

# Numerical Methods for Natural Sciences: Foundation Topics

## January - April semester, 2024

Raghunathan Ramakrishnan

ramakrishnan@tifrh.res.in

Tata Institute of Fundamental Research Hyderabad

Hyderabad, India

*Course material: <https://github.com/raghurama123/nm2024>*

# Content

		Page
References	...	3
Chapter 1: Solutions of equations by fixed-point iteration	...	4

# References

## Textbook

- David G. Moursund, Charles S. Duris, *"Elementary Theory and Application of Numerical Analysis"*, Dover Publishers (1988).

## General References

- Samuel D. Conte, Carl de Boor, *"Elementary Numerical Analysis: An Algorithmic Approach"*, McGraw-Hill (1981).
- Lars Elden, Linde Wittmeyer-Koch and Hans Bruun Nielsen, *"Introduction to Numerical Computing"*, Overseas Press (2006).
- Anthony J. Pettofrezzo, *"Introductory Numerical Analysis"*, Dover Publishers (1984).
- W. Boehm, H. Prautzsch, *"Numerical Methods"*, Universities Press (2003).
- Richard L. Burden, J. Douglas Faires, *"Numerical Analysis"*, Cengage Learning (2011).
- Ward Cheney, David Kincaid, *"Numerical Methods and Computing"*, Cengage Learning (2013).
- Anne Greenbaum, Timothy P. Chartier, *"Numerical Methods"*, Princeton University Press (2012).
- William Bo Rothwell, *"Linux for Developers"*, Pearson (2018). *See the chapters about GitHub.*
- Numpy and Scipy Documentation, <https://docs.scipy.org/doc/>

# Chapter 1: Solutions of equations by fixed-point iteration

# Some definitions

## Program, algorithm, elementary operation

- A *program* is a set of algorithms along with statements for user interaction (*i.e.* input/output). An *algorithm* is a set of pre-defined operations to convert an input to an output.
- For a given problem, there may not be a unique way to write a program, or the algorithms involved, or even the elementary operations involved (that comprise the algorithms).

## Types of Numerical Methods: Direct, Iterative, and Heuristic

- *Direct methods* are predefined recipes (*i.e.* fixed algorithms with fixed elementary steps) for solving a problem. In this case, the error in the final result is only due to finite computer *precision* (*i.e.* rounding-off the numbers involved).
  - The standard formula for finding the root of a quadratic equation is a direct method.
  - It is often the case that for a given problem, a direct method may not exist (either it has not been found, or it may not even exist mathematically). A theorem in algebra says that there is no direct-method for finding a root of a polynomial of degree  $\geq 5$ .
  - We will later see that Gaussian elimination is a direct method for solving systems of linear equations.
- *Iterative methods* require an initial guess for the final solution of a problem. This value will be given as an input to an algorithm which will be repeated until its input and output are the same.
  - Newton-Raphson method for finding the solution of non-linear equations is an iterative method.
  - Iterative methods can be shown (using a theorem) that a solution (if it exists) can be found as a limit.
- *Heuristic methods* are methods developed based on experience. These methods are not guaranteed to give a solution.
  - Simplex method (Nelder-Mead method) for optimization is a heuristic method.

# Square root by fixed-point iteration

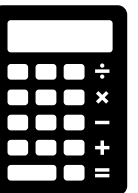
## Square root of a real number

- The square root of a number  $A$  can be determined using the formula  $g(x) = \frac{1}{2} \left( x + \frac{A}{x} \right)$ , and using it successively. The procedure involves starting with an initial value  $x_0$  and determining  $g(x_0)$  using the formula. Now, one sets  $g(x_0)$  as  $x_1$ , and continue the process until  $g(x_n)$  (the output to the formula) is the same as  $x_n$  (the input).

## Example

- Find the square-root of 5 by applying fixed-point iteration. Let's begin with  $x_0 = 1$ .

$n$	$x_n$	$g(x_n) = (x_n + 5/x_n)/2$
0	1.0	3.0
1	3.0	2.333...
2	2.333...	2.238...
		...
		2.236068



- You can repeat these steps by a manual calculation with the help of a pocket-calculator, smart phone, or a computer.
- Now, you are ready to try these steps in Python. Try the notebook ([Chapter01\\_Fixed\\_Point\\_Iteration.ipynb](#)) provided in the course repository<sup>1</sup>.

<sup>1</sup> <https://github.com/raghurama123/nm2024>

## for loop

- You can refine the simple steps given in the notebook into a neat program as follows.

```
# Number, whose square root we want to find
A=5

# Maximum number of steps
MaxIter=4

# Start with a guess
xold=1

# Iterate
for n in range(MaxIter):
    xnew=g(xold,A)
    print(n,xold,xnew)
    xold=xnew
```

```
0 1 3.0
1 3.0 2.3333333333333335
2 2.3333333333333335 2.238095238095238
3 2.238095238095238 2.2360688956433634
```

## Exercise

- Learn about Python's built-in function **range**, and **for** loops in Python.
- Learn what a Python module is. In the following, we are calling a procedure (**sqrt**) from a module (**numpy**). Learn about how to use an alias for a module or a procedure while importing in your code.

```
import numpy
print(numpy.sqrt(5))
```

```
2.23606797749979
```

## Pretty print

- You should always use *formatted strings* to display any output.

```
# Number, whose square root we want to find
A=5

# Maximum number of steps
MaxIter=4

# Start with a guess
xold=1

# Iterate
for n in range(MaxIter):
    xnew=g(xold,A)
    fstr = "{:5d} {:15.8f} {:15.8f}".format(n, xold, xnew)
    print(fstr)
    xold=xnew
```

0	1.00000000	3.00000000
1	3.00000000	2.33333333
2	2.33333333	2.23809524
3	2.23809524	2.23606890

- How does this code differ from the one given in the previous page?
- You may also try the following two lines to print the output in the same format.

```
output = "{val1:5d} {val2:15.8f} {val3:15.8f}"
print(output.format(val1=n, val2=xold, val3=xnew))
```

- In this code, we have limited the number of iterations to a fixed number. Suppose we do not know how many iterations we will require. How will you modify this code by introducing a **while** loop?



# Fixed-point iteration: Statement, Algorithm, and Theorem

## Statement of the method

- The solution of  $f(x) = 0$ ; where  $f(x)$  is single-valued, can be determined by rewriting the equation in the form  $x = g(x)$  and starting with an initial value  $x_0$ . Then, under certain conditions, the sequence  $x_1 = g(x_0), x_2 = g(x_1), \dots$  will converge to one of the solutions of  $x = g(x)$ , denoted as  $x^* = \lim_{n \rightarrow \infty} x_n$ .

## Algorithm

1. Define the function  $g(x)$ .
2. Initialize  $x$  to a real number, let's call it  $x_0$ .
3. Generate the sequentially improved estimates for the root through the formula  $x_{n+1} = g(x_n)$ .
4. Stop when  $|x_{n+1} - x_n|$  is below a threshold and return  $x_n$  as the solution  $x^*$ .

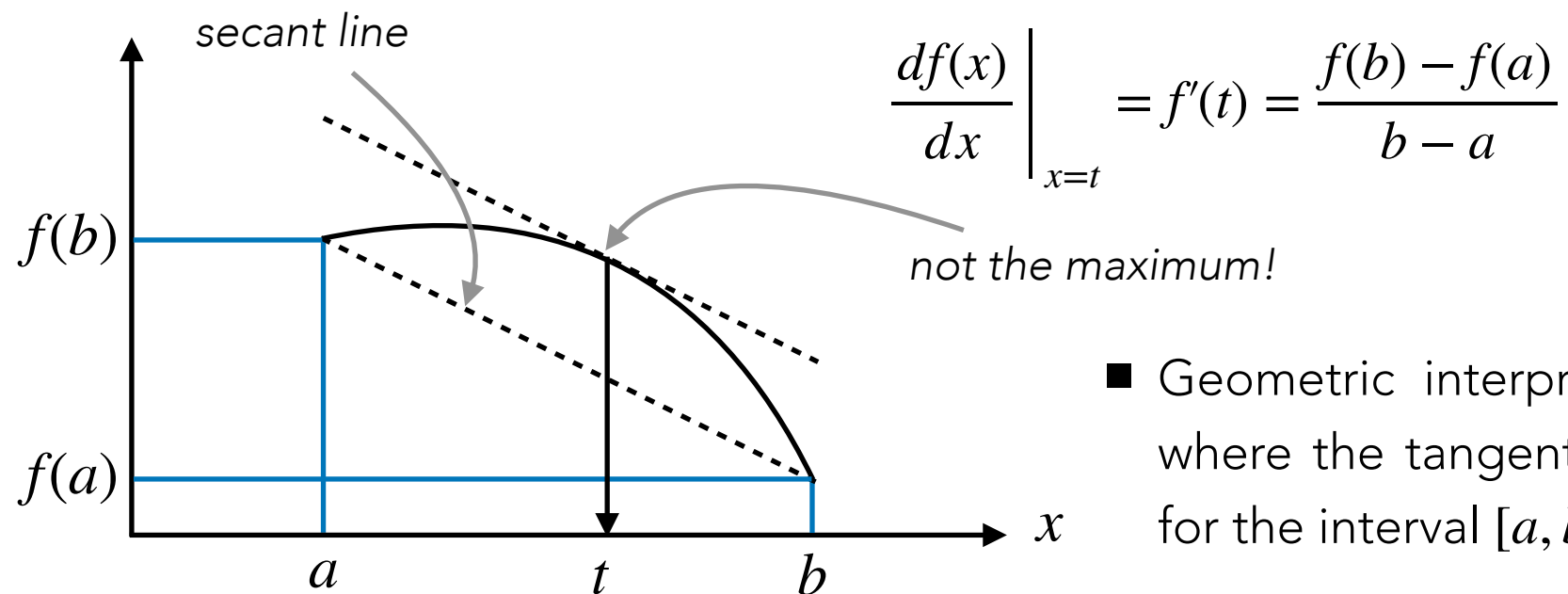
## Theorems

- **Theorem 1:** The equation  $x = g(x)$  has a solution for some value of  $x$ , which we call  $x^* \in [a, b]$ , provided  $g(x) \in C([a, b])$  has its range contained in the same closed interval  $[a, b]$ .
  - The theorem does not tell us if we will have one solution (a unique solution) or the equation has multiple solutions.
- **Theorem 2 (Uniqueness):** If the same  $g(x)$  (obeying conditions stated above) further satisfies  $|g^{(1)}(x)| \leq k$  for a constant  $0 \leq k < 1$  in the open interval  $(a, b)$ , then  $x = g(x)$  has exactly one root in  $[a, b]$ .
  - The previous theorem stated that the domain of  $g(x)$  is  $[a, b]$  (and its range is also contained in this interval). Hence, if  $x_0 \in [a, b]$ , then  $g[x_0] \in [a, b]$ , and by induction  $x_1, x_2, \dots, x^* \in [a, b]$ .
  - In both the theorems, instead of the closed interval  $[a, b]$ , we can consider a symmetric interval  $[x^* - \delta, x^* + \delta]$ , where  $x^*$  is a solution of the  $x = g(x)$ , and  $\delta > 0$ .

# Proof of the uniqueness theorem

## Mean-value theorem

- If  $f(x) \in C([a, b])$ , and if  $f(x)$  is differentiable in  $(a, b)$ , then there is a point  $t \in (a, b)$  such that



- Geometric interpretation: There is a point  $t \in (a, b)$  where the tangent of  $f(x)$  is parallel to its secant line for the interval  $[a, b]$ .

## Proof-by-contradiction for the uniqueness theorem

- Suppose  $g(x)$  defined in Theorem 2 has two roots  $p$  and  $q$  in  $[a, b]$ . At the roots, we have  $g(p) = p$ , and  $g(q) = q$ .
- Hence, according to the mean-value theorem there must be a point  $t \in [p, q]$  such that
$$g'(t) = \frac{g(q) - g(p)}{q - p} = \frac{q - p}{q - p} = 1.$$
- This violates our assumption that  $|g^{(1)}(x)| \leq k$  for some constant  $0 \leq k < 1$  in the open interval  $(p, q)$ .

# Convergence of fixed point iteration

## Upper bound for the error in each iteration

- In the Textbook, Theorem1-5-3 and its corollary are proved using the mean-value theorem. This theorem states that at the  $n$ -th iteration, the upper bound for the error in the root is given by,  $|x_n - x^*| \leq k^n(b - a)$ .
- So, for a given problem, if we select the domain  $[a, b]$  appropriately, and find  $k$ , we can estimate the maximum error that one can expect in each iteration.
- With the known information,  $|g^{(1)}(x)| \leq k$  and  $0 \leq k < 1$ , we can find an approximate value of  $k$ . To do this, we can use  $(x_0, g_0)$ , and  $(x_1, g_1)$  and estimate  $k$  as the finite derivative  $(g_1 - g_0)/(x_1 - x_0)$ .

## Example

- Let's find the error bound for finding the square-root of 5. Let's begin with  $x_0 = 1$ , and assume the domain to be  $[1, 3]$ . The following results are obtained using the notebook ([Chapter01\\_Fixed\\_Point\\_Iteration.ipynb](#)) provided in the course repository.

n	x_n	x_(n+1)	x_n - x^*	upper-bound
0	1.00000000	3.00000000	1.23606798	
1	3.00000000	2.33333333	0.76393202	
2	2.33333333	2.23809524	0.09726536	0.11111111
3	2.23809524	2.23606890	0.00202726	0.03703704
4	2.23606890	2.23606798	0.00000092	0.01234568
5	2.23606798	2.23606798	0.00000000	0.00411523
6	2.23606798	2.23606798	0.00000000	0.00137174
7	2.23606798	2.23606798	0.00000000	0.00045725
8	2.23606798	2.23606798	0.00000000	0.00015242
9	2.23606798	2.23606798	0.00000000	0.00005081

actual error  
in each iteration

Error bound estimated using  $k = 0.3333$

$0 < k \leq 1$

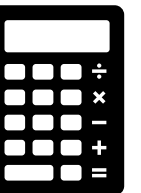
## Choice of $g(x)$ and initial guess

- To find the square root of a number, the equation we want to solve is  $x = \sqrt{A} \rightarrow x^2 = A \rightarrow x^2 - A = 0$ . This equation is of the known form  $f(x) = 0$ . To apply fixed-point iteration (see p.9), we have to rewrite the equation in the form  $x = g(x)$ . Let's look at some choices of  $g(x)$ .
- Choice-1:  $x^2 - A = 0 \rightarrow x^2 = A \rightarrow x = A/x$ . Now,  $g_1(x) = A/x$ .
- Choice-2:  $x^2 - A = 0 \rightarrow x^2 = A \rightarrow x = A/x \rightarrow 2x = x + A/x \rightarrow x = (x + A/x)/2$ . Now,  $g_2(x) = (x + A/x)/2$ .

### Example

- Let's find the square-root of 5 using  $x_0 = 1$  and  $g_1(x)$ .

$n$	$x_n$	$g(x_n) = 5/x_n$
0	1.0	5.0
1	5.0	1.0
2	1.0	5.0
	...	



### Exercise

- To prevent the code from running indefinitely, one can estimate  $k$  and decide whether the loop should run beyond iteration,  $n = 1$ . For  $g_1(x) = A/x$ , we can see that  $k = 1$  violating our assumption about  $g(x)$ .
- Include such a condition in your code.
- The equation  $e^{-x} + x/5 - 1 = 0$  is encountered in the derivation of Wien's displacement law. Write this equation in the form  $x = g(x)$  and apply fixed-point iteration. Analyse how the choice of initial value affects the convergence to a desired value of  $x^*$  for  $g_1(x) = 5(1 - e^{-x})$  and  $g_2(x) = -\log(1 - x/5)$ .

# Bisection method

## Statement of the method

- If  $f(x) \in C[a, b]$ , and  $f(a)f(b) < 0$ , then the equation  $f(x) = 0$  has a solution,  $x^* \in [a, b]$ .

## Algorithm

1. Determine  $f(a)$  and  $f(b)$ , verify  $\text{sign} f(a) \neq \text{sign} f(b)$ . If  $\text{sign} f(a) = \text{sign} f(b)$ , exit.
2. Determine  $c = (a + b)/2$ .
3. If  $|a - b| < \text{threshold}$ ,  $x^* = c$ ; else continue.
4. Determine  $f(c)$  and  $f(a)f(b)$ .
5. If  $\text{sign} f(c) = \text{sign} f(a)$ , set  $a = c$ , else if  $\text{sign} f(c) = \text{sign} f(b)$ , set  $b = c$ .
6. Repeat 2, then the sequence  $c_0, c_1, \dots, c_n \in [a, b]$  will converge to  $x^*$ .

## Upper bound for the error in each iteration

- The upper bound for the error in the root is given by,  $|x_n - x^*| \leq (b - a)/2^n$ .
- Bisection method is guaranteed to find  $x^*$  how ever with a slower convergence compared to other iterative methods. Often bisection method is used to find an initial guess for a solution in an interval.

## Exercise

- Write a Python program to find the square-root of 5 by solving the equation  $x^2 - 5 = 0$  using the bisection method in the interval  $[1, 3]$ . How many iterations does it take to reach  $|a - b| < 0.001$  and compare with the number of iterations taken in fixed-point-iteration method to reach  $|x_{n+1} - x_n| < 0.001$ .

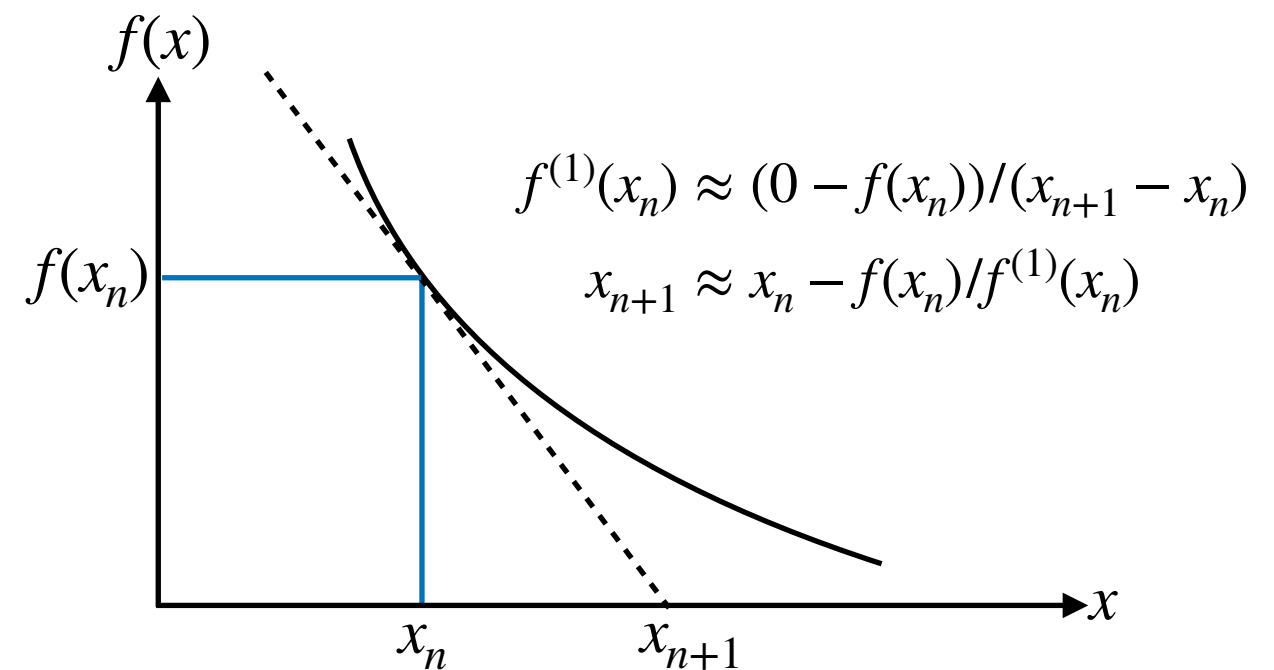
# Newton-Raphson method

## Statement of the method

- Newton-Raphson method is a variant of the fixed-point iteration method for solving  $f(x) = 0$ , in the form  $x = g(x)$  and starting with an initial value  $x_0$ , where  $g(x) = x - f(x)/f^{(1)}(x)$ .
- The iteration predicts  $x_{n+1}$  where the function is estimated to vanish.

## Algorithm

1. If  $f^{(1)}(x_n) < \text{threshold}$ , stop with warning.
2. If  $|f(x_n)| < \text{threshold}$ ,  $x^* = x_n$ ; else continue.
3.  $x_{n+1} = g(x_n) \Rightarrow x_{n+1} = x_n - f(x_n)/f^{(1)}(x_n)$ .

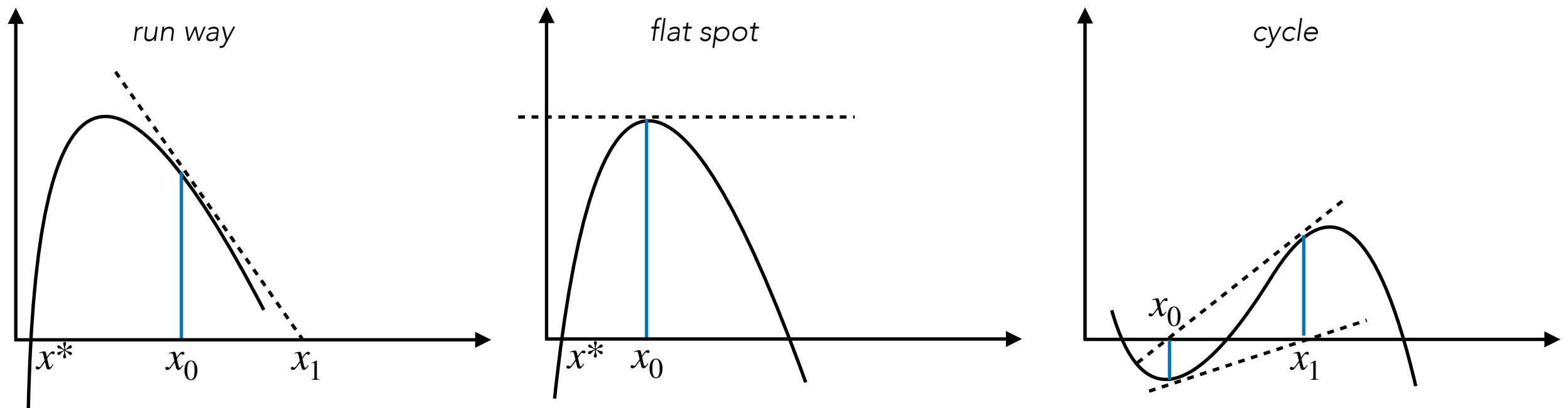


## Exercise

- In general for the fixed point iteration method, show that  $|x^* - x_{n+1}| = |g^{(1)}(t)| |x^* - x_n|$ , where  $t \in (x_n, x^*)$ . Show that for the Newton-Raphson method,  $|x^* - x_{n+1}| = (g^{(2)}(t)/2) |x^* - x_n|^2$  (i.e., the error converges quadratically).
- Write a Python program to find the square-root of 5 by solving the equation  $x^2 - 5 = 0$  using the Newton-Raphson method starting with  $x_0 = 1$ . How many iterations does it take to reach  $|x_{n+1} - x_n| < 0.001$ .
- Multiple roots: If  $f(x)$  has more than one root at  $x = x^*$  (for example,  $f(x) = (x - x^*)^M$ ), then show that Newton-Raphson iteration fails to converge. Show that in this case, the modified Newton-Raphson iteration given below works

$$x_{n+1} = x_n - \frac{f(x_n)f^{(1)}(x_n)}{[f^{(1)}(x_n)]^2 - f(x_n)f^{(2)}(x_n)}$$

## Problematic initial guesses for Newton-Raphson



## Quasi-Newton-Raphson method (Secant method)

- Nearly in all practical applications of root-finding or minimization problems (especially in higher-dimensional problems such as geometry optimization in computational chemistry research), a variant of Newton-Raphson method, called as the quasi-Newton-Raphson method is used. In one-dimension, it is called as the Secant method.
- The word 'quasi' implies that  $f^{(1)}(x)$  is estimated through finite derivatives and an explicit formula for it is not required. This makes the method very useful for problems where the analytic derivative of  $f(x)$  is not amenable.
- Along with  $x_0$ , the approach requires a small increment  $\Delta x$ , and  $x_1$  is set as  $x_0 + \Delta x$ . Then,  $f^{(1)}(x_1)$  is estimated as  $(f(x_1) - f(x_0))/\Delta x$ .
- At the  $n$ -th iteration,  $f^{(1)}(x_n)$  is estimated as  $(f(x_n) - f(x_{n-1}))/ (x_n - x_{n-1})$ .



# Python basics

## 10 things in Python to know

1. Basic datatypes in Python (int, float, complex, string)
2. String manipulation (splitting, splicing)
3. Python list (more commonly encountered datatype during string processing)
4. String concatenation and list appending (building larger strings)
5. For loop (to iterate over a list)
6. While loop (when the number of cycles is unknown apriori)
7. Break vs. continue (gracefully exiting a loop)
8. Functions (writing custom functions, doc string, help)
9. Module (importing modules, math, numpy, numpy arrays.)
10. Input/output (Read from/Write in a file, formatted printing)

## Exercise

- Go through the content in (*Python\_Basics.ipynb*) provided in the course repository.
- Go through the content in (*Python\_NumpyBasics.ipynb*) provided in the course repository.