Basics of Applied Analysis a (応用解析学基礎ａ) Lecture 1

Norbert Pozar (npozar@se.kanazawa-u.ac.jp)

Apr 10, 2018

Today's plan

1. Lecture info

   grading, …

2. Examples of an iterative numerical methods

   Fixed point, Newton's method

Lecture info

Basic data

Name 応用解析学基礎 a,b

Basics of Applied Analysis a, b

Instructor Norbert Pozar (npozar@se.kanazawa-u.ac.jp)

office: **#228**, building 5

Office hours after lecture: **Tue 14:30–15:00**

Send me an email to schedule another time, or if more time is needed.

Self study At least three hours at home every week

Slides Posted on Acanthus portal (LMS)

Grading

Reports **100%** of the final grade

- ~ 2 reports per quarter, 100 points each, some math and programming
- reports and scores will be posted to the **Acanthus Portal** (LMS or WebClass)

Final grade based on the standard rating method:

final score	grade
$\geq 90\%$	S
$\geq 80\%$	A
$\geq 70\%$	B
$\geq 60\%$	C
$< 60\%$	fail

Academic integrity

Discussing reports with other students is OK, but:

- Submit only **your own solutions** in your own words, **do not copy** from other students, books, internet, …
- Submit only **your own code**, do not submit others' code, a code from your previous class, code from the internet, …

Main goal of the lecture

Iterative numerical methods for large linear systems

$$Ax = b$$

Given data A … large $N \times N$ matrix

b … given N-dimensional vector

Unknowns x … N-dimensional vector

Motivation

Foundation of many numerical methods used in practice for solving partial differential equations, data analysis, machine learning, AI, image processing, …

Used to solve of **nonlinear** problems.

Outline of lectures (syllabus)[1]

1. introduction, linear systems
2. review of matrix notions, Gaussian elimination, finite difference method for an elliptic differential equation
3. basic **iterative methods**: Jacobi, Gauss-Seidel, SOR
4. **convergence analysis** of iterative methods
5. **multigrid methods** 1
6. multigrid methods 2
7. **conjugate gradient method** 1
8. conjugate gradient method 2, preconditioning

References

Thomas, J.W., *Numerical Partial Differential Equations: Conservation Laws and Elliptic Equations*, 1999

- Available at Springer Link (from Kanazawa University network):

https://link.springer.com/book/10.1007%2F978-1-4612-0569-2

- We will cover parts of **Section 10**.

Ueberhuber, C.W., *Numerical Computation 1*, 1997

- Springer Link:

https://link.springer.com/book/10.1007%2F978-3-642-59118-1

Programming language for submitted reports

Any language that does the job is OK:

C/C++, Fortran, Python, Rust, …

Code shown during the class

I will mostly use **Python 3**.

- Simple and powerful language, lots of libraries for scientific computing, visualization.
- Very popular in scientific computing, data science, machine learning and AI.

[1] Schedule and content might be adjusted depending on the progress and your interest.

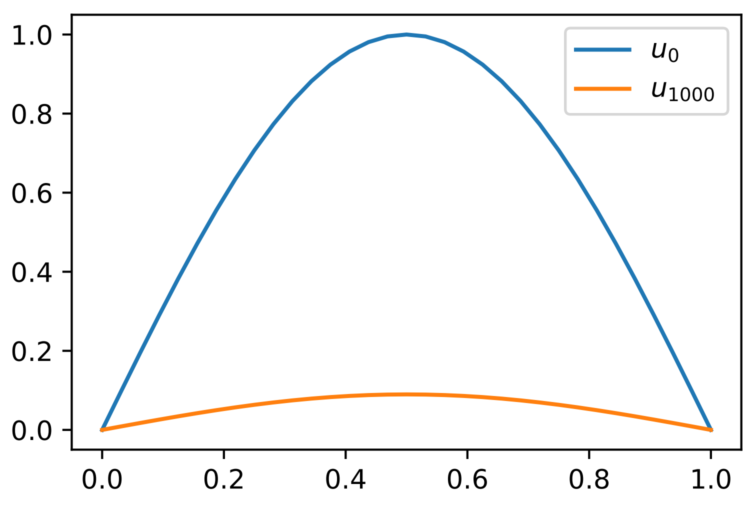
- Accessible to beginners, great for fast prototyping, no segfaults.
- Free and open source with a lot of resources online.

Python (heat equation solver in 1D)

```
import numpy as np
import matplotlib.pyplot as plt
N = 32
c = 1. / 4.   # c = Δt / Δx²
x = np.linspace(0., 1., N + 1)
u0 = np.sin(np.pi * x)
u = np.copy(u0)
for i in range(1000):
    u[1:-1] = (1. - 2. * c) * u[1:-1] \
              + c * u[:-2] + c * u[2:]

plt.plot(x, u0, label = "$u_0$")
plt.plot(x, u, label = "$u_{1000}$")
plt.legend()
plt.show()
```

Output



Jupyter Notebook

- Interactive programming with inputs and outputs in one place, similar to Mathematica or Maple.
- Great for experimenting.

Usage

Start by:

```
$ cd my_code_dir
$ jupyter notebook
```

This opens your browser.

Installing Python

Python, Jupyter and all necessary tools can be installed easily using **Anaconda**:

https://www.anaconda.com/download

Select version **Python 3.6** for your system and follow the instructions.

Learning Python

Follow a tutorial online:

- **Python Numpy Tutorial**: http://cs231n.github.io/python-numpy-tutorial/
- Jupyter Tutorial: http://cs231n.github.io/ipython-tutorial/
- Scipy Lecture Notes: http://www.scipy-lectures.org/
- Langtangen, H.P., *A Primer on Scientific Programming with Python*

Examples of iterative numerical methods

Transcendental equation

Problem

Solve the following for $x \in \mathbb{R}$:

$$\cos x = x$$

. . .

Solution

- $x = 0.7390851332151607\ldots$
- No *explicit* formula, need a numerical method.

Fixed point iteration

We want to find a **fixed point** of the map

$$x \mapsto \cos x$$

That is, a value x such that applying the function cos yields the same value.

. . .

Algorithm

1. Set $x_0 = 0$ (*initial guess*)
2. **Iterate** for $k = 0, \ldots$

$$x_{k+1} = \cos x_k$$

If the sequence x_k has a limit x_∞, it must be a solution of $\cos x = x$.

Python code

```python
from math import cos

x = 0.
for i in range(100):
    x = cos(x)

print(x)
```

Mathematical explanation

The function $f(x) = \cos x$ is a **contraction mapping** on interval $[-1, 1]$:

Definition (Contraction mapping)

There exists[2] a constant $0 \leq L < 1$ such that

$$|f(x) - f(y)| \leq L|x - y| \qquad \text{for all } x, y \in [-1, 1].$$

Theorem (Banach fixed-point theorem)

A contraction mapping $f : [-1, 1] \to [-1, 1]$ has *exactly one* fixed point x^* such that $f(x^*) = x^*$.

Error estimate

$$|x_k - x^*| \leq L^k |x_0 - x^*|$$

. . .

Proof

Recall

$$x_k = f(x_{k-1}) \qquad \text{and} \qquad x^* = f(x^*).$$

Then iteratively

$$|x_k - x^*| = |f(x_{k-1}) - f(x^*)| \leq L|x_{k-1} - x^*| \leq \cdots \leq L^k |x_0 - x^*|.$$

Stopping condition

A similar calculation yields

$$x^{k+1} - x^* \sim \frac{f'(x^*)}{f'(x^*) - 1}(x^{k+1} - x^k)$$

. . .

[2]For $f(x) = \cos x$ we can take $L = \sin 1 = 0.8415\ldots$ by the *mean value theorem*.

```python
from math import cos

x = 0.
for i in range(100):
    x_old = x
    x = cos(x)
    if abs(x - x_old) < 1e-6:
        break

print(x)
```

Newton's (Newton-Raphson) method

Problem

Find the solution $x \in \mathbb{R}$ of

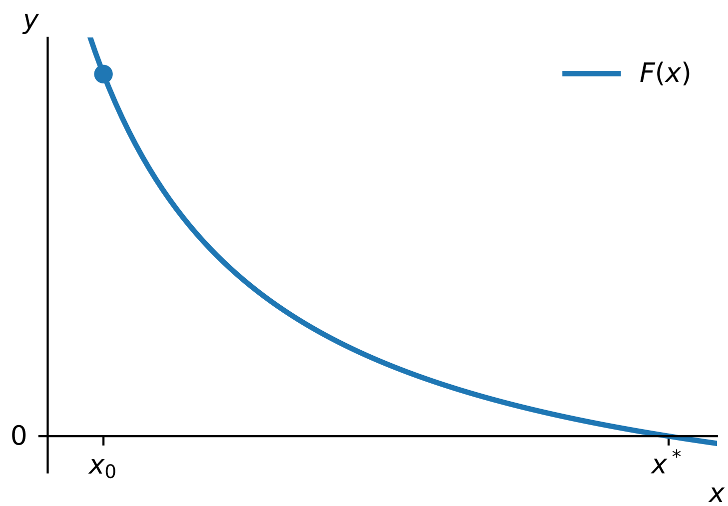
$$F(x) = 0$$

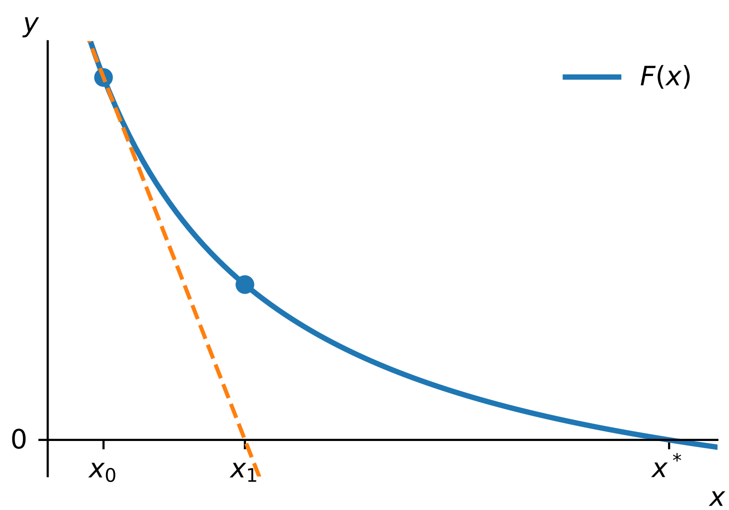
where $F : \mathbb{R} \to \mathbb{R}$ is a continuously differentiable function (C^1).

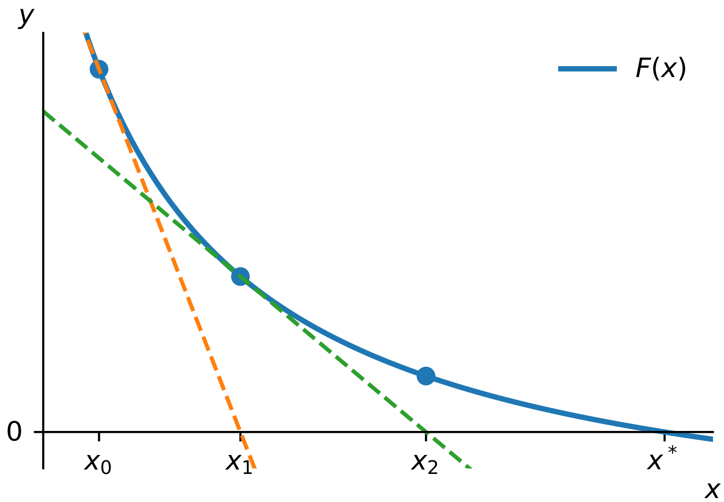
. . .

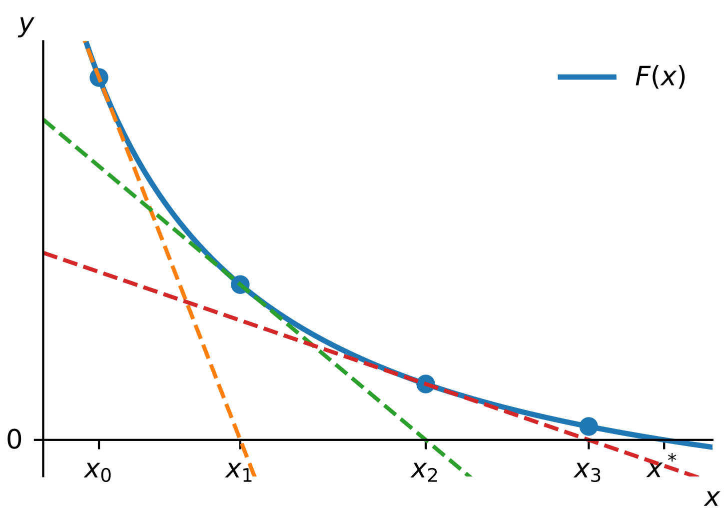
Idea

Approximate the nonlinear equation $F(x) = 0$ by a **linear** equation.


y
F(x)
0
x0
x*
x


y
F(x)
0
x0
x1
x*
x





Newton's iterative method

1. Fix x_0 an initial guess.

2. Linear approximation of F at x_0 (Taylor's theorem):

$$L_{x_0}(x) = F(x_0) + F'(x_0)(x - x_0).$$

3. Find x_1 as the solution of

$$L_{x_0}(x_1) = 0 \qquad \Leftrightarrow \qquad x_1 = x_0 - \frac{F(x_0)}{F'(x_0)}$$

Repeat to find x_2, x_3, …

Exercise: $\cos x = x$

Solve

$$\cos x = x$$

using Newton's method.

Python code

```
from math import cos, sin

x = 0.

for i in range(20):
    x = x - (cos(x) - x)/(-sin(x) - 1)

print(x)
```

Choice of x_0

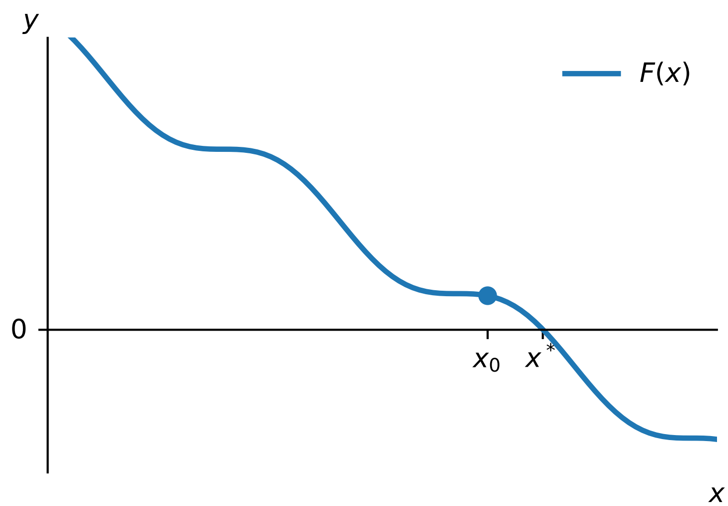
Newton's method can be **very sensitive** to the choice of the initial value x_0

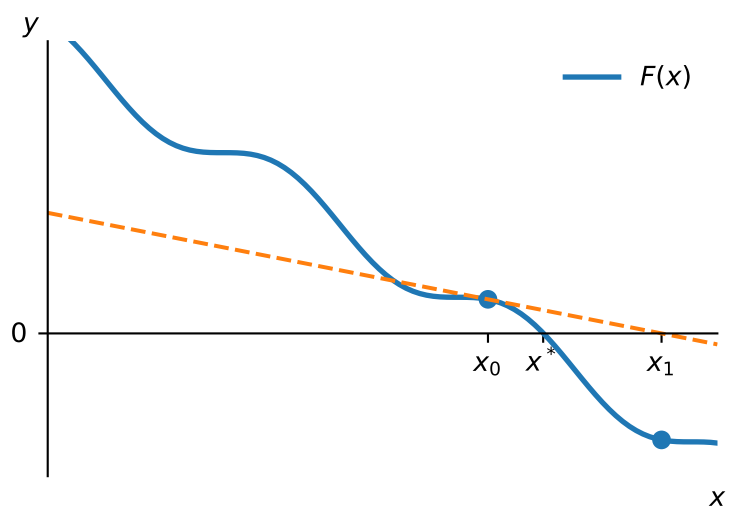
For

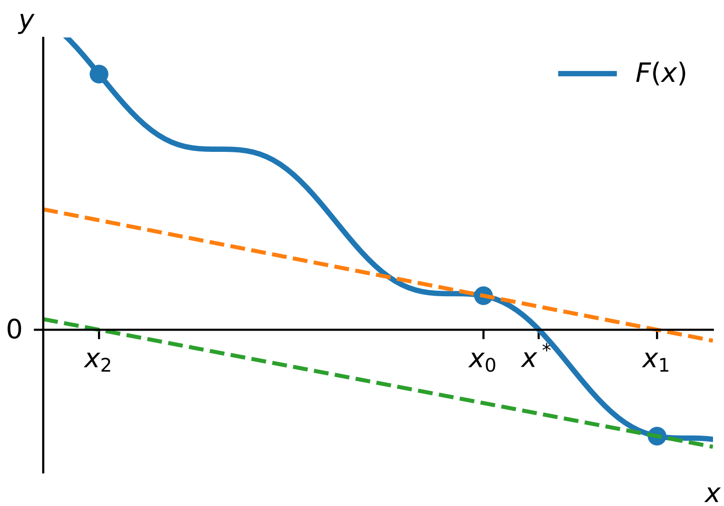
$$\cos x - x = 0$$

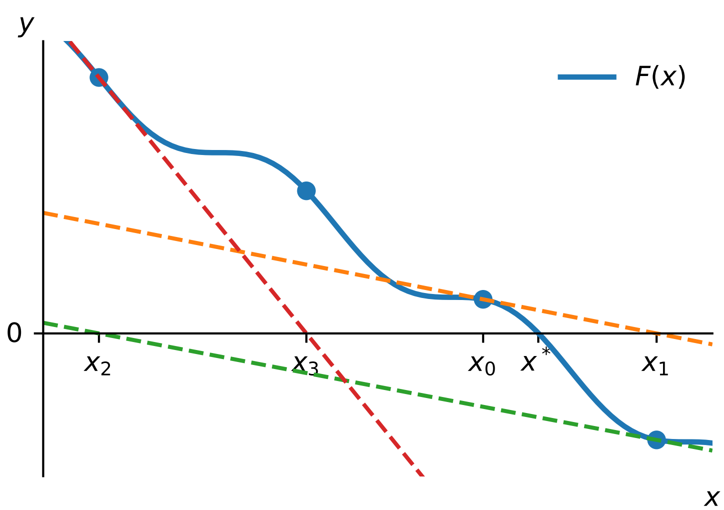
try

$$x_0 = -0.75$$


y
F(x)
0
x0
x*
x


y
F(x)
0
x0
x*
x1
x


y
F(x)
0
x2
x0
x*
x1
x


y
F(x)
0
x2
x3
x0
x*
x1
x

Comparison of the methods

Newton's method

- Very fast convergence
- Requires a **good** initial guess
- Requires the derivative

Fixed point iteration

- Works for **any** initial guess
- Slower convergence

. . .

We can **combine** both methods: A few iterations of the fixed point method to find a good initial guess for Newton's method.

Example: linear equation

Let $a \neq 0$. Solve $ax = 1$ for x numerically.

. . .

Solution

Set

$$F(x) := ax - 1 \qquad F'(x) = a$$

Newton's method

$$x_{k+1} = x_k - \frac{ax_k - 1}{a} = \frac{1}{a}, \qquad k = 0, 1, ...$$

Another way

$$ax = 1 \quad \Leftrightarrow \quad \frac{1}{x} - a = 0$$

Set

$$F(x) := \frac{1}{x} - a \qquad F'(x) = -\frac{1}{x^2}$$

Newton's method

$$x_{k+1} = x_k(2 - ax_k), \qquad k = 0, 1, ...$$

Does not need a division!

Python code

```
a = 0.5
x = 2 - a      # initial guess for 0 <= a <= 1

for i in range(6):
    x = x * (2 - a * x)

print(x)
```

Error estimate for Newton's method

Set $x^* = \frac{1}{a}$.

We have

$$|x_{k+1} - x^*| = |a||x_k - x^*|^2$$

The rate of convergence is **quadratic**: Number of correct digits **doubles** each iteration!

Proof

Exercise

Linear equation

Problem

Solve for $x \in \mathbb{R}$ in

$$ax = b,$$

where $a, b \in \mathbb{R}$ are given numbers, $a \neq 0$.

. . .

Solution

- Clearly the answer is $x = \frac{b}{a}$.
- But is this is the most **efficient** way to get the numerical value of the solution?

Computational cost of mathematical operations

Speed of basic mathematical operations on a **modern CPU**[3]:

Instruction	Cycles
`ADD`, `SUB`, `MUL`	0.5–1
`DIV`, `SQRT`	7–14

Division is about 20× slower than addition, subtraction and multiplication!

But Newton's method

We can compute $\frac{b}{a}$ without division!

Summary: solving $ax = b$

Computational complexity

Symbolic $\frac{b}{a}$: 1 `DIV` instruction.

Iterative method: Initial guess + (2 `MUL` and 1 `SUB`) × number of iterations + 1 `MUL`

Recall: `DIV` about 20× slower than `MUL` or `SUB`

[3] Agner Fog, Instruction tables, http://www.agner.org/optimize/

Conclusion

Iterative method[4] is likely faster (depending on how much precision you need).

Exercise: a quadratic equation

With $a \geq 0$, solve
$$x^2 = a$$
for x without using a square root or a division[5].

Measuring "error"

Absolute error

$$e_{\text{abs}} := x_{\text{approx}} - x_{\text{exact}}$$

> How big is too big?

. . .

Relative error

$$e_{\text{rel}} := \frac{e_{\text{abs}}}{x_{\text{ref}}}$$

Read more about errors in [Ueberhuber, Sec. 2.2, 2.3].

Next time

- Review of linear algebra: vectors, matrices.
- Guassian elimination.
- Finite difference method for an elliptic differential equations

Self study

- Read sections 2.1–2.3 in [Ueberhuber, *Numerical Computation 1*].
- Go through the Python Numpy Tutorial at http://cs231n.github.io/python-numpy-tutorial/

[4] Modern CPUs actually internally use a similar iterative method to implement `DIV`.

[5] *Hint.* Assume that $a > 0$ and note that $x = \frac{a}{x}$. Use Newton's method to find $y := \frac{1}{x}$ instead.