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

tifr HYDERABAD | TATA INSTITUTE OF FUNDAMENTAL RESEARCH

Content

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 Univerity 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 ouput.
- For a given problem, there may not 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 ≥ 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 threom) that a solution (if it exists) can be found as a limit.
- *Heuristic methods* are methods developed based on experience. These methods are not guarateed 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 $g(x) = \dfrac{1}{2}\left(x + \dfrac{A}{x}\right)$, 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 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[1].

[1] *https://github.com/raghurama123/nm2024*

for loop

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

```python
# 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
```

Self-study

- 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.

```python
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 $f(x) = 0$, can be determined by rewriting the equation in the form $x = g(x)$ and beginning with an initial value x_0. Then, under certain conditions, the sequence $x_1 = g(x_0), x_2 = g(x_1), \ldots$ will converge to one of the solutions of $x = g(x)$, which we will denote as $x^* = \lim_{n \to \infty} x_n$.

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^*.

Theorem

- **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:** If the *same* $g(x)$ (satisfying all conditions stated above) further satisfies $|g^{(1)}(x)| \leq k$ for some 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 $x_1, x_2, \ldots, x^* \in [a, b]$.
  - 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 $\delta > 0$.