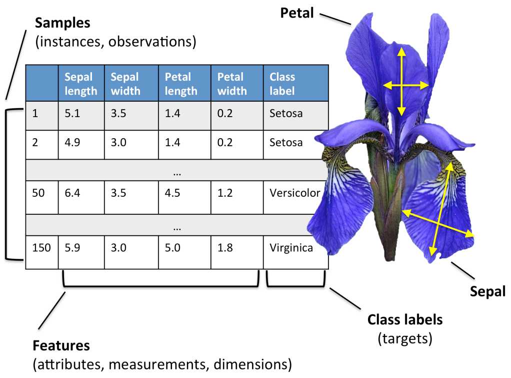
Introduction

Matrices are used to represent data, and linear algebra provides mathematical tools to understand and manipulate them in order to derive useful knowledge like, for instance, linear relations. It is a building block of many of the basic techniques of data analysis, including dimensionality reduction, machine learning, image processing, and language recognition.

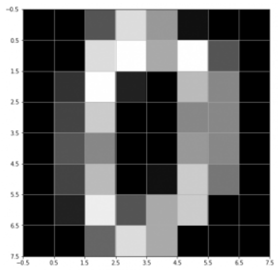
Data is usually collected in matrices, that is, numerical tables representing *samples* (or *data points*) with multiples *attributes* (or *variables*). Typically rows correspond to samples and columns to attributes, like in this representation of Fischer's iris flower dataset:

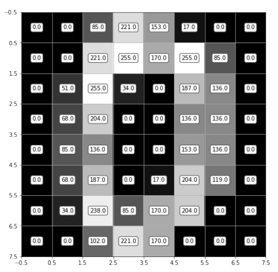


	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
			...		
50	6.4	3.5	4.5	1.2	Versicolor
			...		
150	5.9	3.0	5.0	1.8	Virginica

Samples
(instances, observations)
Petal
Sepal
Class labels
(targets)
Features
(attributes, measurements, dimensions)

Also digital b/w images are represented by matrices, since they are rectangular arrays of pixels, each of them coded by a value in the range from 0 (black) to 255 (white):





0.0	0.0	85.0	221.0	153.0	17.0	0.0	0.0
0.0	0.0	221.0	255.0	170.0	255.0	85.0	0.0
0.0	51.0	255.0	34.0	0.0	187.0	136.0	0.0
0.0	68.0	204.0	0.0	0.0	136.0	136.0	0.0
0.0	85.0	136.0	0.0	0.0	153.0	136.0	0.0
0.0	68.0	187.0	0.0	17.0	204.0	119.0	0.0
0.0	34.0	238.0	85.0	170.0	204.0	0.0	0.0
0.0	0.0	102.0	221.0	170.0	0.0	0.0	0.0

When confronted to a matrix, we might ask questions like:

- Are all the attributes independent?
- Can we identify the linear relationships?
- Can we reduce the size of the data matrix?

For instance, consider the 4×3 matrix

$$A = \begin{bmatrix} 1 & -1 & 0 \\ 2 & 0 & 2 \\ 3 & 2 & 5 \\ 4 & -3 & 1 \end{bmatrix}.$$

We have that $\mathrm{col}_1(A) + \mathrm{col}_2(A) - \mathrm{col}_3(A) = 0$ and so

$$\mathrm{col}_3(A) = \mathrm{col}_1(A) + \mathrm{col}_2(A),$$

that is, the third attribute can be written in terms of the first and the second one, which contain all the significant information. In general, the mathematical notion of *rank* encodes the number of linearly independent attributes. Linear algebra allows to compute it and to find the redundancies in a given data matrix, and thus to reduce its size.

More generally, we can ask how far is a matrix from being rank defective. Data coming from measurements is always inexact, and so we have to consider approximate linear relations between the different attributes. Methods like *principal components analysis (PCA)* and the closely related *singular value decomposition (SVD)* allow to detect these approximate linear relations and to estimate how much information is lost when reducing the matrix through them.

In this notes, we will study the three main problems of numerical linear algebra that are needed for these applications. *Linear equation solving* consist in solving

$$A\,x = b$$

for a given nonsingular $n \times n$ matrix A and n-vector b. The basic algorithms for this task are Gaussian elimination and its variants, like Cholesky's algorithm for positive symmetric matrices.

The *least squares problem* consists in computing the n-vector x minimizing the Euclidean norm

$$\|A\,x - b\|_2$$

for a given $m \times n$ matrix A and m-vector b. In data analysis, it appears prominently when fitting data with prefixed functions depending on a set of parameters.

Finally, the *eigenproblem* amounts to find for a given $n \times n$ matrix A, a scalar λ and a nonzero n-vector x such that

$$A\,x = \lambda\,x,$$

that is, the directions in which the linear map associated to A is a homothecy. The results and techniques for solving it also apply to the computation of the SVD. These problems play a central role in the design of Internet search machines through the Pagerank algorithm, and in dimensionality reduction through the PCA method.

Our approach to all these problems will be based on *matrix factorizations*, that is, representations of the given matrix as a product of simpler ones. For instance, *Gaussian elimination with partial pivoting (GEPP)* is based on the factorization of the given $n \times n$ matrix A as

$$A = P\,L\,U$$

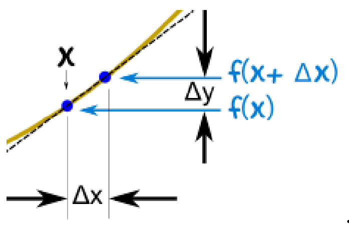
with P a permutation, L a unit lower triangular, and U upper triangular, like in Figure 0.0.1.
Solving the linear equation $A\,x = b$ then breaks into three easier tasks, done by permuting rows and applying forward and backward substitution.

$$\underbrace{\begin{bmatrix} & & 1 & \\ & & & 1 \\ & 1 & & \\ 1 & & & \end{bmatrix}}_{P}\underbrace{\begin{bmatrix} 2 & 1 & 1 & 0 \\ 4 & 3 & 3 & 1 \\ 8 & 7 & 9 & 5 \\ 6 & 7 & 9 & 8 \end{bmatrix}}_{A} = \underbrace{\begin{bmatrix} 1 & & & \\ \frac{3}{4} & 1 & & \\ \frac{1}{2} & -\frac{2}{7} & 1 & \\ \frac{1}{4} & -\frac{3}{7} & \frac{1}{3} & 1 \end{bmatrix}}_{L}\underbrace{\begin{bmatrix} 8 & 7 & 9 & 5 \\ & \frac{7}{4} & \frac{9}{4} & \frac{17}{4} \\ & & -\frac{6}{7} & -\frac{2}{7} \\ & & & \frac{2}{3} \end{bmatrix}}_{U}.$$

FIGURE 0.0.1. A PLU factorization

To ensure the reliability of the obtained results, we will be concerned with the *perturbation properties* of the considered problem and the *numerical stability* of our algorithms. There are two sources of numerical errors: those coming from the approximation of the input data, which is given by measurements and is subject to truncations, and those introduced by the algorithms.

Here the keyword is *condition numbers*. These are the parameters that measure the propagation of errors in a given problem, from the input to the output. For instance, let f be a real valued differentiable function. Then $f(x+\Delta x) \approx f(x) + f'(x)\,\Delta x$, and so the condition number is given by the derivative $f'(x)$.



Hence the first source of numerical errors is controlled by the condition number of the considered problem. If the algorithm is *backward stable*, then the second source is also controlled by this parameter.

We will also be concerned with the *efficiency* (or *speed*) of the proposed methods: before running a computation, it is useful to know how long it will take. The efficiency of an algorithm is modeled by the notion of *complexity*, which is the function measuring the number of floating point operations (flops) performed by this algorithm for a given input. For instance, GEPP solves an $n \times n$ linear system $A\,x = b$ with

$$\approx \frac{2}{3} n^3 \quad \text{flops.}$$

We can then use this knowledge to decide beforehand the feasibility of a certain computation on a specific machine.

Finally, it is also important to identify and exploit any special structure that might be present, in order to both increase speed and reduce the use of storage space. For instance, when A is symmetric and positive definite, Cholesky's algorithm solves the linear equation $A\,x = b$ with

$$\approx \frac{1}{3} n^3 \quad \text{flops,}$$

which is approximately half the complexity of GEPP. If moreover A is *banded* with band width $\approx \sqrt{n}$ like then Cholesky's algorithm uses only

$$O(n^2) \quad \text{flops,}$$

which represents a substantial saving for large values of n.