoretical minimums mainly because genetic algorithm does not work well in converging to the exact minimum, especially in this case where the function is "flat" near the minimums.

The output in the terminal in one run is shown in Fig.5. When opening q2.exe, plots will be shown and saved each generation to better show the optimization process. The program is run at Python 3.9.11, and packages numpy 1.21.5 and matplotlib 3.5.1 are used. The modules are packed in the executable file q1.exe, so there should be no need for additional installation of dependencies.



```
begin optimization
generation 0
avg individual is [3.13334767 3.07820356], avg fnuc -1.06247103967008
generation 1
avg individual is [3.40555569 3.92584851], avg fnuc 1.1252378172650659
generation 2
avg individual is [4.51901572 4.6470664 ], avg fnuc 0.5741984321248172
generation 3
avg individual is [5.76773559 4.84043044], avg fnuc -1.8924438775994643
generation 4
avg individual is [6.05062263 4.8486752 ], avg fnuc -1.9945687169726276
generation 5
avg individual is [6.07260917 4.84055747], avg fnuc -1.9955595664328971
generation 6
avg individual is [6.08101732 4.8360148 ], avg fnuc -1.995901053452989
generation 7
avg individual is [6.08827265 4.8324037 ], avg fnuc -1.996178870975721
generation 8
avg individual is [6.09407511 4.82994089], avg fnuc -1.9963832778528878
generation 9
avg individual is [6.09698257 4.82899964], avg fnuc -1.9964742786052287
generation 10
avg individual is [6.10017043 4.82644325], avg fnuc -1.9966066126167195
generation 11
avg individual is [6.10537547 4.82640961], avg fnuc -1.9967048258181843
```

Figure 5: Output in terminal

3 Temperature Dependence of Magnetization

3.1 Problem Description

Determine M(T) the magnetization as a function of temperature T for simple magnetic materials.

3.2 Analysis and Solution

The spontaneous magnetization can be modeled using this equation:

$$m(t) = \tanh \frac{m(t)}{t} \tag{7}$$

where $m = \frac{M}{N\mu}$ is reduced magnetization, and $t = \frac{k_B T}{N\mu^2\lambda}$ is the reduced temperature. For different t in the range $(0, 2)$, m is solved and is plotted.

The root-searching algorithm used here is bisection method. Because $\tanh(x) < 1$, the function

$$f(m) = \tanh \frac{m}{t} - m \tag{8}$$

is always positive at $m = 1$ and 1 is one end of the initial bracket. To determine the other end of the initial bracket where $f(m) < 0$, the program then searches m_{low} in $(0, 1)$ with step length 0.01.

After finding the positive solution, the opposite solution $-m$ is also acquired, and 0 is a solution.

3.3 Pseudocode

Algorithm 5 Solve for $m(t)$

> **Output:** pairs of $[m, t]$ at different t
>
> **for all** t in 1000 uniformly spaced points $\in (0.01, 2)$ **do**
> $x \leftarrow 0$
>   **while** x ¡ 1 **do**
> $x \leftarrow x + 1$
>   **if** $f(x) > 0$ **then**
> $bracket \leftarrow x, 1$
>   Find root using bisection method (see Algo.1)
>   store solution for current t
>   **end if**
>   **end while**
> **end for**
> Print $m(t) = m$, $m(t) = -m$ and $m(t) = 0$ to t

3.4 Output Example

Fig.6 is the plotted $m(t)$ figure and Fig.7 is a screenshot of the terminal. It

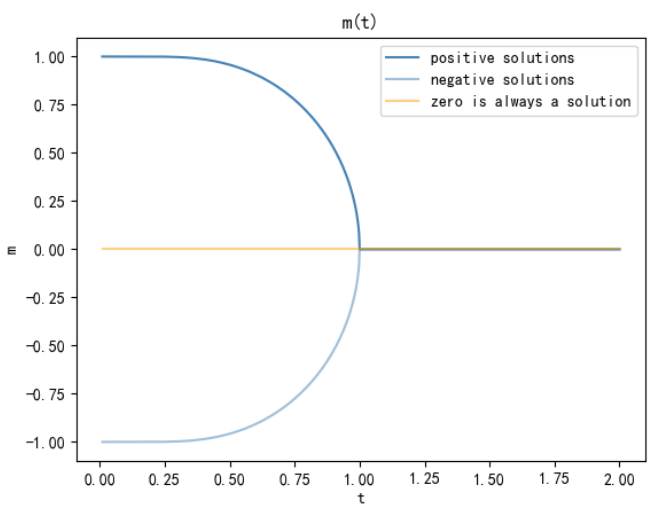
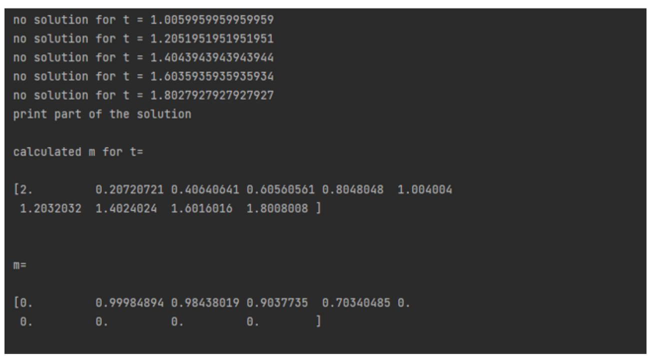


Figure 6: $m(t)$ is plotted to t

can be seen that only when $t < 1$ do the positive and negative solutions exist. This means spontaneous symmetry breaking happens only in low temperatures. When the temperature is above the critical point, the thermal fluctuation dominates and no magnetization will occur in this model.



```
no solution for t = 1.0059959959959959
no solution for t = 1.2051951951951951
no solution for t = 1.4043943943943944
no solution for t = 1.6035935935935934
no solution for t = 1.8027927927927927
print part of the solution

calculated m for t=

[2.         0.20720721 0.40640641 0.60560561 0.8048048  1.004004
 1.2032032  1.4024024  1.6016016  1.8008008 ]


m=

[0.         0.99984894 0.98438019 0.9037735  0.70340485 0.
 0.         0.         0.         0.        ]
```

Figure 7: Screenshot of the terminal when running the program