Basics of Applied Analysis a (応用解析学基礎ａ)
Lecture 2

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

Apr 17, 2018

Announcements

- Slides are on Acanthus portal in LMS (Learning Management System).
- If you find any typos in the slides, let me know!

Last week

- Iterative numerical methods
  - Fixed point iteration
  - Newton's method

Today's plan

1. Review of linear algebra: vectors, matrices
2. Guassian elimination

Review of linear algebra: vectors, matrices

Euclidean vector space

$\mathbb{R} = (-\infty, \infty)$ … set of real numbers

$n \in \mathbb{N}$ natural number … **dimension**

$$\mathbb{R}^n := \underbrace{\mathbb{R} \times \cdots \times \mathbb{R}}_{n \text{ times}} = \{(x_1, x_2, \dots, x_n) \mid x_1 \in \mathbb{R},\ x_2 \in \mathbb{R}, \dots,\ x_n \in \mathbb{R}\}$$

Elements in $\mathbb{R}^n$ are called **n-dimensional (Euclidean) vectors**[1].

Operations

- addition

$$x + y := (x_1 + y_1, \dots, x_n + y_n), \qquad x, y \in \mathbb{R}^n$$

- multiplication by a scalar

$$\alpha x := (\alpha x_1, \dots, \alpha x_n) \qquad x \in \mathbb{R}^n, \alpha \in \mathbb{R}$$

Representation on a computer

Stored in memory as a 1-dimensional **array**.

Language	Representation
Python	`np.array`
C	`double[]`, `double*`
C++	`vector<double>`

Python `np.array`

`np.array` implements addition and multiplication by a scalar.

```
import numpy as np

x = np.array([1., 2.])
y = np.array([3., 4.])

print(x + y)
print(3. * x)
```

[1]Fun visual refresher on linear algebra: Essence of linear algebra by 3Blue1Brown on YouTube

Inner (dot, scalar) product

$$
\begin{aligned}
x \cdot y \equiv \langle x, y \rangle &:= x_1 y_1 + \cdots x_n y_n, \quad x, y \in \mathbb{R}^n \\
&:= \sum_{i=1}^{n} x_i y_i \\
&:= \sum_{i} x_i y_i
\end{aligned}
$$

Takes two vectors and produces a number (scalar) $\in \mathbb{R}$.

Python: `dot`

```
import numpy as np

x = np.array([1., 2.])
y = np.array([3., 4.])

print(x.dot(y))
print(np.dot(x, y))
```

Norm: length of a vector

$x \in \mathbb{R}^n$

Euclidean (ℓ^2) norm

$$||x|| \equiv ||x||_2 \equiv ||x||_{\ell^2} \equiv |x| := (x \cdot x)^{\frac{1}{2}} = (\sum_i |x_i|^2)^{\frac{1}{2}}$$

Maximum norm

$$||x||_\infty := \max(|x_1|, \dots, |x_n|) = \max_{1 \le i \le n} |x_i|$$

Note

$$||x||_\infty \le ||x||_2 \le \sqrt{n} ||x||_\infty \qquad x \in \mathbb{R}^n$$

```
import numpy as np
from math import sqrt

x = np.array([1., 2.])

def euclidean_norm(x):
    return sqrt(x.dot(x))

def max_norm(x):
    return np.max(np.abs(x))

print(euclidean_norm(x))
print(max_norm(x))

# built in
print(np.linalg.norm(x))
print(np.linalg.norm(x, ord = np.Inf))
```

Cauchy-Schwarz inequality

Theorem

$$|x \cdot y| \leq ||x||_2||y||_2 \qquad x, y \in \mathbb{R}^n$$

Definition

$\theta \in [0, \pi]$ is the angle between vectors $x, y \in \mathbb{R}^n$ if

$$\cos\theta = \frac{x \cdot y}{||x||_2||y||_2}$$

If $\theta = \frac{\pi}{2}$, that is, if $x \cdot y = 0$, the vectors are **orthogonal**.

Matrix

A **(real)** $m \times n$ **matrix** is a table of mn numbers with m rows and n columns:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & & \vdots \\ \vdots & & \ddots & \\ a_{m1} & \cdots & & a_{mn} \end{bmatrix}$$

a_{ij} are components of A. We write also $A = (a_{ij})$.

Diagonal matrix

A $n \times n$ matrix with nonzero components only on the diagonal is called a **diagonal matrix**:

$$\begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & a_{nn} \end{bmatrix}$$

Identity matrix

A $n \times n$ diagonal matrix with ones on the diagonal is called the **identity matrix**:

$$I = I_n := \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & 1 \end{bmatrix}$$

```
# 2 x 3 matrix
A = np.array([[1., 2., 3.], [4., 5., 6.]])

# 3 x 3 identity matrix
I = np.eye(3)

# 3 x 3 diagonal matrix
D = np.diag([1., 2., 3.])
```

Matrix multiplication

Definition

For $m \times n$ matrix A and $n \times p$ matrix B, we define their product

$$AB$$

as the $m \times p$ matrix with components

$$(AB)_{ik} = \sum_{j=1}^{n} a_{ij} b_{jk}$$

Note

$$(AB)C = A(BC)$$

but in general

$$AB \neq BA$$

Matrix-vector multiplication

A vector $x \in \mathbb{R}^n$ can be viewed as a $n \times 1$ (column) matrix

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

We define the product of a $m \times n$ matrix A and $x \in \mathbb{R}^n$ to be the vector $\in \mathbb{R}^m$ that corresponds to the $m \times 1$ matrix

$$Ax$$

Python: `dot` function

```
A = np.array([[1., 2., 3.], [4., 5., 6.]])
B = np.ones((3, 2))
x = np.array([1., 2., 3.])

print(A.dot(B))
print(A.dot(x))
```

Matrix as a linear map

Since

$$A(\alpha x + \beta y) = \alpha Ax + \beta Ay$$

an $m \times n$ matrix A defines a **linear map** from $\mathbb{R}^n$ to $\mathbb{R}^m$ as

$$\mathbb{R}^n \ni x \mapsto Ax \in \mathbb{R}^m$$

Transposed matrix

If A is an $m \times n$ matrix with components a_{ij}, we define its **transpose** A^T as the $n \times m$ matrix with components

$$a_{ij}^T = a_{ji}$$

Note

$$(AB)^T = B^T A^T$$

and

$$\langle Ax, y \rangle = \langle x, A^T y \rangle \qquad x \in \mathbb{R}^n, y \in \mathbb{R}^m$$

Python

```python
# 2 x 3 matrix
A = np.array([[1., 2., 3.], [4., 5., 6.]])
print(A.T)

x = np.array([3., 1., 2.])
y = np.array([3., 7.])

# <Ax, y> = <x, A^T y>
print((A.dot(x)).dot(y) == x.dot(A.T.dot(y)))
```

Symmetric matrix

An $n \times n$ matrix is **symmetric** if

$$A = A^T$$

Example

- I_n and any diagonal matrix are symmetric.
- A is a $n \times n$ matrix:

$$\frac{A + A^T}{2}$$

- A is an $m \times n$ matrix:

$$A^T A \qquad \text{and} \qquad A A^T$$

Inverse

An $n \times n$ matrix A is said to be **invertible** if there exists an $n \times n$ matrix A^{-1} such that

$$AA^{-1} = A^{-1}A = I_n$$

We call A^{-1} the **inverse** of A.

Eigenvalues, eigenvectors

Definition

Let A be a $n \times n$ matrix. We say that $\lambda \in \mathbb{R}$ is an **eigenvalue** of A if there exists $x \in \mathbb{R}^n$, $x \neq 0$, such that

$$Ax = \lambda x$$

Such x is called an **eigenvector** of A.

Recall: λ is a root of the **characteristic polynomial**

$$P(\lambda) := \det(A - \lambda I_n)$$

Exercise

Find the eigenvalues of

$$A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

```python
import numpy as np

A = np.array([[2, 1], [1, 2]])

lam, v = np.linalg.eig(A)

print(lam)
print(v)
```

Orthogonal basis of eigenvectors

Spectral theorem

If A is a *symmetric* $n \times n$ matrix then there exist eigenvectors $x^{(1)}, \dots, x^{(n)} \in \mathbb{R}^n$ of A that are pairwise orthogonal and form a basis of $\mathbb{R}^n$.

Corollary

Any symmetric matrix is **similar** to a diagonal matrix: there exist an invertible $n \times n$ matrix P and an $n \times n$ diagonal matrix D such that

$$A = PDP^{-1}$$

The diagonal components of D are the eigenvalues of A.

Positive definite matrix

A symmetric $n \times n$ matrix A is said to be **positive definite** if

$$\langle Ax, x \rangle > 0 \qquad \text{for all } x \in \mathbb{R}^n, x \neq 0$$

If $>$ is replaced by $\geq$, it is a **nonnegative definite** matrix.

It is equivalent to all eigenvalues being positive resp. nonnegative.

Example. Symmetric matrices $A^T A$ and AA^T are nonnegative definite.

Size of a matrix

Let A be an $m \times n$ matrix.

Operator norm of A is defined as

$$||A|| := \sup_{\substack{x \in \mathbb{R}^n \\ x \neq 0}} \frac{||Ax||}{||x||}$$

Measures the maximal possible **"stretch"** of a vector by applying matrix A.

If A is *symmetric* then

$$||A|| = \max\{|\lambda| \mid \lambda \text{ is an eigenvalue of } A\}$$

Disadvantage: Difficult to compute…

Norm inequalities

$$||Ax|| \leq ||A|| ||x||$$

for any $m \times n$ matrix A and $x \in \mathbb{R}^n$

$$||AB|| \leq ||A|| ||B||$$

Gaussian elimination

Algorithm for solving linear systems

A invertible $n \times n$ matrix, $b \in \mathbb{R}^n$

$$Ax = b$$

$x \in \mathbb{R}^n$ can be found by the **Gaussian elimination**:

1. Perform row operations on the $n \times (n+1)$ matrix $[A|b]$ to convert A into an **upper triangular** matrix $\tilde{A}$ with 1's on the diagonal and b into $\tilde{b}$.
2. Solve the system $\tilde{A}x = \tilde{b}$ by **back substitution**.

Exercise

Solve

$$Ax = b$$

for

$$A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \qquad b = (1, 1)$$

using Gaussian elimination.

```
def gaussian_elimination(A, b):
    A = np.copy(A)
    b = np.copy(b)
    n = len(b)
```

```python
for i in range(n):
    b[i] /= A[i, i]
    A[i] /= A[i, i]
    for j in range(i + 1, n):
        b[j] -= b[i] * A[j, i]
        A[j] -= A[i] * A[j, i]

for i in reversed(range(n-1)):
    b[i] -= b[i + 1:].dot(A[i, i + 1:])

return b
```

Problems with Gaussian elimination

- Very **slow** and **memory demanding** for large problems[2]

  Number of required operations is proportional to n^3 and the amount of memory to n^2 for matrix $n \times n$.

  We say that the **time complexity** of the algorithm is $O(n^3)$ and the **space complexity** is $O(n^2)$.

  In applications, n is commonly 10^6 and higher.

- Numerically **unstable** in some cases

  Needs special treatment to avoid problems with rounding errors (pivoting, …).

Next time

- Iterative methods for solving $Ax = b$

  Jacobi, Gauss-Seidel, SOR

  and their implementation…

Self study

- **Review** linear algebra notions discussed today.

Blyth, Robertson, Basic Linear Algebra

https://link.springer.com/book/10.1007%2F978-1-4471-3496-1

[2]But see Tridiagonal matrix algorithm for a banded matrix.