Оптимизация и численные методы

Болонкин Максим Викторович

Филиал МГУ им. М.В. Ломоносова в г. Ташкенте

Generally

- This is an overview course, there is no single story line
- The course objective: to introduce you to some of the most relevant computational and numerical tools commonly used in research and engineering
- The course materials are borrowed from various sources, I put attribution wherever possible
- Slides and course materials are predominantly in English
- This course has been offered since 2018, I used to change the actual topic several times
- This time the course will return to the original content but with certain updates and improvements

Course outline

"Computational and numerical tools" (known as *"Optimization and numerical methods"* in your curriculum)

What this course is about:

- Some of the most relevant computational and numerical tools commonly used in research and engineering
- Tools = algorithms and implementations

What this course is **not** about:

- Minimal math component: some math for understanding, rare derivations, almost no analysis/proofs
- Not a programming course: you are expected to know basic programming
- Breadth, not depth

Logistics

This course consists of **lectures and programming seminars**.

Lecture notes and video links will be posted online:
`https://www.mbolonkin.info/teaching/cs170-2026`

Seminars: you (students) will be teaching in the practical sessions
Final Project: details will be announced later

Final Grade: will be based on your performance in the seminars and the final project plus optional final exam if you want to improve your grade.

Final Project

The main source of your grade in this course will be your final project.

- One project per person (exceptions for approved collaborations)
- Project phases:
  - Proposal and approval
  - Midterm milestone
  - Final report and presentation
- The project will involve solving a practical problem using computational and numerical tools covered in this course.
- You will present your proposal and results in a final presentation.
- You are allowed to work on your thesis project as your final project for this course, but you need to get approval from me first.
- Final report must be submitted for you to receive your final grade.
- You are welcome to use AI tools as long as you understand the code and can explain it in your final presentation.

Programming language

Expected programming tools for this course:

- Matlab/Octave
- Python + Jupyter Notebooks (Recommended)

Why Python?

Python has well supported scientific libraries

- NumPy - supports wide variety of matrix operations
- SciPy - a lot of computational methods implemented
- MatPlotLib - library for plotting and visualizing
- Scikit-Image, Scikit-Learn, etc.
- Very good documentation

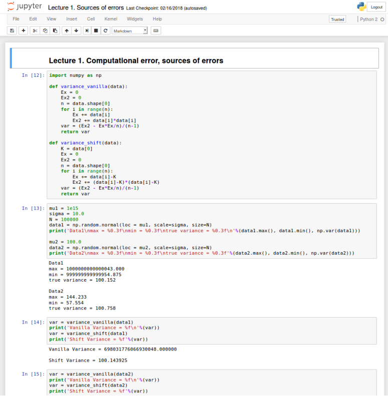
Why Python?

Python has well supported scientific libraries

- NumPy - supports wide variety of matrix operations
- SciPy - a lot of computational methods implemented
- MatPlotLib - library for plotting and visualizing
- Scikit-Image, Scikit-Learn, etc.
- Very good documentation

Jupyter Notebooks:

- Convenient tool for coding and debugging
- Good for reproducible research
- Getting more and more popular



Any questions?

Why bother?

- Computation allows exploring models without analytical solution
- This is usually the case for real-world problems
- Advances in hardware and software make it easier to use computational models

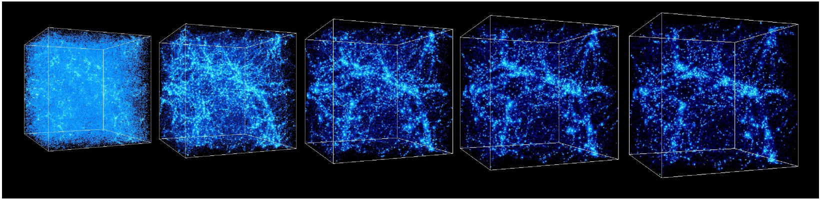


Рис.: Galaxy formation (cosmicweb.uchicago.edu)

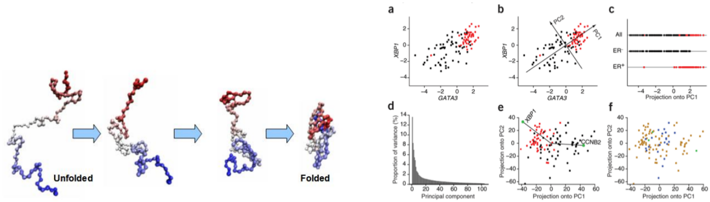


Рис.: Protein unfolding simulation and analysis of gene expression

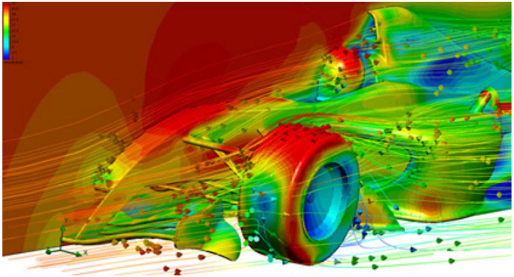


Рис.: Fluid dynamics simulation

Top 10 Algorithms of 20th Century

In 2000 January-February issue of *Computing in Science and Engineering* editors selected 10 algorithms that influenced science and engineering in 20th century.

1. **Metropolis algorithm for Monte Carlo**

   1946: John von Neumann, Stan Ulam, and Nick Metropolis

   Through the use of random processes, this algorithm offers an efficient way to stumble toward answers to problems that are too complicated to solve exactly.

2. **Simplex method for linear programming**

   1947: George Dantzig, at the RAND Corporation

   An elegant solution to a common problem in planning and decision-making.

3. **Krylov subspace iteration methods**

   1950: Magnus Hestenes, Eduard Stiefel, and Cornelius Lanczos

   A technique for rapidly solving the linear equations that abound in scientific computation.

4. **The decompositional approach to matrix computations**

   1951: Alston Householder of Oak Ridge National Laboratory

   A suite of techniques for numerical linear algebra.

5 **The Fortran optimizing compiler**

1957: John Backus, IBM

Turns high-level code into efficient computer-readable code.

6 **QR algorithm for computing eigenvalues**

1959–61: J.G.F. Francis

Another crucial matrix operation made swift and practical.

7 **Quicksort algorithm for sorting**

1962: Tony Hoare

For the efficient handling of large databases.

8 **Fast Fourier transform**

1965: James Cooley of the IBM T.J. Watson Research Center and John Tukey of Princeton University and AT&T Bell Laboratories

Perhaps the most ubiquitous algorithm in use today, it breaks down waveforms (like sound) into periodic components.

9 **Integer relation detection**

1977: Helaman Ferguson and Rodney Forcade of Brigham Young University

A fast method for spotting simple equations satisfied by collections of seemingly unrelated numbers.

10 **Fast multipole method**

1987: Leslie Greengard and Vladimir Rokhlin of Yale University

A breakthrough in dealing with the complexity of n-body calculations, applied in problems ranging from celestial mechanics to protein folding.

In 2016 editors of the CiSE updated the list of top 10 algorithms, adding the following:

1. **Newton and quasi-Newton methods**
2. **Jpeg**
3. **PageRank**
4. **Kalman filter**

What we are going to cover in this course:

- **Data fitting**
  Some topics: linear least squares, non-linear least squares, maximum likelihood, maximum a posteriori, expectation maximization
- **Optimization**
  Some topics: gradient-free optimization, gradient descent, non-linear optimization, heuristics
- **Dimensionaity reduction and component analysis**
  Some topics: PCA, ICA, t-SNE, LDA
- **Signal Processing**
  Some topics: signals, filters, DFT
- **Randomized methods**
  Some topics: random number sampling, Monte-Carlo methods, RANSAC
- **Differential equations**
  Some topics: numerical solutions for Ordinary Differential Equations (ODE), Partial Differential Equations (PDE)

General strategy for Numerical Solution

- Replacing difficult problem by simplified problem with the same or close enough solution

  Infinite → Finite
  Differential → Algebraic
  Continuous → Discrete
  Non-linear → Linear
  ...

- Solution obtained only approximates the exact solution of the original problem
  - Approximation before computation (empirical measurements, previous computation)
  - Approximation during computation (covered in next part)
  - Final result accuracy reflects all those errors, possibly amplified by an agorithm/problem

Lecture 1. Numerical stability, sources of errors

Maksim Bolonkin

Moscow State University

Types of errors

If we approximate x with $\tilde{x}$ there are two types of errors:

- Absolute error

$$e = x - \tilde{x}, \quad \tilde{x} = x + e$$

- Relative error

$$\epsilon = \frac{x - \tilde{x}}{x}, \quad \tilde{x} = x \cdot (1 + \epsilon)$$

Most of the times only absolute value of the error is relevant.

The same classification applies to performance metrics comparison:

	224×224		320×320 / 299×299	
	top-1 err	top-5 err	top-1 err	top-5 err
ResNet-101 [14]	22.0	6.0	-	-
ResNet-200 [15]	21.7	5.8	20.1	4.8
Inception-v3 [39]	-	-	21.2	5.6
Inception-v4 [37]	-	-	20.0	5.0
Inception-ResNet-v2 [37]	-	-	19.9	4.9
ResNeXt-101 (**64 × 4d**)	20.4	5.3	**19.1**	**4.4**

Table 5. State-of-the-art models on the ImageNet-1K validation set (single-crop testing). The test size of ResNet/ResNeXt is 224×224 and 320×320 as in [15] and of the Inception models is 299×299.

Model		
Cimpoi '15 [4]		66.7
Zhang '14 [30]		74.9
Branson '14 [2]		75.7
Lin '15 [20]		80.9
Simon '15 [24]		81.0
CNN (ours)	224px	82.3
2×ST-CNN	224px	83.1
2×ST-CNN	448px	83.9
4×ST-CNN	448px	**84.1**

Sources of errors

Some common sources of errors in computational processes:

1. *Rounding errors* from inexact computer arithmetics.
2. *Truncation/discretization error* from approximate formulas.
3. *Termination of iterations*
4. *Randomness induced errors* (e.g. in Monte-Carlo methods)

Rounding error vs. truncation error

Consider approximating function's derivative (finite difference method).

$$f_{diff}(x; h) = \frac{f(x+h) - f(x)}{h}$$

From Taylor expansion

$$f(x+h) = f(x) + hf'(x) + f''(\theta)\frac{h^2}{2}, \quad \text{where } \theta \in [x, x+h]$$

we can see that

$$f_{diff}(x; h) = \frac{f(x+h) - f(x)}{h} = f'(x) + f''(\theta)\frac{h}{2}$$

If second derivative is bounded on $[x, x+h]$ then

$$|f'(x) - f_{diff}(x; h)| \leq Mh \quad \text{for some } M$$

Rounding error vs. truncation error

On the other hand there is a computer arithmetics we are dealing with. Let $\tilde{f}_{diff}$ denote an approximation of f_{diff} due to a finite precision.
Numerator of $\tilde{f}_{diff}$ introduces rounding error $\leq \epsilon|f(x)|$

$$|f_{diff}(x;h) - \tilde{f}_{diff}(x;h)| \leq \left|\frac{f(x+h)-f(x)}{h} - \frac{f(x+h)-f(x)+\epsilon f(x)}{h}\right|$$

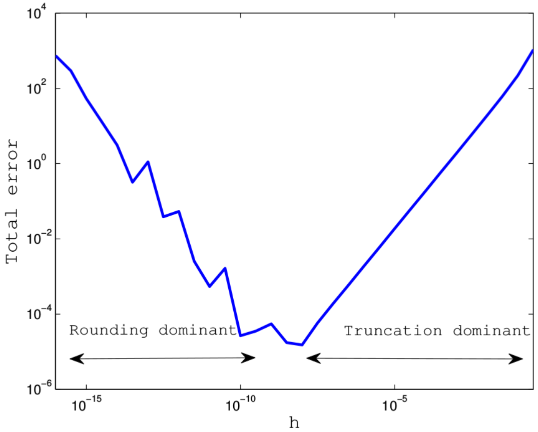
$$\leq \epsilon\frac{|f(x)|}{h}$$

Therefore total error is

$$|f'(x) - \tilde{f}_{diff}(x;h)| \leq Mh + \frac{\epsilon|f(x)|}{h}$$

Rounding error vs. truncation error (example)

Let's assume $f(x) = e^{5x}$, and we want to estimate derivative at $x = 1$.



Machine precision

$$\underbrace{\pm}_{\text{1 sign bit}} \quad \underbrace{d_1, d_2, \ldots, d_p}_{\text{p mantissa bits}} \quad \underbrace{E}_{\text{exponent bits}}$$

Exponent resides in the interval $L \leq E \leq U$.

	total bits	p	L	U
IEEE single precision	32	23	-126	127
IEEE double precision	64	52	-1022	1023

Rounding error ϵ (or ϵ_{mach}) is introduced when mantissa requires more than p bits.
In IEEE double precision, $\epsilon = 2^{-52} \approx 2.22 \times 10^{-16}$

Well-posed problem (according to Hadamard definition):

1. a solution exists
2. the solution is unique
3. solution's behavior changes continuously with respect to changes in initial conditions

Otherwise a problem is **ill-posed**. It's solution might be sensitive to noise in input data.

Well-posed problems	Ill-posed problems
Multiplication by a small number $$y = ax, \quad a \ll 1$$	Division by a small number $$y = x/a, a \ll 1$$
Multiplication by a matrix $$y = Ax$$	Inversion of the matrix when matrix is nearly singular $$x = A^{-1}y$$
Integration $y(t) = y(0) + \int_0^t x(t)dt$	Differentiation $x(t) = y'(t)$

Condition number

Many numerical operations can be represented as

$$y = f(x)$$

Small change in x leads to some change in y:

$$y + \Delta y = f(x + \Delta x)$$

Condition number quantifies the relative change

$$k = \frac{|\Delta y / y|}{|\Delta x / x|}$$

Condition number: function evaluation

Suppose $f : \mathbb{R} \to \mathbb{R}$ is a differentiable function.

$$\frac{\Delta y}{y} = \frac{f(x+\Delta x) - f(x)}{f(x)} = \frac{f(x+\Delta x) - f(x)}{\Delta x} \frac{\Delta x}{f(x)}$$

When Δx is small, we have

$$\frac{\Delta y}{y} \approx \frac{f'(x)\Delta x}{f(x)}$$

Therefore,

$$k \approx \left| \frac{x f'(x)}{f(x)} \right|$$

Condition number: system of equations

Let's consider system of equations

$$Ax = b$$

From linearity of matrix-vector multiplication we have

$$A(x + \Delta x) = b + \Delta b \quad \Longrightarrow \quad A\Delta x = \Delta b$$

Recalling the matrix norm definition, we have

$$k = \frac{||\Delta b||/||b||}{||\Delta x||/||x||} = \frac{||A\Delta x||}{||\Delta x||} \frac{||x||}{||Ax||} \leq ||A|| \cdot ||A^{-1}||$$

Algorithm example: sample variance

Problem: Given an array X of numbers find variance s^2
Solution: Relies on formula

$$s^2 = \frac{\sum_{i=0}^{N} x_i^2 - \frac{1}{N}(\sum_{i=1}^{N} x_i)^2}{N-1}$$

$n \leftarrow \text{size(X)}$
$Sum \leftarrow 0$, $SumSq \leftarrow 0$
for all $x \in X$ **do**
  $Sum \leftarrow Sum + x$
  $SumSq \leftarrow SumSq + x^2$
end for
$Var \leftarrow (SumSq - Sum \times Sum/n)/(n-1)$

What is the problem with this approach?

Algorithm example: sample variance

In [1]:
```python
import numpy as np

def variance_vanilla(data):
    Ex = 0
    Ex2 = 0
    n = data.shape[0]
    for i in range(n):
        Ex += data[i]
        Ex2 += data[i]*data[i]
    var = (Ex2 - Ex*Ex/n)/(n-1)
    return var
```

In [2]:
```python
sigma = 10.0
N = 100000
```

In [4]:
```python
mu1 = 1e15
data1 = np.random.normal(loc = mu1, scale=sigma, size=N)
print('Data1\nmax = %0.3f\nmin = %0.3f\ntrue variance = %0.3f'%(data1.max(), data1.min(), np.var(data1)))
var = variance_vanilla(data1)
print('Vanilla Variance = %f\n'%(var))

mu2 = 100.0
data2 = np.random.normal(loc = mu2, scale=sigma, size=N)
print('Data2\nmax = %0.3f\nmin = %0.3f\ntrue variance = %0.3f'%(data2.max(), data2.min(), np.var(data2)))
var = variance_vanilla(data2)
print('Vanilla Variance = %f\n'%(var))
```

```
Data1
max = 1000000000000041.500
min = 999999999999953.750
true variance = 99.610
Vanilla Variance = 698031776066930048.000000

Data2
max = 143.500
min = 56.895
true variance = 100.596
Vanilla Variance = 100.596730
```

Problem with that solution: catastrophic cancellation on large numbers with small variance
Solution: Variance is invariant to shift:

$$Var(X - K) = Var(X)$$

$n \leftarrow \text{size(X)}$
$K \leftarrow X[0]$
$Sum \leftarrow 0,\ SumSq \leftarrow 0$
for all $x \in X$ **do**
  $Sum \leftarrow Sum + (x - K)$
  $SumSq \leftarrow SumSq + (x - K)^2$
end for
$Var \leftarrow (SumSq - Sum \times Sum/n)/(n - 1)$

Does that solve all the problems?

Algorithm example: sample variance

In [5]:
```
def variance_shift(data):
    K = data[0]
    Ex = 0
    Ex2 = 0
    n = data.shape[0]
    for i in range(n):
        Ex += data[i]-K
        Ex2 += (data[i]-K)*(data[i]-K)
    var = (Ex2 - Ex*Ex/n)/(n-1)
    return var
```

In [6]:
```
sigma = 10.0
N = 100000
```

In [7]:
```
mu1 = 1e15
data1 = np.random.normal(loc = mu1, scale=sigma, size=N)
print('Data1\nmax = %0.3f\nmin = %0.3f\ntrue variance = %0.3f'%(data1.max(), data1.min(), np.var(data1)))
var = variance_shift(data1)
print('Vanilla Variance = %f\n'%(var))

mu2 = 100.0
data2 = np.random.normal(loc = mu2, scale=sigma, size=N)
print('Data2\nmax = %0.3f\nmin = %0.3f\ntrue variance = %0.3f'%(data2.max(), data2.min(), np.var(data2)))
var = variance_shift(data2)
print('Vanilla Variance = %f\n'%(var))
```

```
Data1
max = 1000000000000043.250
min = 999999999999956.375
true variance = 100.852
Vanilla Variance = 100.852711

Data2
max = 146.156
min = 59.814
true variance = 99.726
Vanilla Variance = 99.726858
```

The Patriot Missile Failure

February 25, 1991: an American Patriot Missile battery in Dharan, Saudi Arabia, failed to track and intercept an incoming Iraqi Scud missile.

- Time was stored in tenths of seconds since boot time
- Multiplied by 0.1 to get time in seconds
- 0.1 was stored in 24-bit representation leading to 0.95×10^{-7} abs. error
- In 100 hours total error was 0.34 seconds
- Incoming missile's speed was $1,676$ mps
- 28 dead soldiers, about 100 injured

Explosion of the Ariane 5

June 4, 1996: Ariane 5 rocket launched by the European Space Agency exploded just forty seconds after lift-off

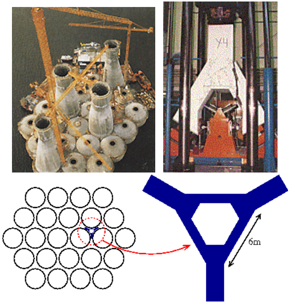
- 64 bit floating point number (vertical speed) was converted to a 16 bit signed integer
- Floating point number was greater than $32,768$
- The destroyed rocket and its cargo were valued at $500 million

Real world examples

The sinking of the Sleipner A offshore platform

August 23, 1991: concrete base structure for Sleipner A sprang a leak and sank under a controlled ballasting operation during preparation for deck mating in Gandsfjorden outside Stavanger, Norway

- Error to inaccurate finite element approximation of the linear elastic model of the tricell
- The shear stresses were underestimated by 47
- Total economic loss of about $700 million



The Vancouver Stock Exchange, 1982

- New index introduced at initial value of 1000.00
- Index was updated after each transaction
- Because of truncation index had fallen to 520
- If rounded instead, it would be 1098.892

German parliament election, 1992

- The Green party was calculated to have 5% of votes (min necessary)
- Means no seats for people from major party (Social Democrats) list
- Actual result was 4.97% of votes, rounded up due to one digit after decimal point