Time-Varying Systems and Computations
Introductory Lecture

Klaus Diepold

27. Oktober 2013

1 Tasks in Computational Science and Engineering

Computational Science is an important field of research that has an increasing impact in all disciplines of science and engineering. Engineers have to study this more mathematical subject matter in order to solve their problems. The engineering approach to problem solving also introduces characteristic viewpoints and technically motivated questions onto computations, which may not be covered and answered in their entirety by Mathematics. So engineers have an opportunity to contribute their own native methods to the toolbox of computational methods. The present course is an attempt to establish such an engineering viewpoint on scientific computation by developing system theoretic concepts and frameworks to better understand computational issues and to find new methods and solutions, which are amenable to solving engineering problems.

We can see three major domains, where engineers employ numerical computations for problem solving. These three domains are

1. Numerical simulation of physical phenomena – bottom-up computations,
2. Analysis of empirical data sets – top-down computations,
3. Numerical computations embedded in a physical world – cyber-physical systems.

Each of these domains comes with its own particular mathematical structural characteristics. Also, the computations involved and their respective implementations need to satisfy technical requirements in order to be useful from an engineering point of view. I will discuss all three domains in a little more detail to clarify the particularities of each domain.

1.1 Numerical Simulation

1.1.1 Characteristics

Simulation-based engineering work utilizes mathematical descriptions of physical phenomena, computing the static and dynamic properties of these physical phenomena by evaluating the corresponding formulas numerically. The mathematical descriptions come in the form of equations, which may be linear or non-linear, the may come as any kind of ordinary differential equations or as one of many variations of partial differential equations, depending on the problem at hand. For numerical computations these equations need to be linearized and discretized, solutions include numerical differentiation and integration techniques. For actually solving a numerical simulation problem we need real data for either setting the boundary conditions (boundary value problem) or for setting the initial conditions (initial value problem).

Real data may come from application specific considerations or from real world measurements. Setting the equations for complex phenomena often is based on relatively simple equations to describe the behavior of elementary subsystems. A large number of such elementary subsystems are then put together to establish a large-scale simulation for the entire system. Take circuit-simulation as an example, where the relationship between the current and the voltage at an elementary electronic device is described by simple differential equations. All the electronic devices comprised in an complete circuit or even chip are then combined by Kirchhoff equations. the result is a large algebro-differential system of equations which needs to be solved numerically.

Since large-scale computations are compiled of many elementary computational models we also refer to this as a *bottom-up* approach to computational engineering.

1.1.2 Typical Application Examples

There exists a long list of application domains that employ a bottom-up strategy to numerical computations. This domain is characterized by the existence of mathematical formulas to describe the physical behavior of elementary building elements. Typical examples are:

- Circuit Simulation (analog and digital),
- Weather Forecast,
- Aerodynamics,
- Fluid Dynamics,
- Particle Physics,
- High-Energy Physics,
- Geology,
- Geography,
- Hydrology,
- Structural Design,
- Electromagnetic Wave Propagation,
- Light and matter transport,
- ...

1.2 Data Analysis

1.2.1 Characteristics

Engineers and scientists analyze empirical measurement data to find out some sort of relationship or physical phenomena that is reflected in the data. This is a data-driven approach to science and engineering. The assumption is that a underlying physical law creates structures and patterns in the data, which needs to be identified and extracted by means of data analysis techniques. In many cases we assume that we can apply a linear model. We can use linear models even for non-linear relations if we linearize this relation around an operation point using a differential approach. Such an approach also requires techniques to estimate the complexity of the models, which is often expressed in terms of the rank of a matrix or the dimension of a dominant subspace spanned by the data samples. Georg Simon Ohm employed such

an approach to find out about the relation between current and voltage, which lead to the concept of resistance.

The matrices that we encounter in data analysis tasks are often very large, they are mostly dense matrices (i.e. all or nearly all matrix entries are different from zero) and they may not exhibit clearly visible structure. However, the data matrices often exhibit low-rank structures and some sort of hidden structure, i.e. some non-trivial dependencies between the matrix entries, which is not known a priori. While the corresponding numerical and statistical techniques have been around for some time, they are re-invented in recent times under the label of 'machine learning'. The effectiveness of data-analytical and statistical techniques is one aspect for research and development. The efficiency of the corresponding computations is another aspect with high impact for practical purposes.

1.2.2 Typical Application Examples

The list of application examples in this category is growing by the minute. Some of the key words in this context are summarized in the following list.

- Data Mining,
- Internet Search (e.g. Page Rank),
- Machine Learning (supervised and unsupervised)
- Statistical Data Analysis,
- Process Control,
- User Studies,
- Analysis of Measurement Data
- ...

1.3 Embedded Systems

1.3.1 Characteristics

Many technical systems that interact with the real world contain parts of the internal workings which are implemented in terms of a computer. Such a computer is interfacing with the real world through sensors and actuators ad are also termed 'caber-physical systems'. The physical behavior is implemented by a numerical algorithm, which needs to be implemented on a computing platform that has restricted computational resources (limited computational capabilities and small memory capacity. As a consequence this type of applications requires efficient numerical algorithms, in particular if such an embedded system shall incorporate functionalities requiring increased levels of intelligence.

1.3.2 Typical Application Examples

- Mobile Communications,
- Automotive Applications (Driver Assistance Systems),
- Process Control,
- Aeronautical Control,
- Cyber-Physical Systems,

- Robotics,
- Automation Systems
- ...

2 Linear Algebra and Linear Systems

2.1 Standard Representation of Linear Systems

We can use the equivalence between Linear Systems on the one hand and Linear Algebra on the other hand if we want to represent such systems or if we are computing with input and output signals of linear systems. The engineer and researcher formulates all related problems in terms of the standard computational problem using the matrix-vector notation

$$T \cdot u = y, \tag{1}$$

where T is a $m \times n$-matrix with real- or complex-valued entries. The variable u denotes a vector containing n real- or complex-valued entries of the input signal and y represents the m-vector containing the real- or complex-valued values of the output signal.

2.2 Standard Problem Formulations

Using equation 1 we can distinguish between three different standard problems, depending on which of the three variables are supposed to be known.

- Matrix T and vector u are known and we wish to compute the vector y. This can be seen as feeding a linear system, which is represented by the matrix T with an input signal u and we wish to determine the output signal y. This problem can be briefly represented as

  $$(T, u) \mapsto y.$$

  We denote this operation as *filtering*.

- Vectors u and y are known and we wish to determine the matrix T from the observations of input and output signals. This problem can be briefly represented as

  $$(u, y) \mapsto T.$$

  We denote this operation as *system identification*.

- Matrix T is known along with the vector y and we wish to compute the input signal that caused the output signal y. This problem can be briefly represented as

  $$(T, y) \mapsto u.$$

  We use the term *inversion problem* or *signal estimation* refer to this operation.

2.3 Typical examples for numerical linear algebra computations

Numerical linear algebra is composed of many elementary computations, which require arithmetic operations. The number of elementary arithmetic operations required to compute such elementary results determine their computational complexity. The number of arithmetic operations is one measure to assess the efficiency of a given algorithm. It is a costly effort to determine the exact operation count for a given computation. For engineers it is often helpful to know the asymptotic complexity of an algorithm, that is, to know how the number of operations grows with the size of the problem. The asymptotic behavior amounts to look for very large values of the parameter n, which describes the size of the problem to be solved. For elementary linear algebra computations the following complexities are known:

- Inner product $x^T y$ – $\mathcal{O}(n)$ operations
- Matrix-vector product $Ax = y$ – $\mathcal{O}(n^2)$ operations
- Matrix addition $A + B = C$ – $\mathcal{O}(n^2)$ operations
- Matrix-matrix product $AB = C$ – $\mathcal{O}(n^3)$ operations
- Matrix inversion A^{-1} – $\mathcal{O}(n^3)$ operations

For those asymptotic values we assume that the matrices involved have no particular structure that we could exploit to reduce the amount of computations. We further assume that the matrices are of size $n \times n$ and the vectors have corresponding sizes.

2.4 Data Sparse Matrices

In general, for a given $n \times n$-matrices we require memory for storing n^2 matrix entries. However, in engineering applications we often encounter matrices, which are data sparse, that is, where we need much less memory to store these matrices. Such matrices have 'structure' or their degrees of freedom are otherwise reduced. Simple examples for matrices with structure are symmetric or triangular matrices, which require to store only $n(n+1)/2$ values or rotation matrices, which require only $n(n-1)/2$ values. However, these matrices still consist of $\mathcal{O}(n^2)$ free values.

For our discussion we are considering matrices which have $\mathcal{O}(n)$ free parameters. While such matrices require a reduced amount of storage space we want to find computational schemes for such matrices, which also need only $\mathcal{O}(n)$ computations for e.g. matrix-vector multiplication, or $\mathcal{O}(n^2)$ for matrix-matrix multiplication or inversion.

Some matrices have many zero-entries such that there are $\mathcal{O}(n)$ non-zero entries left, which need to be stored. There are other matrices where the matrix entries exhibit inner relationships such that only $\mathcal{O}(n)$ entries need to be stored.

- Diagonal matrices

$$D = \begin{bmatrix} d_1 & & & & \\ & d_2 & & & \\ & & d_3 & & \\ & & & \ddots & \\ & & & & d_n \end{bmatrix}$$

- Block-diagonal matrices

$$\boldsymbol{D} = \begin{bmatrix} \boxed{D_1} & & & & \\ & \boxed{D_2} & & & \\ & & \boxed{D_3} & & \\ & & & \ddots & \\ & & & & \boxed{D_n} \end{bmatrix}$$

- Sparse matrices

$$D = \begin{bmatrix} d_0 & & d_1 & & d_2 \\ & & & d_3 & \\ d_4 & & & & \\ & d_5 & d_6 & & d_7 \\ d_8 & & & & d_9 \end{bmatrix}$$

- Toeplitz-, and Hankel-, Vandermonde-matrices

$$T = \begin{bmatrix} t_0 & t_1 & t_2 & t_3 & t_4 \\ t_{-1} & t_0 & t_1 & t_2 & t_3 \\ t_{-2} & t_{-1} & t_0 & t_1 & t_2 \\ t_{-3} & t_{-2} & t_{-1} & t_0 & t_1 \\ t_{-4} & t_{-3} & t_{-2} & t_{-1} & t_0 \end{bmatrix}, \qquad H = \begin{bmatrix} h_0 & h_1 & h_2 & h_3 & h_4 \\ h_1 & h_2 & h_3 & h_4 & h_5 \\ h_2 & h_3 & h_4 & h_5 & h_6 \\ h_3 & h_4 & h_5 & h_6 & h_7 \\ h_4 & h_5 & h_6 & h_7 & h_8 \end{bmatrix}$$

$$V = \begin{bmatrix} 1 & v_1 & v_1^2 & v_1^3 & v_1^4 \\ 1 & v_2 & v_2^2 & v_2^3 & v_2^4 \\ 1 & v_3 & v_3^2 & v_3^3 & v_3^4 \\ 1 & v_4 & v_4^2 & v_4^3 & v_4^4 \\ 1 & v_5 & v_5^2 & v_5^3 & v_5^4 \end{bmatrix}$$

- Low-rank matrices

$$A = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{bmatrix} \cdot \begin{bmatrix} v_1 & v_2 & v_3 & v_4 & v_5 \end{bmatrix} = \begin{bmatrix} u_1v_1 & u_1v_2 & u_1v_3 & u_1v_4 & u_1v_5 \\ u_2v_1 & u_2v_2 & u_2v_3 & u_2v_4 & u_2v_5 \\ u_3v_1 & u_3v_2 & u_3v_3 & u_3v_4 & u_3v_5 \\ u_4v_1 & u_4v_2 & u_4v_3 & u_4v_4 & u_4v_5 \\ u_5v_1 & u_5v_2 & u_5v_3 & u_5v_4 & u_5v_5 \end{bmatrix}$$

- Band matrices

$$B = \begin{bmatrix} b_{11} & b_{12} & & & \\ b_{21} & b_{22} & b_{23} & & \\ & b_{32} & b_{33} & b_{34} & \\ & & b_{43} & b_{44} & b_{45} \\ & & & b_{54} & b_{55} \end{bmatrix}$$

- ... and many more types of structured matrices, including block versions such as block-Toeplitz and Toeplitz-block matrices

$$B_T = \begin{bmatrix} B_1 & B_2 & B_3 & & \\ B_2 & B_1 & B_2 & B_3 & \\ B_3 & B_2 & B_1 & B_2 & B_3 \\ & B_3 & B_2 & B_1 & B_2 \\ & & B_3 & B_2 & B_1 \end{bmatrix}, \qquad T_B = \begin{bmatrix} b_{11} & b_{12} & & & \\ b_{21} & b_{22} & b_{23} & & \\ & b_{32} & b_{33} & b_{34} & \\ & & b_{43} & b_{44} & b_{45} \\ & & & b_{54} & b_{55} \end{bmatrix}$$

It is fair to say that in engineering applications we mostly encounter either dense matrices, which exhibit exploitable structure or we encounter sparse matrices. Most of the standard algorithms engineering students learn in the context of signal processing are efficient versions of linear algebra algorithms. Foe sparse matrices mathematicians and numerical analysts have developed a whole suite of specialized algorithms that allow to take advantage of the sparsity of matrices.

2.5 Evaluation of Numerical Computations

2.5.1 Evaluation Criteria

When solving a problem we are faced frequently with a choice among algorithms. On what basis should we choose? There are two often contradictory goals.

1. We would like an algorithm that is easy to understand, code, and debug.
2. We would like an algorithm that makes efficient use of the computer's resources, especially, one that runs as fast as possible.

When we are writing a program to be used once or a few times, goal (1) is most important. The cost of the programmer's time will most likely exceed by far the cost of running the program, so the cost to optimize is the cost of writing the program. When presented with a problem whose solution is to be used many times, the cost of running the program may far exceed the cost of writing it, especially, if many of the program runs are given large amounts of input. Then it is financially sound to implement a fairly complicated algorithm, provided that the resulting program will run significantly faster than a more obvious program. Even in these situations it may be wise first to implement a simple algorithm, to determine the actual benefit to be had by writing a more complicated program. In building a complex system it is often desirable to implement a simple prototype on which measurements and simulations can be performed, before committing oneself to the final design. It follows that programmers must not only be aware of ways of making programs run fast, but must know when to apply these techniques and when not to bother.

2.5.2 Runtime of a program

The running time of a program depends on factors such as:

1. the input to the program,
2. the quality of code generated by the compiler used to create the object program,
3. the nature and speed of the instructions on the machine used to execute the program,
4. the level of parallelism supported by the computational platform used for implementation, and
5. the time complexity of the algorithm underlying the program.

The fact that running time depends on the input tells us that the running time of a program should be defined as a function of the input. Often, the running time depends not on the exact input but only on the ßizeöf the input. A good example is the process known as sorting, which we shall discuss in Chapter 8. In a sorting problem, we are given as input a list of items to be sorted, and we are to produce as output the same items, but smallest (or largest) first. For example, given 2, 1, 3, 1, 5, 8 as input we might wish to produce 1, 1, 2, 3, 5, 8 as output. The latter list is said to be sorted smallest first. The natural size measure for inputs to a sorting program is the number of items to be sorted, or in other words, the length of the input list. In general, the length of the input is an appropriate size measure, and we shall assume that measure of size unless we specifically state otherwise.

Literatur

[1] G. Strang. *Computational Science and Engineering*. Wellesley-Cambridge Press, 2007.