Introduction to Computational Physics

Lecture of Prof. H. J. Herrmann

Swiss Federal Institute of Technology ETH, Zürich, Switzerland

Script by
Dr. H. M. Singer, Lorenz Müller and Marco - Andrea Buchmann

Computational Physics, IfB, ETH Zürich

General Information

Useful Addresses and Information

The content of this class is available online on the following pages:

- http://www.comphys.ethz.ch/index.php/lectures
- http://www.ifb.ethz.ch/education/IntroductionComPhys

Pdf-files of both the slides and the exercises are also provided on these two pages.

Who is the Target Audience of This Lecture?

The lecture gives an introduction to computational physics for students of the following departments:

- Mathematics and Computer Science (Bachelor and Master course)
- Physics (major course, "Wahlfach")
- Material Science (Master course)
- Civil Engineering (Master course)

Some Words About Your Teacher

Prof. Hans. J. Herrmann is full professor at the Institute of Building Materials (IfB) since April 2006. His field of expertise is computational and statistical physics, in particular granular materials. His present research subjects include dense colloids, the formation of river deltas, quicksand, the failure of fibrous and polymeric composites and complex networks.
Prof. Herrmann can be reached at

hjherrmann@ethz.ch

His office is located in the Institute of Building materials (IfB), HIF E12, ETH Hönggerberg, Zürich.
The personal web page is located at www.icp.uni-stuttgart.de/~hans and at www.comphys.ethz.ch

Some Words About the Authors

The script was started in 2007 by Dr. H.M. Singer, who wrote large parts of the chapter on random number generators, a part of the chapter on percolation, and large parts of the chapters on solving equations.

In 2009/2010, Lorenz Müller and Marco - Andrea Buchmann continued the script and filled in the gaps, expanded said chapters and added chapters on Monte Carlo methods, fractal dimensions and the Ising model.

If you have suggestions or corrections, please let us know:

marcobuchmann@student.ethz.ch

Thank you!

Outline of This Course

This course consists of two parts:

In a first part we are going to consider stochastic processes, such as percolation. We shall look at random number generators, Monte Carlo methods and the Ising model, particularly their applications.

In the second part of the class we shall look into numerical ways of solving equations (e.g. ordinary differential equations). We shall get to know a variety of ways to solve these and learn about their advantages as well as their disadvantages.

Prerequisites for this Class

- You should have a basic understanding of the UNIX operating system and be able to work with it. This means concepts such as the 'shell', stream redirections and compiling programs should be familiar.
- You should ideally have some knowledge about a higher level programming language such as Fortran, C/C++, Java, ... In particular you should also be able to write, compile and debug programs yourself.
- It is beneficial to know how to make scientific plots. There are many tools, which can help you with that. For example Matlab, Maple, Mathematica, R, SPlus, gnuplot, etc.
- Requirements in mathematics:
  - You should know the basics of statistical analysis (averaging, distributions, etc.).
  - Furthermore, some knowledge of linear algebra and analysis will be necessary.
- Requirements in physics
  - You should be familiar with Classical Mechanics (Newton, Lagrange) and Electrodynamics.
  - A basic understanding of Thermodynamics is also beneficial.

Contents

What is Computational Physics?

Computational physics is the study and implementation of numerical algorithms to solve problems in physics by means of computers. Computational physics in particular solves equations numerically. Finding a solution numerically is useful, as there are very few systems for which an analytical solution is known. Another field of computational physics is the simulation of many-body/particle systems; in this area, a virtual reality is created which is sometimes also referred to as the 3rd branch of physics (between experiments and theory).

The evaluation and visualization of large data sets, which can come from numerical simulations or experimental data (for example maps in geophysics) is also part of computational physics.

Another area in which computers are used in physics is the control of experiments. However, this area is not treated in the lecture.

Computational physics plays an important role in the following fields:

- Computational Fluid Dynamics (CFD): solve and analyze problems that involve fluid flows
- Classical Phase Transition: percolation, critical phenomena
- Solid State Physics (Quantum Mechanics)
- High Energy Physics / Particle Physics: in particular Lattice Quantum Chromodynamics ("Lattice QCD")
- Astrophysics: many-body simulations of stars, galaxies etc.
- Geophysics and Solid Mechanics: earthquake simulations, fracture, rupture, crack propagation etc.
- Agent Models (interdisciplinary): complex networks in biology, economy, social sciences and many others

Suggested Literature

Books:

- H. Gould, J. Tobochnik and Wolfgang Christian: „*Introduction to Computer Simulation Methods*“ 3rd edition (Addison Wesley, Reading MA, 2006).
- D. P. Landau and K. Binder: „*A Guide to Monte Carlo Simulations in Statistical Physics*“ (Cambridge University Press, Cambridge, 2000).
- D. Stauffer, F. W. Hehl, V. Winkelmann and J. G. Zabolitzky: „*Computer Simulation and Computer Algebra*“ 3rd edition (Springer, Berlin, 1993).
- K. Binder and D. W. Heermann: „*Monte Carlo Simulation in Statistical Physics*“ 4th edition (Springer, Berlin, 2002).
- N. J. Giordano: „*Computational Physics*“ (Addison Wesley, Reading MA, 1996).
- J. M. Thijssen: “*Computational Physics*”, (Cambridge University Press, Cambridge, 1999).

Book Series:

- „*Monte Carlo Method in Condensed Matter Physics*“, ed. K. Binder (Springer Series).
- „*Annual Reviews of Computational Physics*“, ed. D. Stauffer (World Scientific).
- „*Granada Lectures in Computational Physics*“, ed. J. Marro (Springer Series).
- „*Computer Simulations Studies in Condensed Matter Physics*“, ed. D. Landau (Springer Series).

Journals:

- Journal of Computational Physics (Elsevier).
- Computer Physics Communications (Elsevier).
- International Journal of Modern Physics C (World Scientific).

Conferences:

- Annual Conference on Computational Physics (CCP): In 2007, the CCP was held in Brussels (Sept. 5th-8th 2007), in 2008 it was in Brazil.

Part I

Stochastic Processes

Chapter 1

Random Numbers

Random numbers (RN) are an important tool for scientific simulations. As we shall see in this class, they are used in many different applications, including the following:

- Simulate random events and experimental fluctuations, for example radioactive decay
- Complement the lack of detailed knowledge (e.g. traffic or stock market simulations)
- Consider many degrees of freedom (e.g. Brownian motion, random walks).
- Test the stability of a system with respect to perturbations
- Random sampling

Special literature about random numbers is given at the end of this chapter.

1.1 Definition of Random Numbers

Random numbers are a sequence of numbers in random or uncorrelated order. In particular, the probability that a given number occurs next in the sequence is always the same. Physical systems can produce random events, for example in electronic circuits ("electronic flicker noise") or in systems where quantum effects play an important role (such as for example radioactive decay or the photon emission from a semiconductor). However, physical random numbers are usually "bad" in the sense that they are usually correlated.

The algorithmic creation of random numbers is a bit problematic, since the computer is completely deterministic but the sequence should be non-deterministic. One therefore considers the creation of pseudo-random numbers, which are calculated with a deterministic algorithm, but in such a way that the numbers are

almost homogeneously, randomly distributed. These numbers should follow a well-defined distribution and should have long periods. Furthermore, they should be calculated quickly and in a reproducible way.

A very important tool in the creation of pseudo-random numbers is the modulo-operator **mod** (in C++ %), which determines the remainder of a division of one integer number with another one.

Given two numbers a (dividend) and n (divisor), we write a **modulo** n or a **mod** n which stands for the remainder of division of a by n. The mathematical definition of this operator is as follows: We consider a number $q \in \mathbb{Z}$, and the two integers a and n mentioned previously. We then write a as

$$a = nq + r \tag{1.1}$$

with $0 \leq r < |n|$, where r is the the remainder. The **mod**-operator is useful because one obtains both big and small numbers when starting with a big number.

The pseudo-random number generators (RNG) can be divided into two classes: the multiplicative and the additive generators.

- The multiplicative ones are simpler and faster to program and execute, but do not produce very good sequences.
- The additive ones are more difficult to implement and take longer to run, but produce much better random sequences.

1.2 Congruential RNG (Multiplicative)

The simplest form of a congruential RNG was proposed by Lehmer in 1948. The algorithm is based on the properties of the **mod**-operator. Let us assume that we choose two integer numbers c and p and a seed value x_0 with $c, p, x_0 \in \mathbb{Z}$. We then create the sequence $x_i \in \mathbb{Z}$, $i \in \mathbb{N}$ iteratively by

$$x_i = (cx_{i-1}) \,\mathbf{mod}\, p \tag{1.2}$$

This creates random numbers in the interval $[0, p-1]$ [1]. In order to transform these random numbers to the interval $[0, 1[$ we simply divide by p

$$0 \leq z_i = \frac{x_i}{p} < 1 \tag{1.3}$$

[1] Throughout the manuscript we will adopt the notation that closed square brackets [] in intervals are equivalent to $\leq$ and $\geq$ and open brackets][correspond to $<$ and $>$ respectively. Thus the interval $[0, 1]$ corresponds to $0 \leq x \leq 1$, $x \in \mathbb{R}$ and $]0, 1]$ means $0 < x \leq 1$, $x \in \mathbb{R}$.

with $z_i \in \mathbb{R}$ (actually $z_i \in \mathbb{Q}$).

Since all integers are smaller than p the sequence must repeat after at least $(p-1)$ iterations. Thus, the *maximal period* of this RNG is $(p-1)$. If we pick the seed value $x_0 = 0$, the sequence sits on a fixed point 0 (therefore, $x_0 = 0$ cannot be used).

In 1910, R. D. Carmichael proved that the maximal period can be obtained if p is a Mersenne prime number[2] and if the number is at the same time the smallest integer number for which the following condition holds:

$$c^{p-1} \bmod p = 1. \tag{1.4}$$

In 1988, Park and Miller presented the following numbers, which produce a relatively long sequence of pseudo-random numbers, here in pseudo-C code:

```
const int p=2147483647;
const int c=16807;
int rnd=42;  // seed
rnd=(c*rnd)%p;
print rnd;
```

The number p is of course a Mersenne prime with the maximal length of an integer (32 bit): $2^{31} - 1$.

The distribution of pseudo-random numbers calculated with a congruential RNG can be represented in a plot of consecutive random numbers (x_i, x_{i+1}), where they will form some patterns (mostly lines) depending on the chosen parameters. It is of course also possible to do this kind of visualization for three consecutive numbers (x_i, x_{i+1}, x_{i+2}) in 3D (see e.g. Fig. 1.1). There is even a theorem which quantifies the patterns observed. Let us call the normalized numbers of the pseudo-random sequence $\{z_i\} = \{x_i\}/p$, $i \in \mathbb{N}$. Let then $\pi_1 = (z_1, ..., z_n)$, $\pi_2 = (z_2, ..., z_{n+1})$, $\pi_3 = (z_3, ..., z_{n+2})$, ... be the points of the unit n-cube formed from n successive z_i.

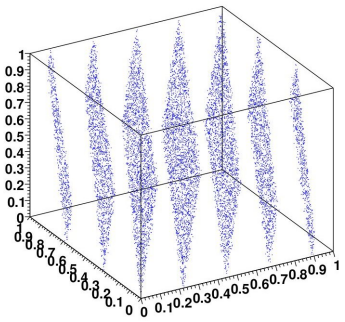


Figure 1.1: Sample plot of consecutive random numbers with clearly visible hyperplanes ("RANDU" algorithm with $c = 65539, p = 2^{31}, x_0 = 1$) [20]

[2] A *Mersenne number* is defined as $M_n = 2^n - 1$. If this number is also prime, it is called a *Mersenne prime*.

Theorem 1 (Marsaglia, 1968)**.** *If $a_1, a_2, ..., a_n \in \mathbb{Z}$ is any choice of integers such that*

$$a_1 + a_2 c + a_3 c^2 + ... + a_n c^{n-1} \equiv 0 \,\mathbf{mod}\, p,$$

then all of the points $\pi_1, \pi_2, ...$ will lie in the set of parallel hyperplanes defined by the equations

$$a_1 y_1 + a_2 y_2 + ... + a_n y_n = 0, \pm 1, \pm 2, ..., \;\; y_i \in \mathbb{R},\, 1 \leq i \leq n$$

There are at most

$$|a_1| + |a_2| + ... + |a_n|$$

of these hyperplanes, which intersect the unit n-cube and there is always a choice of $a_1, ..., a_n$ such that all of the points fall in fewer than $(n!p)^{1/n}$ hyperplanes.

Proof. (Abbreviated) The theorem is proved in four steps: *Step 1*: If

$$a_1 + a_2 c + a_3 c^2 + ... + a_n c^{n-1} \equiv 0 \,\mathbf{mod}\, p$$

then one can prove that

$$a_1 z_i + a_2 z_{i+1} + ... + a_n z_{i+n-1}$$

is an integer for every i and thus
Step 2: The point $\pi_i = (z_i, z_{i+1}, .., z_{i+n-1})$ must lie in one of the hyperplanes

$$a_1 y_1 + a_2 y_2 + ... + a_n y_n = 0, \pm 1, \pm 2, ..., \;\; y_i \in \mathbb{R},\, 1 \leq i \leq n.$$

Step 3: The number of hyperplanes of the above type, which intersect the unit n-cube is at most

$$|a_1| + |a_2| + ... + |a_n|,$$

and
Step 4: For every multiplier c and modulus p there is a set of integers $a_1, ..., a_n$ (not all zero) such that

$$a_1 + a_2 c + a_3 c^2 + ... + a_n c^{n-1} \equiv 0 \,\mathbf{mod}\, p$$

and

$$|a_1| + |a_2| + ... + |a_n| \leq (n!p)^{1/n}.$$

This is of course only the outline of the proof. The exact details can be read in G. Marsaglia, Proc. Nat. Sci. U.S.A. **61**, 25 (1968). □

In a very similar way it is possible to show that for congruential RNGs the distance between the planes must be larger than

$$\sqrt{\frac{p}{n}} \tag{1.5}$$

1.3 Lagged Fibonacci RNG (Additive)

A more complicated version of a RNG is the Lagged Fibonacci algorithm proposed by Tausworth in 1965. Lagged Fibonacci type generators permit extremely large periods and even allow for advantageous predictions about correlations.

Lagged Fibonacci generators often use integer values, but in the following we shall focus on binary values.

Consider a sequence of of binary numbers $x_i \in \{0, 1\},\ 1 \leq i \leq b$. The next bit in our sequence, x_{b+1} is then given by

$$x_{b+1} = (\sum_{j \in \mathcal{J}} x_{b+1-j}) \,\mathbf{mod}\, 2 \tag{1.6}$$

with $\mathcal{J} \subset [1, .., b]$. In other words, the sum includes only a subset of all the other bits, so the new bit could for instance simply be based on the first and third bit, $x_{b+1} = (x_1 + x_3)\mathbf{mod}\, 2$ (or of course any other subset!).

Let us try to illustrate some points with a two element lagged Fibonacci generator. Consider two natural numbers $c, d \in \mathbb{N}$ with $d \leq c$, and we define our sequence recursively as

$$x_{i+1} = (x_{i-c} + x_{i-d}) \,\mathbf{mod}\, 2$$

Of course we immediately see that we need some initial sequence of at least c bits to start from (a so-called seed sequence). One usually uses a congruential generator to obtain the seed sequence.

Much as in the case of congruential generators, there are conditions for the choice of the numbers c and d. In this case, c and d must satisfy the Zierler-Trinomial condition which states that

$$T_{c,d}(z) = 1 + z^c + z^d \tag{1.7}$$

cannot be factorized in subpolynomials, where z is a binary number. The number c is chosen up to $100,000$ and it can be shown that the maximal period is $2^c - 1$, which is much larger than for congruential generators. The smallest numbers satisfying the Zierler conditions are $(c, d) = (250, 103)$. The generator is named after the discoverers of the numbers, Kirkpatrick and Stoll (1981).

The following pairs (c, d) are known:

(c, d)			
(250	,	103)	**Kirkpatrick – Stoll (1981)**
(4187	,	1689)	**J.R. Heringa et al. (1992)**
(132049	,	54454)	
(6972592	,	3037958)	**R.P. Brent et al. (2003)**

1.3.1 Implementation

There are two methods to convert the obtained binary sequences to natural numbers (e.g. 32 bit unsigned variables):

- One runs 32 Fibonacci generators in parallel (this can be done very efficiently). The problem with this method is the initialization, as the 32 initial sequences do not only need to be uncorrelated each one by itself but also among each other. The quality of the initial sequences has a major impact on the quality of the produced random numbers.
- One extracts a 32 bit long part from the sequence. This method is relatively slow, as for each random number one needs to generate 32 new elements in the binary sequence. Furthermore, it has been shown that random numbers produced in this way show strong correlations.

1.4 How Good is a RNG?

There are many possibilities to test how random a sequence generated by a given RNG really is. There is an impressive collection of possible tests for a given sequence $\{s_i\}$, $i \in \mathbb{N}$, for instance:

1. Square test: the plot of two consecutive numbers $(s_i, s_{i+1})\,\forall i$ should be distributed homogeneously. Any sign of lines or clustering shows the non-randomness and correlation of the sequence $\{s_i\}$.
2. Cube test: this test is similar to the square test, but this time the plot is three-dimensional with the tuples (s_i, s_{i+1}, s_{i+2}). Again the tuples should be distributed homogeneously.
3. Average value: the arithmetic mean of all the numbers in the sequence $\{s_i\}$ should correspond to the analytical mean value. Let us assume here that the numbers s_i are rescaled to be in the interval $s_i \in [0, 1[$. The arithmetic mean should then be

$$\bar{s} = \lim_{N\to\infty} \frac{1}{N} \sum_{i=1}^{N} s_i = \frac{1}{2} \tag{1.8}$$

So the more numbers are averaged, the better $\frac{1}{2}$ will be approximated.

4. Fluctuation of the mean value (χ^2-test): the distribution around the mean value should behave like a Gaussian distribution.

5. Spectral analysis (Fourier analysis): If we assume that the $\{s_i\}$ are values of a function, it is possible to perform a Fourier transform by means of the Fast Fourier Transform (FFT). If the frequency distribution corresponds to white noise (uniform distribution), the randomness is good, otherwise peaks will show up (resonances).

6. Correlation test: Analysis of correlations such as

$$< s_i * s_{i+d} > - < s_i^2 > \tag{1.9}$$

for different d.

Of course this list is not complete. There are many other tests that can be used to check the randomness of pseudo-random sequences.

Very famous are Marsaglia's "Diehard" tests for random numbers. These Diehard tests are a battery of statistical tests for measuring the quality of a set of random numbers. They were developed over many years and published for the first time by Marsaglia on a CD-ROM with random numbers in 1995. These tests are [3]:

- Birthday spacings: If random points are chosen in a large interval, the spacing between the points should be asymptotically Poisson distributed. The name stems from the birthday paradox[4].

- Overlapping permutations: When analyzing five consecutive random numbers, the 120 possible orderings should occur with statistically equal probability.

- Ranks of matrices: Some number of bits from some number of random numbers are formed to a matrix over {0,1}. The rank of this matrix is then determined and the ranks are counted.

[3]The tests are no better or worse than the ones presented previously. They have become famous though thanks to their rather creative naming.

[4]The birthday paradox states that the probability of two randomly chosen persons having the same birthday in a group of 23 (or more) people is more than 50%. In case of 57 or more people the probability is already more than 99%. Finally for at least 366 people the probability is exactly 100%. This is not paradoxical in a logical sense, it is called paradox nevertheless since intuition would suggest probabilities much lower than 50%.

- Monkey test: Sequences of some number of bits are taken as words and the number of overlapping words in a stream is counted. The number of words not appearing should follow a known distribution. The name is based on the infinite monkey theorem[5].

- Parking lot test: Randomly place unit circles in a 100 x 100 square. If the circle overlaps an existing one, try again. After 12,000 tries, the number of successfully "parked" circles should follow a certain normal distribution.

- Minimum distance test: Find the minimum distance of 8000 randomly placed points in a 10000 x 10000 square. The square of this distance should be exponentially distributed with a certain mean.

- Random spheres test: put 4000 randomly chosen points in a cube of edge 1000. Now a sphere is placed on every point with a radius corresponding to the minimum distance to another point. The smallest sphere's volume should then be exponentially distributed.

- Squeeze test: 2^{31} is multiplied by random floats in $[0, 1[$ until 1 is reached. After 100,000 repetitions the number of floats needed to reach 1 should follow a certain distribution.

- Overlapping sums test: Sequences of 100 consecutive floats are summed up in a very long sequence of random floats in $[0, 1[$. The sums should be normally distributed with characteristic mean and standard deviation.

- Runs test: Ascending and descending runs in a long sequence of random floats in $[0, 1[$ are counted. The counts should follow a certain distribution.

- Craps test: 200,000 games of craps[6] are played. The number of wins and the number of throws per game should follow a certain distribution.

1.5 Non-Uniform Distributions

We have so far only considered the uniform distribution of pseudo-random numbers. The congruential and lagged Fibonacci RNG produce numbers in $\mathbb{N}$ which can easily be mapped to the interval $[0, 1[$ or any other interval by simple shifts and multiplications. However, if the goal is to produce random numbers which are distributed according to a certain distribution (e.g. Gaussian), the algorithms presented so far are not very well suited. There are however tricks that permit us

[5]The infinite monkey theorem states that a monkey hitting keys at random on a typewriter keyboard for an infinite amount of time will almost surely (i.e. with probability 1) type a particular chosen text, such as the complete works of William Shakespeare.

[6]Dice game

to transform uniform pseudo-random numbers to other distributions. There are essentially two different ways to perform this transformation:

- If we are looking at a distribution whose analytic description is known, it may be possible to apply a mapping
- However, if the analytic description is unknown (or the transformation cannot be applied), we have to use the so-called rejection method.

These methods are explained in the following sections.

1.5.1 Transformation Methods of Special Distributions

For a certain class of distributions it is possible to create pseudo-random numbers from uniformly distributed random numbers by finding a mathematical transformation. The transformation method works particularly nicely for the most common distributions (exponential, Poisson and normal distribution). While the transformation is rather straightforward, it is not always feasible - this depends on the analytical description of the distribution. The idea is to find the equivalence between area slices of the uniform distribution P_u and the distribution of interest. The uniform distribution is written as

$$P_u(z) = \begin{cases} 1 & \text{for } z \in [0,1] \\ 0 & \text{otherwise} \end{cases} \tag{1.10}$$

Let us now consider the distribution $P(y)$. If we compare the areas of intergration, we find

$$z = \int_0^y P(y')dy' = \int_0^z P_u(z')dz' \tag{1.11}$$

where z is a uniformly distributed random variable and y a random variable distributed according to the desired distribution. Let us rewrite the integral of $P(y)$ as $I_P(y)$ then we find $z = I_P(y)$ and therefore

$$y = I_P^{-1}(z) \tag{1.12}$$

This shows that a transformation between the two distributions can be found only if

1. the integral $I_P(y) = \int_0^y P(y')dy'$ can be solved analytically in a closed form
2. there exists an analytic inverse of $z = I_P(y)$ such that $y = I_P^{-1}(z)$

Of course, these conditions can be overcome to a certain extent by precalculating/tabulating and inverting $I_P(y)$ numerically, if the integral is well-behaved (i.e. is non-singular). Then, with a little help from precalculated tables, it is possible to transform the uniform numbers numerically.

We are now going to demonstrate this method for the two most commonly used distributions: the Poisson distribution and the Gaussian distribution. We are already going to see in the case of the Gaussian distribution that quite a bit of work is required to create such a transformation.

The Poisson Distribution

The Poisson distribution is defined as

$$P(y) = ke^{-yk}. \tag{1.13}$$

By applying the area equality of eq. (1.11) we find

$$z = \int_0^y ke^{-y'k}dy' = \int_0^z P_u(z')dz' \tag{1.14}$$

thus

$$z = -e^{-y'k}|_0^y = 1 - e^{-yk}. \tag{1.15}$$

Solving for y yields

$$y = -\frac{1}{k}\ln(1-z). \tag{1.16}$$

The Gaussian Distribution

Analytical methods of generating normally distributed random number are very useful, since there are many applications and examples where such numbers are needed.

The Gaussian or normal distribution is written as

$$P(y) = \frac{1}{\sqrt{\pi\sigma}}e^{-\frac{y^2}{\sigma}} \tag{1.17}$$

Unfortunately, only the limit for $y \to \infty$ can be solved analytically:

$$\int_0^\infty \frac{1}{\sqrt{\pi\sigma}}e^{-\frac{y'^2}{\sigma}}dy' = \frac{\sqrt{\pi}}{2}. \tag{1.18}$$

However, Box and Muller[7] have introduced the following elegant trick to circumvent this restriction. Let us assume we take two (uncorrelated) uniform random variables z_1 and z_2. Of course we can apply the area equality of eq. (1.11) again but this time we write it as a product of the two random variables

$$z_1 \cdot z_2 = \int_0^{y_1} \frac{1}{\sqrt{\pi\sigma}}e^{-\frac{y_1'^2}{\sigma}}dy_1' \cdot \int_0^{y_2} \frac{1}{\sqrt{\pi\sigma}}e^{-\frac{y_2'^2}{\sigma}}dy_2' = \int_0^{y_2}\int_0^{y_1} \frac{1}{\pi\sigma}e^{-\frac{y_1'^2+y_2'^2}{\sigma}}dy_1'dy_2'. \tag{1.19}$$

[7] G. E. P. Box and Mervin E. Muller, A Note on the Generation of Random Normal Deviates, The Annals of Mathematical Statistics (1958), Vol. **29**, No. 2 pp. 610-611

This integral can now be solved by transforming the variables y_1' and y_2' into polar coordinates:

$$r^2 = y_1^2 + y_2^2 \tag{1.20}$$
$$\tan\phi = \frac{y_1}{y_2} \tag{1.21}$$

with

$$dy_1' dy_2' = r' dr' d\phi' \tag{1.22}$$

Substituting in eq. (1.19) leads to

$$z_1 \cdot z_2 = \frac{1}{\pi\sigma} \int_0^{\phi} \int_0^{r} e^{-\frac{r'^2}{\sigma}} r' dr' d\phi' \tag{1.23}$$
$$= \frac{\phi}{\pi\sigma} \int_0^{r} e^{-\frac{r'^2}{\sigma}} r' dr' \tag{1.24}$$
$$= \frac{\phi}{\pi\sigma} \cdot \frac{\sigma}{2} \left(1 - e^{-\frac{r^2}{\sigma}}\right) \tag{1.25}$$
$$z_1 \cdot z_2 = \underbrace{\frac{1}{2\pi} \arctan\left(\frac{y_1}{y_2}\right)}_{\equiv z_1} \cdot \underbrace{\left(1 - e^{-\frac{y_1^2+y_2^2}{\sigma}}\right)}_{\equiv z_2} \tag{1.26}$$

By separating these two terms (and associating them to z_1 and z_2, respectively) it is possible to invert the functions such that

$$y_1^2 + y_2^2 = -\sigma \ln(1 - z_2) \tag{1.27}$$
$$\frac{y_1}{y_2} = \tan(2\pi z_1) = \frac{\sin(2\pi z_1)}{\cos(2\pi z_1)} \tag{1.28}$$

Solving these two coupled equations finally yields

$$y_1 = \sqrt{-\sigma \ln(1 - z_2)} \sin(2\pi z_1) \tag{1.29}$$
$$y_2 = \sqrt{-\sigma \ln(1 - z_2)} \cos(2\pi z_1) \tag{1.30}$$

Thus, using two uniformly distributed random numbers z_1 and z_2, one obtains (through the Box-Muller transform) two normally distributed random numbers y_1 and y_2.

1.5.2 The Rejection Method

As we have seen in subsection 1.5.1, there are two conditions that have to be satisfied in order to apply the transformation method: integrability and invertability. If either of these conditions is not satisfied, there exists no analytical method to obtain random numbers in this distribution. It is important to note that this is particularly relevant for experimentally obtained data (or other sources), where no analytical description is available. In that case, one has to resort to a numerical method to obtain arbitrarily distributed random numbers, which is called the rejection method.

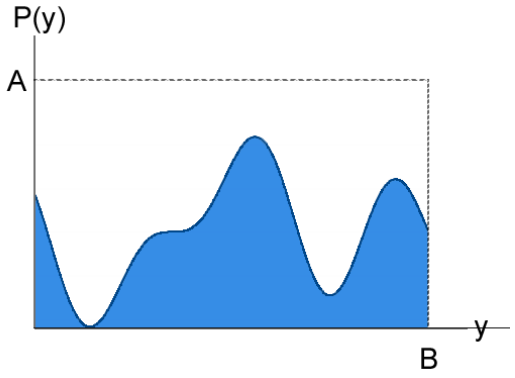


Figure 1.2: Illustration of the boundaries for the rejection method. Sample points are placed within the box and rejected if they lie above the curve, else accepted [20]

Let $P(y)$ be the distribution of which we would like to obtain random numbers. A necessary condition for the rejection method to work is that $P(y)$ is well-behaved, in this case "well-behaved" means that $P(y)$ is finite over the domain of interest $P(y) < A$ for $y \in [0, B]$, with $A, B \in \mathbb{R}$ and $A, B < \infty$. We then define an upper bound to be the box with edge length B and A. (see Fig. 1.2)

We now produce two pseudo-random variables z_1 and z_2 with $z_1, z_2 \in [0, 1[$. If we consider the point with coordinates $(Bz_1, Az_2)^T$, we see that it surely lies within the defined box. If the point lies above the curve $P(y)$, i.e. $Az_2 > P(Bz_1)$, the point is rejected (hence the name of the method). Otherwise $y = Bz_1$ is retained as a random number, which is distributed according to $P(y)$.

The method works in principle quite well, however certain issues have to be taken into consideration when using it.

- It is desirable to have a good guess for the upper bound. Obviously, the better the guess, the less points are rejected. In the above description of the algorithm we have assumed a rectangular box. This is however not a necessary conditions. The bound can be any distribution for which random numbers are easily generated.
- While the method is sound, in practice it is often faster to invert $P(y)$ numerically as mentioned already in subsection 1.5.1.
- There is a method to make the rejection method faster (but also more complicated): We use N boxes to cover $P(y)$ and define the individual box with

side length A_i and $b_i = B_{i+1} - B_i$ for $l \leq i \leq N$. Then, the approximation of $P(y)$ is much better (this is related to the idea of the Riemann-integral)

Literature

- Numerical Recipes
- D. E. Knuth: "The Art of Programming Vol. 2: Seminumerical Algorithms" (Addison-Wesley, Reading MA, 1997): Chapter 3.3.1
- J.E. Gentle, "Random number generation and Monte Carlo Methods", (Springer, Berlin, 2003).

Chapter 2

Percolation

Percolation in material science and chemistry describes the movement or filtering of fluids through porous media. The name stems originally from Latin and is still common in Italian[1].

A very simple and basic model of such a process was first introduced by Broadbent and Hammersley (Proc. Cambridge Phil. Soc. Vol. 53, p.629 (1957)).

While the original idea was to model the fluid motion through a porous material (e.g. a container filled with glass beads), it was found that the model had many other applications. Furthermore, it was observed that the model had some interesting universal features of so called critical phenomena[2].

Applications include

- General porous media: for example used in the oil industry and as a model for the pollution of soils
- Sol-Gel transitions
- "Mixtures" of conductors and insulators: find the point at which a conducting material becomes insulating
- Spreading of fires for example in forest
- Spreading of epidemics or computer virii
- Crash of stock markets (D. Sornette, Professor at ETH)
- Landslide election victories (S. Galam)
- Recognition of antigens by T-cells (Perelson)

[1] *ita*: percolare: 1 Passare attraverso. ~ filtrare. 2 Far filtrare. (1 pass through ~ filter, 2 make sth filter).

[2] Critical phenomena is the collective name associated with the physics of critical points.

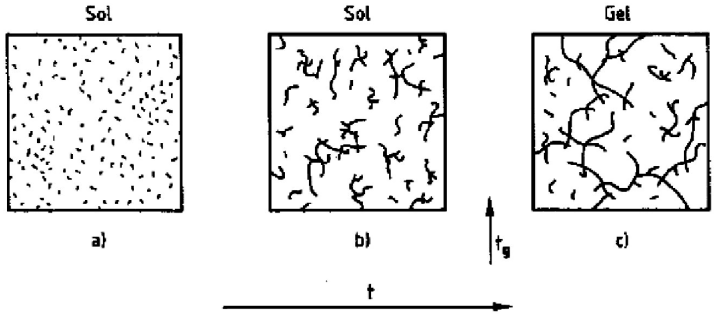


Figure 2.1: Sol-Gel transition: a) Monomers are dispersed in the liquid (sol). b) When the liquid is cooled down, the monomers start polymerizing and grow. c) Once a huge macromolecule (which spans through the whole container) has formed the Gel transition occurs. [2]

2.1 The Sol-Gel Transition

The formation of gelatine is quite astonishing from the chemical and physical point of view. Initially, gelatine is a fluid containing many small monomers (emulsion) which is referred to as sol. If we place the sol in a fridge, it becomes a "solid" gel. The process taking place is schematically illustrated in Fig. 2.1: Upon cooling, the monomers start polymerizing and the polymers start growing. At some point in time, one molecule has become sufficiently big to span from one side of the container to the other, which is when the so-called percolation transition occurs. The polymerization as well as the growth is experimentally accessible; one can for instance measure the shear modulus or the viscosity as a function of time. After a characteristic time ("gel time") t_G , the shear modulus suddenly increases from zero to a finite value. Similarly, the viscosity increases and becomes singular at t_G; this is reflected in experimental findings such as those in Fig. 2.2.

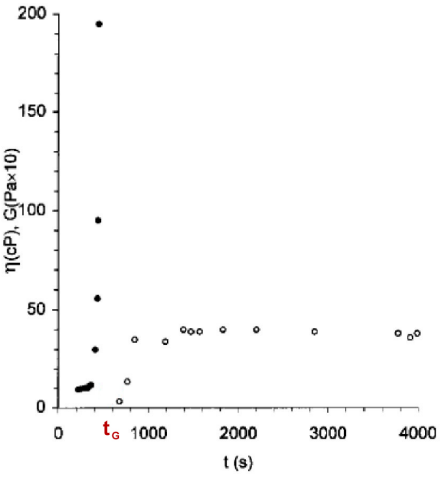
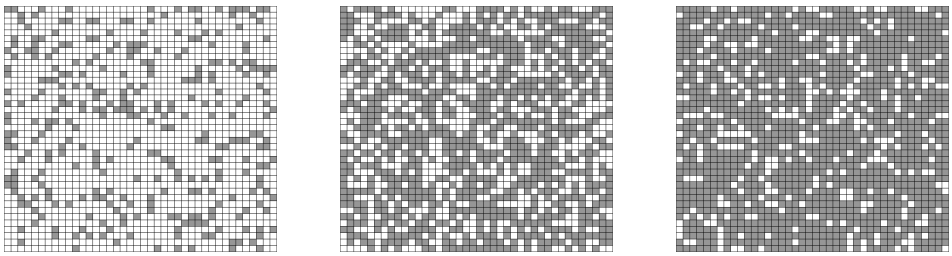


Figure 2.2: Viscosity (filled circles) and shear modulus (open circles) as a function of time. [2]

2.2 The Percolation Model

Modeling this process is surprisingly simple. Let us assume that we create a square lattice with side length L (e.g. $L = 16$) such that every cell can either be occupied or empty. The initial configuration is that all fields are empty. We fill each cell

with probability p, that is, for each cell we create a random number $z \in [0, 1[$ and compare it to p. If $z < p$, the cell will be marked as occupied otherwise the cell remains empty. This can be done for different values of p, leading to different results (see Fig 2.3). One immediately notices that for small values of p, most cells are empty, whereas for big values of p, most cells are occupied. There exists a critical probability $p = p_c = 0.592...$ when for the first time a fully connected cluster of cells spans two opposite sides of the box (this is also called a percolating cluster). While p_c is defined for infinite clusters, we can also find it for finite sizes; in that case, p_c is the average probability at which a percolating cluster first occurs. The critical probability is sometimes referred to as the percolation threshold.



(a) $p = 0.2$ (b) $p = 0.6$ (c) $p = 0.8$

Figure 2.3: Percolation on a square lattice with L=40 for different values of p. As the occupation probability p increases, more cells are filled. [20]

2.2.1 The Burning Method

While the previously noted observation of a critical probability was still rather general, it would be desirable to have a tool at one's disposal that could provide exact information about a configuration as in Fig. 2.3 and detect whether a spanning cluster exists.

The burning method provides us with exactly this information; it does not only provide us with a boolean feedback (yes/no) but even calculates the minimal path length (the minimal distance between opposite sides, following only occupied sites). The name of the method stems from its implementation; let us assume that we have a grid with occupied and unoccupied sites. An occupied site represents a tree while an unoccupied site stands for empty space. If we start a fire at the very top of our grid, all trees in the first row will start to light up as they are in the immediate vicinity of the fire. Obviously, not only the first row of trees will fall victim to this forest fire, as the neighbors of the trees in the first row will soon catch on fire as well. The second iteration step is thus that all trees that are neighbors of already burning trees are being torched.

Clearly, the iterative method only comes to an end when the fire finds no new victims (i.e. unburnt occupied sites neighboring a burning site) to devour and consequently dies out or if the fire has reached the bottom. If the inferno has reached the bottom, we can read off the length of the shortest path from the iteration counter since the number of iterations defines the minimum shortest path of the percolating cluster.

The algorithm is as follows:

1. Label all occupied cells in the top line with the marker t=2.
2. Iteration step t+1:
   (a) Go through all the cells and find the cells which have label t.
   (b) For each of the found t-label cells do
      i. Check if any direct neighbor (North, East, South, West) is occupied and not burning (label is 1).
      ii. Set the found neighbors to label t+1.
3. Repeat step 2 (with t=t+1) until either there are no neighbors to burn anymore or the bottom line has been reached - in the latter case the latest label minus 1 defines the shortest path.

A graphical representation of the burning algorithm is given in Fig. 2.4.

Figure 2.4: The burning method: a fire is started in the first line. At every iteration, the fire from a burning tree lights up occupied neighboring cells. The algorithm ends if all (neighboring) trees are burnt or if the burning method has reached the bottom line. [20]

2.2.2 The Percolation Threshold

When we carry out some simulations using the percolation model, we observe that the probability to obtain a spanning cluster depends on the occupation probability p (see Fig. 2.5). When we also vary the lattice size, we see that the transition from low wrapping probabilities to high wrapping probabilities becomes more abrupt with increasing lattice size. If we choose very large lattices the wrapping probability will start to resemble a step function. The occupation probability at which we observe the increasingly abrupt transition is called percolation threshold p_c.

The fact that for small lattices the transition is smeared out stems from the nature of the process which is based on randomness. When we carry out a sequence of simulations with increasing lattice sizes, we can determine the percolation threshold to be approximately $p_c = 0.59274...$.

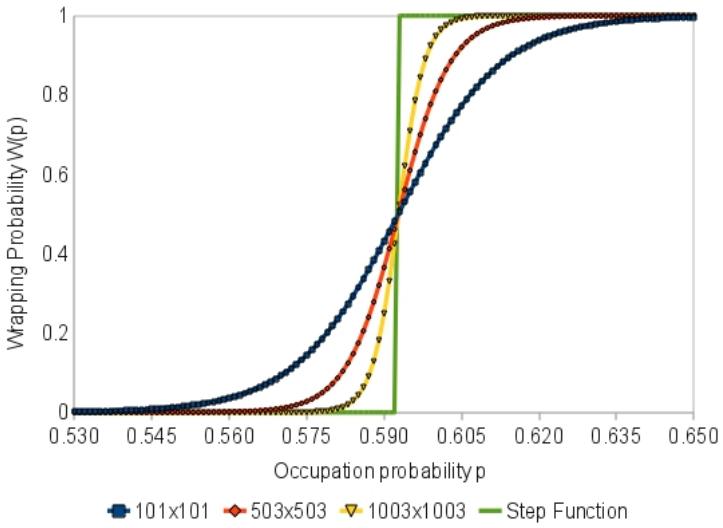


Figure 2.5: Wrapping probability as a function of the occupation probability for different lattice sizes and a step function for comparison. One observes that $p_c = 0.592746$ [3][21]

The percolation threshold is a characteristic value for a given type of lattice. One finds different percolation thresholds for different lattices (Fig. 2.6 shows values for some lattice types).

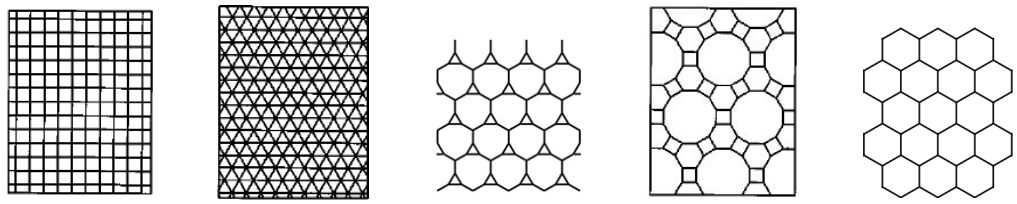
Percolation can be performed not only on lattice sites, but also on the corresponding bonds. Considering bonds changes the number of neighbors in a given lattice and therefore also the percolation threshold.

lattice	**site**	**bond**
cubic (body-centered)	0.246	0.1803
cubic (face-centered)	0.198	0.119
cubic (simple)	0.3116	0.2488
diamond	0.43	0.388
honeycomb	0.6962	0.65271
4-hypercubic	0.197	0.1601
5-hypercubic	0.141	0.1182
6-hypercubic	0.107	0.0942
7-hypercubic	0.089	0.0787
square	0.592746	0.50000*
triangular	0.50000*	0.34729*

Figure 2.6: Percolation threshold for different lattices for site percolation and bond percolation. Numbers with a star (*) can be calculated analytically. [10]

By looking at the values for different lattices in Fig. 2.6, it becomes clear that different geometries (e.g. those in Fig. 2.7) have significantly different thresholds. For example, the honeycomb lattice (Fig. 2.7e) has the highest 2D threshold. Furthermore, some lattices (such as the triangular lattice seen in Fig. 2.7b) can be calculated analytically (denoted by a *). Intuitively, it is clear that the threshold needs to depend on the geometry as, when we think of the burning algorithm, the speed at which the "fire" will spread depends on the geometry of the lattice.

Please note that if not otherwise stated we will only use "sites" for the percolation. This implies in general also the behavior for "bonds".



(a) Square lattice (b) Triangular lattice (c) Martini lattice (d) (4,6,12) lattice (e) Honeycomb lattice

Figure 2.7: Different lattices [1]

2.2.3 The Order Parameter

We have determined the percolation threshold in the previous section and stated that for infinitely large systems the transition indeed occurs exactly at p_c. Let us now consider probabilities $p > p_c$, where we will always find a spanning cluster (for sufficiently large systems). Naturally, one asks how many of the sites belong to the biggest spanning cluster. More precisely, we can define the fraction of sites which belong to the biggest cluster $P(p)$ as a function of the occupation probability p. We call this measure the order parameter (for its evolution with p, see Fig. 2.8). Obviously, for $p < p_c$ the order parameter is 0, as there are no spanning clusters. For $p \geq p_c$ the order parameter increases. When analyzing the behavior of the order parameter, one finds that $P(p)$ behaves like a power law in the region close to the percolation threshold

$$P(p) \propto (p_c - p)^{\beta}$$

with a dimension-dependent exponent β. Such a behavior is called "universal criticality". This concept will be discussed in much more detail in later chapters. The concept of universality of critical phenomena is a framework or theory which describes many different systems, which at first sight have very little in common; examples include phase transitions, magnetization and percolation.

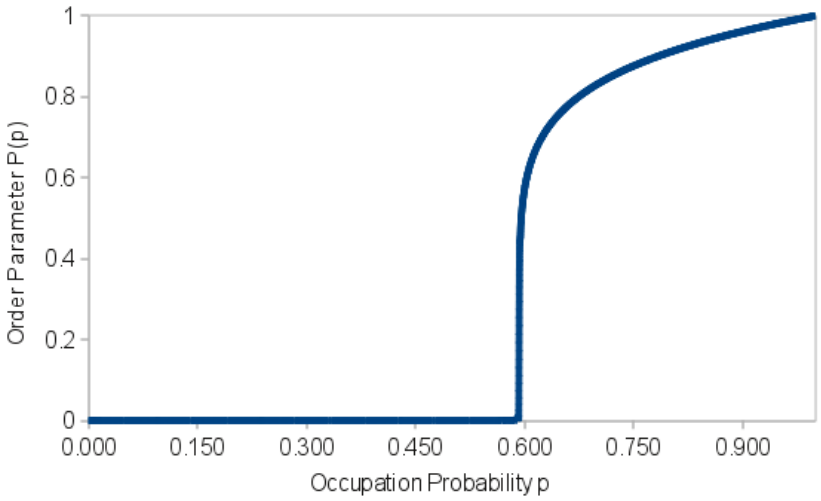


Figure 2.8: Order parameter $P(p)$ as a function of the occupation probability p. The system shows a "critical" behavior; Starting at p_c, the order parameter is $\propto (p - p_c)^{\beta}$, where $\beta = \frac{5}{36}$ in 2D and $\beta \approx 0.41$ in 3D. [20]

2.2.4 The Cluster Size Distribution

After having investigated the percolation threshold and the fraction of sites in the spanning cluster, the natural extension would be to know how the different

clusters are distributed. A tool similar to the burning algorithm is necessary to achieve such a task and to identify all the different clusters. In fact, there are several such algorithms; the most popular (and for this purpose most efficient) algorithm is the Hoshen-Kopelman algorithm, which was developed in 1976.

Let us write the lattice as a matrix N_{ij}, which can have values of 0 (site is unoccupied) and 1 (site is occupied) and let k be the running variable (we use k to label the clusters in N_{ij}). Additionally, an array M_k is introduced, the mass of cluster k, which counts the number of sites belonging to a given cluster k. We start the algorithm by setting $k = 2$ (since 0 and 1 are already taken) and searching for the first occupied site in N_{ij}. We then add this site to the array $M_{k=2} = 1$ and set the entry in N_{ij} to k (so it is branded as pertaining to the cluster k).

We then start looping over all lattice sites N_{ij} and try to detect whether an occupied site belongs to an already known cluster or a new one. We comb through the lattice from top-left to bottom-right; the criterion is rather simple:

- If a site is occupied and the top and left neighbors are empty, we have found a new cluster and we set k to $k + 1$, $N_{ij} = k$ and $M_k = 1$. We continue the loop.
- If one of the sites (top or left) has the value k_0 (i.e. has already been absorbed into a cluster), we increase the value of the corresponding array M_{k_0} by one (setting M_{k_0} to $M_{k_0} + 1$). We name the new site accordingly, i.e. $N_{ij} = k_0$.
- If both neighboring sites are occupied with k_1 and k_2 respectively (assuming $k_1 \neq k_2$)- meaning that they are already part of a cluster - we choose one of them (e.g. k_1). We set the matrix entry to the chosen value, $N_{ij} \leftarrow k_1$, and increase the array value not only by one but also by the whole number of sites already in the second cluster (in the example k_2), $M_{k_1} \leftarrow M_{k_1} + M_{k_2} + 1$. Of course we have to mark the second array M_{k_2} in some way so that we know that its cluster size has been transferred over to M_{k_1} which we do by setting it to $-k_1$. We have thus branded M_{k_2} in such a way that we immediately recognize that it does not serve as a counter anymore (as a cluster cannot consist of a negative number of sites). Furthermore, should we encounter an occupied site neighboring a k_2 site, we can have a look at M_{k_2} to see that we are actually dealing with cluster k_1 (revealing the "true" cluster number).

The last point is crucial, as we usually deal with a number of sites that are marked in such a way that we have to first recursively detect which cluster they pertain to before carrying out any further part of the algorithm. The recursive detection stops as soon as we have found a k_0 with $M_{k_0} \geq 0$ (i.e. a "true" cluster number).

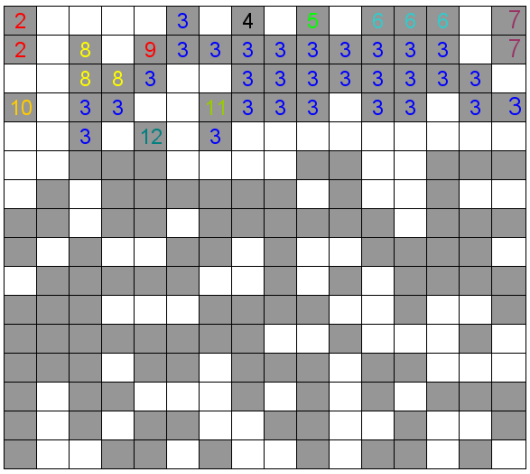


Figure 2.9: The Hoshen-Kopelman algorithm applied to a percolation cluster. The numbers denote the running cluster variable k, however without the negative links as described in the text. [20]

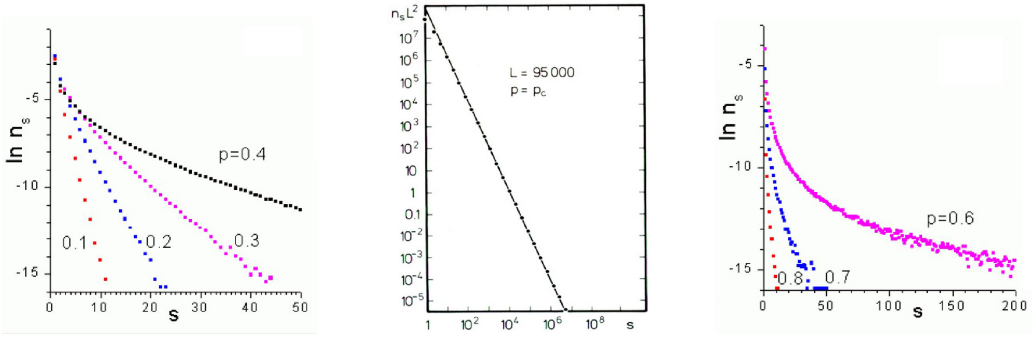
Once all the sites N_{ij} have been visited, the algorithm ends up with a number l of clusters (where $l < k_{max}$). The only thing left to do is to construct a histogram of the different cluster sizes. This is done by looping through all the clusters $k \leftarrow 2..k_{max}$ while skipping negative M_k.

Let us write down the Hoshen-Kopelman algorithm:

1. $k \leftarrow 2$, $M_k \leftarrow 1$
2. for all i, j of N_{ij}
   (a) if top and left are empty (or non-existent) $k \leftarrow k+1$, $N_{ij} \leftarrow k$, $M_k \leftarrow 1$
   (b) if one is occupied with k_0 then $N_{ij} \leftarrow k_0$, $M_{k_0} \leftarrow M_{k_0} + 1$
   (c) if both are occupied with $k_1$and k_2 (and $k_1 \neq k_2$) then choose one, e.g. k_1 and $N_{ij} \leftarrow k_1$, $M_{k_1} \leftarrow M_{k_1} + M_{k_2} + 1$, $M_{k_2} \leftarrow -k_1$
   (d) If both are occupied with k_1, $N_{ij} \leftarrow k_1$, $M_{k_1} \leftarrow M_{k_1} + 1$
   (e) if any of the k's considered has a negative mass M_k, find the original cluster they reference and use its cluster number and weight instead
3. for $k \leftarrow 2..k_{max}$ do
   (a) if $M_k > 0$ then $n(M_k) \leftarrow n(M_k) + 1$

A visualization of this algorithm is given in Fig. 2.9. The algorithm is very efficient since it scales linearly with the number of sites.

Once we have run the algorithm for a given lattice (or collection of lattices and taken the average) we can evaluate the results. We find different behaviors of the relative cluster size n_s (where s denotes the size of the clusters) depending on the occupation probability p. These results are illustrated in Fig. 2.10, where the first graph (Fig. 2.10a) represents the behavior for subcritical occupation probabilities ($p < p_c$), the second graph (Fig. 2.10b) shows the behavior for the critical occupation probability($p = p_c$) and the third graph (Fig. 2.10c) depicts the behavior for overcritical occupation probabilities($p > p_c$).The corresponding distributions are also noted.



(a) $n_{p<p_c}(s) \propto s^{-\theta} e^{as}$ (b) $n_{p_c}(s) = s^{-\tau}$ (c) $n_{p>p_c}(s) \propto e^{-bs^{1-1/d}}$

Figure 2.10: Cluster size distribution for $p < p_c$ (left), $p = p_c$ (center) and $p > p_c$ (right). [10]

One finds that in the subcritical regime ($p < p_c$), $n_p(s)$ obeys a power law multiplied with an exponential function, whereas in the overcritical region ($p > p_c$), we observe a distribution that resembles an exponential decay but with an argument that is stretched with a power $s^{1-1/d}$.

We are thus able to describe the subcritical and overcritical behavior of the cluster size distribution - the only remaining question is how we can go from one to the other, or, in other words, what the cluster size distribution at the critical occupation probability p_c is. As one can see from Fig. 2.10b, the function appears as a straight line in the logarithmic plot, implying an exponential behavior $n_{p_c}(s) = s^{-\tau}$. Of course we would still like to know the exponent τ; this exponent depends on the dimension of the problem but can be calculated (in 2D it is $\frac{187}{91}$ while in 3D we find 2.18). One can also calculate the bounds for τ: $2 \leq \tau \leq 5/2$.

We can summarize the behavior of the cluster size distribution n_s in the three different regions:

$$n_p(s) \propto \begin{cases} s^{-\theta} e^{as} & p < p_c \\ s^{-\tau} & p = p_c \\ e^{-bs^{1-1/d}} & p > p_c \end{cases} \tag{2.1}$$

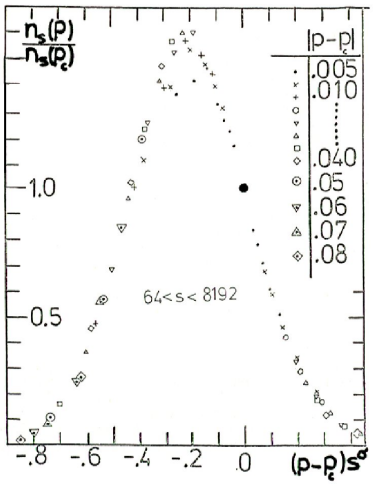


Figure 2.11: Scaling behavior of the percolation cluster size distribution [10]

In a next step, we can compare the non-critical distributions to the cluster size distribution for p_c; We plot a rescaled distribution

$$\tilde{n}_p(s) = n_p(s)/n_{p_c}(s)$$

which is defined on all three regions from (2.1). We may feel adventurous and want to plot $\tilde{n}_p(s)$ against $(p-p_c)s^\sigma$ as in Fig. 2.11. We immediately notice that we obtain a global curve which can be described by

$$n_p(s) = s^{-\tau}\Re_\pm[(p-p_c)s^\sigma] \tag{2.2}$$

with the scaling functions[3] $\Re_\pm$, where the subscript $\pm$ stands for $p > p_c$ (+) and $p < p_c$ (-), respectively.

When we calculate the second moment[4] of the distribution we find a power law:

$$\chi =< \sum_s{}' s^2 n(s) > \tag{2.3}$$

[3]Scaling function: A function $f(x, y)$ of two variables that can be expressed as a function of one variable $f(x')$.

[4]The n-th moment of a distribution $P(x)$ is defined as

$$\mu_n = \int x^n P(x)dx$$

therefore the 0-th moment of a normalized distribution is simply $\mu_0 = 1$ and the first moment is $\mu_1 = \int xP(x)dx = E(x)$ (the expectation value). Accordingly, μ_2 is the standard deviation of the distribution around the expectation value. As we are dealing with a discretely valued argument s, the integral becomes a sum.

The apostrophe (') indicates the exclusion of the largest cluster in the sum (as the largest cluster would make χ infinite at $p > p_c$). One finds

$$\chi \propto C_{\pm}|p - p_c|^{-\gamma} \tag{2.4}$$

with $\gamma = 43/18 \approx 2.39$ in 2D and $\gamma = 1.8$ in 3D. The second moment is a very strong indicator of p_c as we can see a very clear divergence around p_c. Of course we see a connection to the Ising model (which is explained further later on), where the magnetic susceptibility diverges near the critical temperature.

It can be shown that the scaling exponents we have seen so far are related by

$$\gamma = \frac{3 - \tau}{\sigma}$$

These exponents can be found for different lattice types (such as those in Fig. 2.7) and in different dimensions. The table shown in Fig. 2.12 gives an overview of the different exponents measured for a regular (square) lattice up to 6 dimensions.

Table 2. Percolation exponents for $d = 2, 3, 4, 5, 6 - \varepsilon$ and in the Bethe lattice together with the page number defining the exponent. Rational numbers give (presumably) exact results, whereas those with a decimal fraction are numerical estimates.

Exponent	$d=2$	$d=3$	$d=4$	$d=5$	$d=6-\varepsilon$	Bethe	Page
α	$-2/3$	$-0{\cdot}62$	$-0{\cdot}72$	$-0{\cdot}86$	$-1+\varepsilon/7$	-1	39
β	5/36	0·41	0·64	0·84	$1-\varepsilon/7$	1	37
γ	43/18	1·80	1·44	1·18	$1+\varepsilon/7$	1	37
ν	4/3	0·88	0·68	0·57	$\frac{1}{2}+5\varepsilon/84$	1/2	60
σ	36/91	0·45	0·48	0·49	$\frac{1}{2}+O(\varepsilon^2)$	1/2	35
τ	187/91	2·18	2·31	2·41	$\frac{5}{2}-3\varepsilon/14$	5/2	33
$D(p=p_c)$	91/48	2·53	3·06	3·54	$4-10\varepsilon/21$	4	10
$D(p<p_c)$	1·56	2	12/5	2·8	–	4	62
$D(p>p_c)$	2	3	4	5	–	4	62
$\zeta(p<p_c)$	1	1	1	1	–	1	56
$\zeta(p>p_c)$	1/2	2/3	3/4	4/5	–	1	56
$\theta(p<p_c)$	1	3/2	1·9	2·2	–	5/2	54
$\theta(p>p_c)$	5/4	$-1/9$	1/8	$-449/450$	–	5/2	54
$f_{\max}$	5·0	1·6	1·4	1·1		1	42
μ	1·30	2·0	2·4	2·7	$3-5\varepsilon/21$	3	91
s	1·30	0·73	0·4	$0{\cdot}1_5$		0	93
D_B	1·6	$1{\cdot}7_4$	1·9	2·0	$2+\varepsilon/21$	2	95
$D_{\min}(p=p_c)$	1·13	1·34	1·5	1·8	$2-\varepsilon/6$	2	97
$D_{\min}(p<p_c)$	1·17	1·36	1·5		–	2	98
$D_{\max}(p=p_c)$	1·4	1·6	1·7	1·9	$2-\varepsilon/42$	2	97

For the exponents at p_c, the Bethe lattice values are exact at $d \geqslant 6$. A dash means that 6 is not the upper critical dimension for the ε-expansion.

Figure 2.12: Critical exponents for 2 to 6 dimensions. [10]

2.2.5 Size Dependence of the Order Parameter

We are now going to consider the situation at the critical occupation probability p_c. The size of the largest cluster shall be denoted by s_∞ and the side length of the square lattice by L. When we plot s_∞ against L, we notice that there is a power law at work,

$$s_\infty \propto L^{d_f}$$

where the exponent d_f depends on the dimension of the problem. For a square lattice (2D) we find $d_f = \frac{91}{48}$ and for a three dimensional cube we find $d_f \approx 2.51$. We shall show later on that

$$d_f = d - \frac{\beta}{\nu}$$

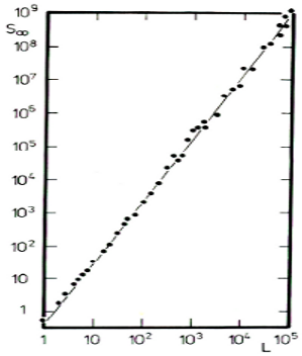


Figure 2.13: Size dependence of the order parameter [10]

2.2.6 The Shortest Path

In this chapter we have presented ways to calculate whether there exists a spanning cluster by means of the burning algorithm and we have determined and analyzed the cluster size distribution with the Hoshen-Kopelman algorithm. One unanswered question, however, is how the shortest path of a spanning cluster behaves as a function of the system size. Simulations yield

$$t^s \propto L^{d_{min}}$$

which is also a power law. The exponent d_{min} depends on the dimension:

$$d_{min} = \begin{cases} 1.13 & \text{in 2D} \\ 1.33 & \text{in 3D} \\ 1.61 & \text{in 4D} \end{cases}$$

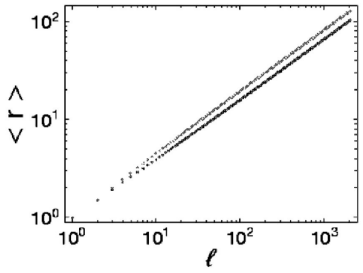


Figure 2.14: Shortest path t_s as a function of the system size for 4-dimensional site (upper) and bond (lower) percolation. [2]

Of course it is possible to determine d_{min} for site percolation and bond percolation as well as for different lattice types. As an example, Fig. 2.14 shows the site and bond percolation for a 4-dimensional system (square lattice), which was calculated by Ziff in 2001.

Literature

- D. Stauffer: „Introduction to Percolation Theory“ (Taylor and Francis, 1985).
- D. Stauffer and A. Aharony: „Introduction to Percolation Theory, Revised Second Edition“ (Taylor and Francis, 1992).
- M. Sahimi: „Applications of Percolation Theory“ (Taylor and Francis, 1994).
- G. Grimmett: „Percolation“ (Springer, 1989).
- B. Bollobas and O.Riordan: „Percolation“ (Cambridge Univ. Press, 2006).

Chapter 3

Fractals

3.1 Self-Similarity

The fractal dimension is a concept which has been introduced in the field of fractal geometry. The underlying idea is to find a measure to describe how well a given (fractal) object fills a certain space.

A related and simpler concept is self-similarity. Before introducing fractal dimensions, it may therefore be useful to consider some examples of self-similarity. In a nut-shell, one could define an object to be 'self-similar' if it is built up of smaller copies of itself. Such objects occur both in mathematics and in nature. Let us consider a few examples.

The Sierpinski-Triangle

Figure 3.1: The Sierpinski triangle - a self-similar mathematical object, which is created iteratively. [1]

The Sierpinski-triangle[1] is a mathematical object constructed by an iterative application of a simple operation. A triangle as in Fig 3.1 is subdivided into 4 sub-triangles and the center triangle is discarded, leaving a hole. In the next step of the iteration, each of the three remaining triangles is again subdivided and each central triangle is removed.

[1]Invented by the Polish mathematician Wacław Franciszek Sierpiński (1882-1969)

This obviously produces an object that is built up of elements that are almost the same as the complete object, which can be referred to as 'approximate' self-similarity. It is only in the limit of infinite iterations that the object becomes exactly self-similar: In this case the building blocks of the object are exactly the scaled object. One also says that the object becomes a fractal in the limit of infinite iterations.

Self-Similarity in Nature

Naturally occurring self-similar objects are usually only approximately self-similar. As an illustration of this point, consider a tree; a tree has different branches, and the whole tree looks similar to a branch connected to the tree trunk. The branch itself resembles a smaller branch attached to it and so on. Evidently, this breaks down after a few iterations, when the leaves of the tree are reached.

Figure 3.2: Barnsley fern[1]

Another example are gold colloids, which were shown to arrange in fractals of fractal dimension 1.70 (the meaning of this will soon be explained) by David Weitz in 1984. Colloidal gold is a suspension of sub-micrometer-sized gold particles in a fluid (for example water). These gold colloids arrange in fractals.

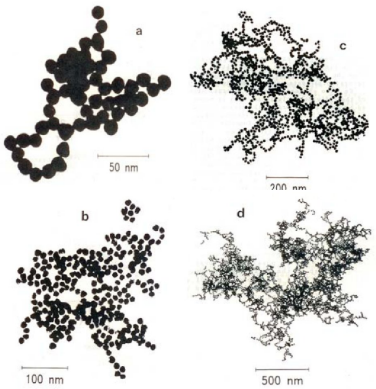


Figure 3.3: Gold Colloids at different scales. [2]

A rather beautiful example of a fractal is the fern which can be modelled using affine transformations (see Barnsley fern). The algorithm recursively produces a fern that resembles the natural fern very closely and illustrates self-similarity.

3.2 Fractal Dimension: Mathematical Definition

Keeping these examples in mind, it will be easier to put the mathematical definition of 'fractal dimension' into context. Note that we shall not try to give a mathematically rigorous definition but rather a good impression of what it would look like. To determine the fractal dimension of an object, one can use the following (theoretical) procedure:

Consider all coverings of the object with spheres of radius $r_i \leq \epsilon$, where ϵ is an arbitrary infinitesimal. Let $N_\epsilon(c)$ be the number of spheres used in the covering c. Then, the volume of the covering is

$$V_\epsilon(c) = \sum_{i=1}^{N_\epsilon(c)} r_i^d$$

where d is the dimension of the spheres (i.e. the dimension of the space into which the object is embedded).

We define V_ϵ^* as the volume of the covering that uses as few spheres as possible and has minimal volume

$$V_\epsilon^* = \min_{V_\epsilon(c)} \left(\min_{N_\epsilon(c)} \left(V_\epsilon(c) \right) \right)$$

The fractal dimension of the object can then be defined as

$$d_f := \lim_{\epsilon \to 0} \frac{\log\left(V_\epsilon^*/\epsilon^d\right)}{\log\left(L/\epsilon\right)} \tag{3.1}$$

Interpretation of 'Fractal Dimensions'

The definition of the fractal dimension (3.1) can be interpreted in the following way: When the length of the object is stretched by a factor of a, its volume (or 'mass') grows by a factor of a^{d_f}. We obtain this interpretation by rewriting equation 3.1 (in the limit $\epsilon \to 0$) as

$$\frac{V_\epsilon^*}{\epsilon^d} = \left(\frac{L}{\epsilon}\right)^{d_f}$$

Let us consider the effect of scaling L by a. V_ϵ^* will scale as claimed.

As a simple example consider the Sierpinski triangle as seen in Fig. 3.4. Stretching its sides by a factor of 2 evidently increases its volume by a factor of 3 ('volume' in two dimensions is usually referred to as 'area'). By inserting these values into equation (3.1) we find that the Sierpinski triangle has the fractal dimension $\log(3)/\log(2) \approx 1.585$.

3.3 The Box Counting Method

The box counting method is a method of numerically determining the fractal dimension of an object. It is conceptually easy, since it is close to the mathematical definition.

In the box counting method, we start with a picture of the fractal and superimpose a lattice with lattice constant ϵ. We define the number of boxes in the lattice that are not empty (contain a part of the fractal) as $N(\epsilon)$. We do this for a large range of ϵ and plot $N(\epsilon)$ vs. ϵ in a log-log plot. A typical result can be seen in Fig. 3.5.

We recognize a region where the slope is constant in the plot; it is in this region that the slope equals the fractal dimension of the object and the object is only self-similar in this region. Outside this self-similar regime the finite resolution and finite size of the picture disturb the self-similarity. As an illustration of this, it is useful to remember the previously mentioned interpretation of the definition of the fractal dimension; recall that ϵ is proportional to the length scale, while $N(\epsilon)$ is proportional to the volume of the fractal object.

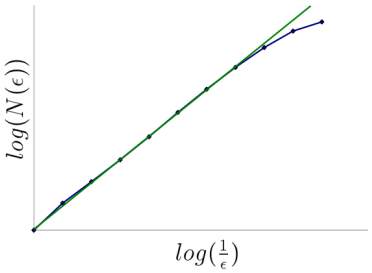


Figure 3.5: Plot of the data of a box count.[20]

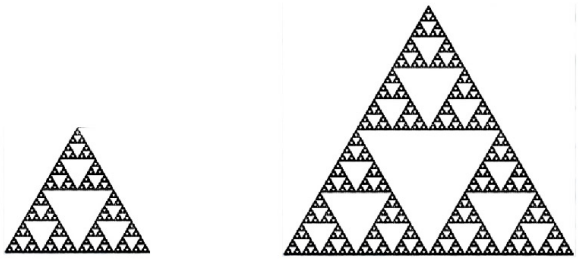


Figure 3.4: Stretching the length by a factor of 2, increases the volume by a factor of 3

Interlude: Multifractality

By slightly adapting the box counting method, another subtlety of the fractal dimension can be understood, the so-called 'multifractality'. Instead of only considering whether a box is completely empty or not, one also counts how many filled dots or pixels there are in a certain box i; we will denote this number by N_i. Additionally, we introduce the fraction $p_i = N_i/N$, where N is the total number of occupied points. Thus p_i is the fraction of dots contained in the box i.

So far we have not done anything new and original. Let us define the 'q-mass' M_q of an object as

$$M_q = \sum_i p_i^q$$

Furthermore, we shall introduce the associated fractal dimension d_q by the relation

$$M_q \propto L^{d_q}$$

where L is the length of the system.

By similar reasoning one can find

$$d_q = \frac{1}{1-q} \lim_{\epsilon \to 0} \left(\lim_{N \to \infty} \left(\frac{\log((M_q/N)^{(1/q)})}{\log(\epsilon)} \right) \right)$$

For some objects d_q is the same for all q and the same as the fractal dimension. For other objects, called 'strange attractors', one obtains different fractal dimensions for different values of q. This topic is of marginal interest in this course; the interested reader may find additional information by searching the keywords 'multifractality' and 'Renyi dimensions'.

3.4 The Sandbox Method

We will now return to the simpler definition of fractal dimension and leave aside further considerations of multifractality. The sandbox method is another easily implemented method of determining the fractal dimension of an approximately self-similar object numerically. An illustration of this method can be found in Fig. 3.6

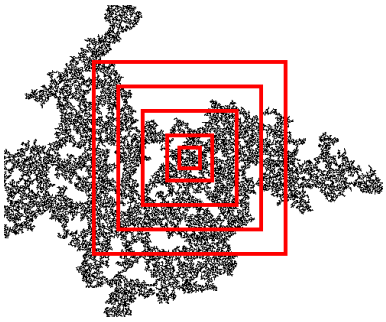


Figure 3.6: Illustration of the sandbox method: one measures the number of filled pixels in increasingly large centrally placed box.[17][21]

Again, we start with a picture of a (nearly) fractal object. We place a small box of size R in the center

of the picture and count the number of occupied sites (or pixels) in the box $N(R)$. We then successively increase the box size R in small steps until we cover the whole picture with our box, always storing $N(R)$. We finally plot $N(R)$ vs. R in a log-log plot where the fractal dimension is the slope (see e.g. Fig. 3.7).

3.5 The Correlation-Function Method

In the correlation function method, the correlation function of the point density of an object is considered. The correlation function is a measure for the amount of order in a system; it describes how microscopic variables are correlated over various distances. To give an intuitive idea of the meaning of the correlation function, one might say that a large value implies that the two quantities considered strongly influence each other, whereas zero would indicate that they are independent.

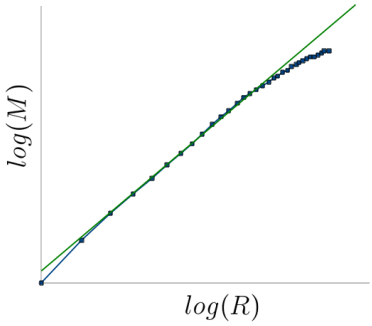


Figure 3.7: Plot of the data of a sandbox method algorithm; the fractal dimension is the slope in a log-log plot of the number of filled pixels in a box vs. the box radius

Let $\rho(x)$ be the density of the object at point x. The correlation function of the density at the origin and at a distance r is then

$$c(r) = \langle \rho(0) \cdot \rho(r) \rangle$$

The angled brackets denote a suitable averaging (for instance over all points at a distance r from randomly chosen origins). By measuring the correlation function $c(r)$ one can obtain the fractal dimension of the object,

$$c(r) \propto r^{d_f - d}$$

We see that in a double-logarithmic plot of $c(r)$ vs. r the slope of the (visually) linear part is the fractal dimension minus the dimension of the space the object is embedded into).

3.6 Correlation Length ξ

The correlation function $c(r)$ can also be written as

$$c(r) = \frac{\Gamma(\frac{d}{2})}{2\pi^{d/2} r^{d-1} \Delta r} [M(r + \Delta r) - M(r)]$$

We see that $c(r)$ essentially counts the number of filled sites (in 2D: pixels) within a band of size Δr at a distance r from the center and normalizes this expression with the surface area of the sphere in d dimensions at radius r. We apply this to clusters resulting from percolation (like the ones we considered in chapter 2). When we compute $c(r)$ for a given cluster (illustrated in Fig 3.8), we find that typically the correlation function decreases exponentially with the radius r (neglecting an offset C)

$$c(r) \propto C + exp\left(-\frac{r}{\xi}\right)$$

where the constant C vanishes in the sub-critical regime (i.e. $p < p_c$). The newly introduced quantity ξ is called the correlation length. It describes the typical length scale over which the correlation function of a given system decays. In the sub-critical regime, the correlation length ξ is proportional to the radius of a typical cluster.

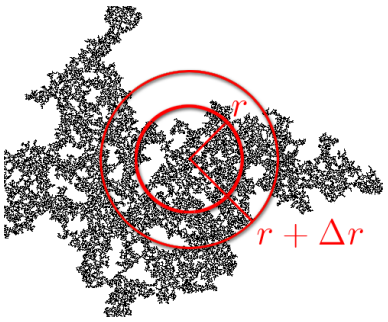


Figure 3.8: Illustration of the correlation function $c(r)$; at a given radius r we count the number of sites within a band of radius Δr [17][21]

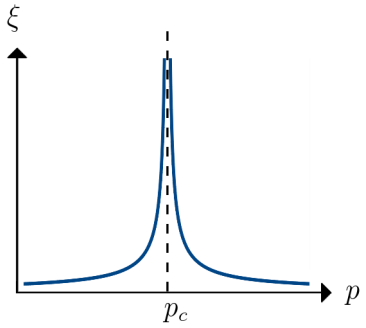


Figure 3.9: Singular behavior of ξ with respect to p [20]

When we analyze the dependence of the correlation length ξ on the occupation probability p (as in Fig. 3.9), we find a singularity at the critical occupation probability p_c,

$$\xi \propto |p - p_c|^{-\nu} \quad \text{where} \quad \nu = \begin{cases} \frac{4}{3} & \text{in 2 dimensions;} \\ 0.88 & \text{in 3 dimensions.} \end{cases}$$

We have thus established that the correlation length is singular at the critical occupation probability p_c. Apparently the assumption of exponential behavior is no longer valid at p_c. Measurements give a correlation function at p_c that behaves like

$$c(r) \propto r^{-(d-2+\eta)} \quad \text{where} \quad \begin{cases} \eta = \frac{5}{24} & \text{in 2 dimensions;} \\ \eta \approx -0.05 & \text{in 3 dimensions.} \end{cases}$$

We see that at p_c, the correlation functions decays like a power law with an exponent that contains a new parameter η which takes different values depending

on the dimension. With this background we can move on to the most important part of the chapter, the finite size effects.

3.7 Finite Size Effects

One encounters problems when the system size L is smaller than the correlation length ξ, particularly close to the critical point, where one observes a roundoff (see Fig. 3.10). One can observe this roundoff in the correlation length ξ, again plotted against the occupation probability p where one observes a maximum at p_c instead of the singularity that was mentioned above. Unsurprisingly, the correlation length gets cut at the size of the system.

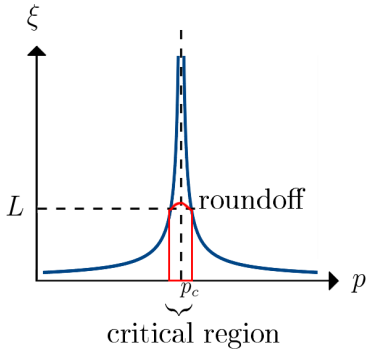


Figure 3.10: The finite size of the system leads to a cutoff in the correlation length [20]

To investigate this further, let us refer to the two points that define the critical region (the region where the cutoff occurs) as p_1 and p_2, the start and end points of the critical region in Fig. 3.11. We find

$$L = \xi(p_1) \propto (p_1 - p_c)^{-\nu}$$

and

$$p_1 - p_2 \approx 2(p_1 - p_c)$$

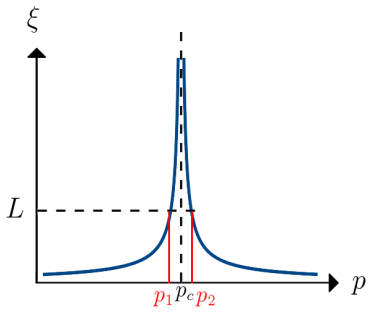


Figure 3.11: Position of p_1 and p_2 [20]

i.e. p_c comes to lie approximately in the center of the critical region, whose size is

$$p_1 - p_2 \propto L^{-\frac{1}{\nu}}$$

We can draw a first conclusion at this point: if we let $L \to \infty$, the critical region will vanish. This is obviously impossible with a finite computer. Thus L is finite and close to p_c we observe an approximation instead of the real values,

$$p_{\text{eff}}(L) = p_c(1 - aL^{-\frac{1}{\nu}})$$

The best we can do at this point is using the data acquired at finite sizes to guess the values for an infinite system; this process is called extrapolation which can be done for instance to find the critical occupation probability p_c.

3.7.1 Finite Size Scaling

When we consider the second moment χ of the cluster size distribution as a function of p and L, we observe that it can be reduced to a one variable function (remember that χ was introduced in chapter 2). This is due to the self-similarity of percolating clusters near the critical point.

If we plot χ against the occupation probability p for several values of L, we obtain at first plots that differ around the critical point (see e.g. 3.12 where this is illustrated for magnetic susceptibility). The most important difference is the size of the peaks, though even the overall "shape" of the functions seems different! How can we possibly find something these functions share?

Out of sheer curiosity, one might have a glimpse at the peak and try to find an expression for the size of the peak, depending only on the system size. Furthermore, one might introduce new parameters that are combinations of the previous ones and find that the plots start to look very much alike once the right expression is chosen. One possible combination of parameters is illustrated in Fig. 3.13 where we note that points that originally were far apart (as in Fig. 3.12) now come to lie on top of each other - we have found data collapse! Let us try to comprehend what this data collapse means. We have a function χ that was originally a function of two parameters (the occupation probability p and the system size L) and that now behaves as though it was a one-parameter function.

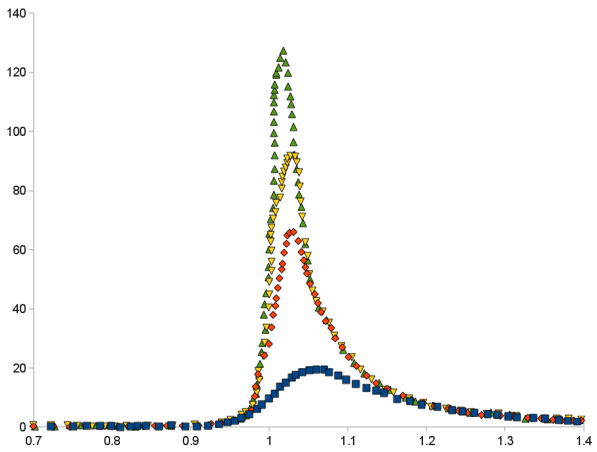


Figure 3.12: Illustration of the system size dependence of χ. Shown here: Magnetic susceptibility vs. temperature which shows a similar behavior. Note that the peak size increases with the system size. Sizes used: L=20 (blue), L=40 (red), L=50 (yellow), L=60 (green)[20][12]

We can express χ as follows:

$$\chi(p, L) = L^{\left(\frac{\gamma}{\nu}\right)} \aleph_\chi \left[(p - p_c) L^{\frac{1}{\nu}} \right]$$

where $\aleph$ is the so-called scaling function. When we go to the critical occupation probability p_c, the scaling function $\aleph$ approaches a constant and we find that the peak $\chi_{\max}$ depends on the system size,

$$\chi_{\max}(L) \propto L^{\frac{\gamma}{\nu}}$$

which could be verified by plotting the maxima of Fig. 3.12). These expressions are reminiscient of those previously introduced in the context of percolation and the idea is the same (e.g. compare (3.7.1) to (2.2)). We recall from chapter two that a scaling function is a function of two variables (in this context p, L) that can be expressed in terms of only *one* variable.

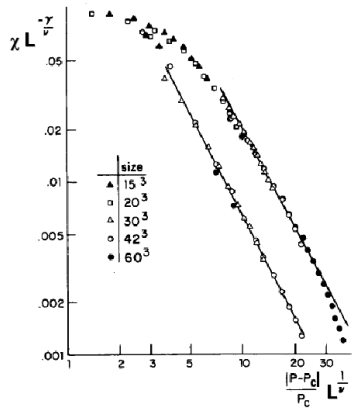


Figure 3.13: Finite size scaling of χ; one observes data collapse when plotting $\chi L^{-\gamma/\nu}$ against $\frac{|p-p_c|}{p_c} L^{1/\nu}$. The straight lines have a slope of $-\gamma$ (as the critical behavior of χ for infinite systems, (2.4)) . [11]

3.7.2 Size Dependence of the Order Parameter

Let us consider the fraction of sites in the spanning cluster at the critical occupation probability p_c. This too was mentioned already earlier on in the chapter on percolation (in section 2.2.5). We have seen that at p_c, we have

$$PL^d = s_\infty \propto L^{d_f}$$

where

$$d_f \begin{cases} = \frac{91}{48} & \text{in 2 dimensions;} \\ \approx 2.51 & \text{in 3 dimensions.} \end{cases}$$

and

$$d_f = d - \frac{\beta}{\nu}$$

In section 2.2.5, we simply introduced d_f without a word about its origins. When we consider fractal dimensions however, d_f will come out of the equations!

3.8 Fractal Dimension in Percolation

The fraction of sites P in the spanning cluster ("order parameter") is

$$P \propto (p - p_c)^{\beta}$$

Futhermore, P (when considered as a function of L and p) can be written as a function of a single parameter near p_c. This is referred to as finite scaling:

$$P(p, L) = L^{-\frac{\beta}{\nu}} \aleph_P \left[(p - p_c) L^{\frac{1}{\nu}} \right]$$

Moreover, at p_c, we find that not only the order parameter is system size dependent,

$$P \propto L^{-\frac{\beta}{\nu}}$$

but that the number of sites is also system size dependent

$$M \propto L^{d_f}$$

When we combine all of this, we obtain

$$M \propto PL^d \propto L^{\left(-\frac{\beta}{\nu}+d\right)} \propto L^{d_f}$$

and we have thus found the fractal dimension

$$d_f = d - \frac{\beta}{\nu}$$

3.9 Examples

3.9.1 Volatile Fractals

In a volatile fractal, the cluster is "redefined" at each scale, e.g. the path that spans $L_1 < L_2$ is not necessarily part of the path spanning L_2 as can be seen in Fig. 3.14.

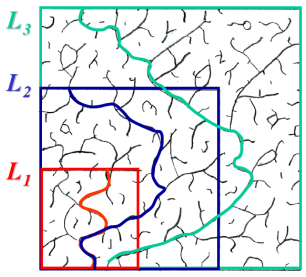


Figure 3.14: Example of a volatile fractal[2]

3.9.2 Bootstrap Percolation

The canonical bootstrap model is one where sites are initially occupied randomly (with a given probability p) as in the percolation model. Then, any sites that do not have at least m neighbors are removed. In the case of $m = 1$, all isolated sites are removed, in the case of $m = 2$ all "dangling" sites are removed. This does

however not affect p_c. If we choose $m \geq 3$, p_c and the nature of the transition are strongly dependent on the lattice; for sufficiently large m a first order transition is observed.

The bootstrap percolation method has many applications, including some systems in solid state physics, fluid flow in porous media and others.

An example of bootstrap percolation is given in Fig. 3.15

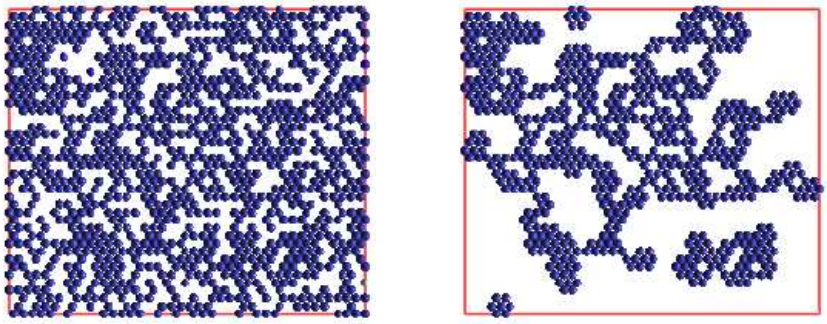


Figure 3.15: freshly occupied lattice (left) and the same lattice (right) after removing all sites with less than m neighbors ("culling") [18]

3.10 Cellular Automata

A cellular automaton is a model that consists of a regular grid of cells, each one of a finite number of states (e.g. ± 1). The grid can be in any finite number of dimensions. The model is discrete (as are its deterministic dynamics), i.e. we are dealing with boolean variables on a lattice from t to $t+1$, that is to say that all three major components are discrete:

- The variables have discrete values (boolean, i.e. 0 and 1)
- The sites are discrete (we are considering a lattice, not a continuous volume)
- We use finite time steps, and compute the next step $t+1$ based on the last time step t.

The binary variable σ_i at a given site i at the next time step $t+1$ is determined by

$$\sigma_i(t+1) = f_i(\sigma_1(t), ..., \sigma_k(t)) \tag{3.2}$$

where k is the number of inputs. We see that there are thus 2^{2^k} possible rules!

3.10.1 Classification of Cellular Automata

Let us consider $k = 3$ (3 inputs) for starters. There are $2^3 = 8$ possible binary entries $(111, 110, 101, 100, 011, 010, 001, 000)$ and the rule needs to be defined for each of these elements. Let us consider the following rule:

entries:	111	110	101	100	011	010	001	000
$f(n)$:	0	1	1	0	0	1	0	1

Furthermore, we define

$$c = \sum_{n=0}^{2^k-1} 2^n f(n)$$

which for the presented rule is

$$c = 0 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 64 + 32 + 4 + 1 = 101$$

We can thus identify a rule by its c number. Let us consider a couple of rules to see how this works (rule 4,8,20,28 and 90):

entries:	111	110	101	100	011	010	001	000
$f_4(n)$:	0	0	0	0	0	1	0	0
$f_8(n)$:	0	0	0	0	1	0	0	0
$f_{20}(n)$:	0	0	0	1	0	1	0	0
$f_{28}(n)$:	0	0	0	1	1	1	0	0
$f_{90}(n)$:	0	1	0	1	1	0	1	0

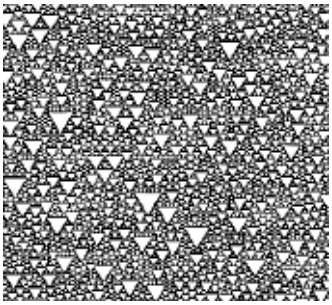
3.10.2 Time Evolution

We can study the evolution with time of a given rule. Given the very different character of the rules, the evolution patterns differ significantly; in fact, in some cases special patterns start to appear while in other cases, the area goes blank! Two examples are illustrated in Fig. 3.16.

3.10.2.1 Classes of Automata

Wolfram divided the rules into four groups according to their evolution with time:

- Class 1: Almost all initial patterns evolve quickly into a stable, homogeneous state, any randomness in the initial pattern disappears.
- Class 2: Almost all initial patterns evolve quickly into stable or oscillating structures. Some of the randomness in the initial pattern may be filtered out, but some remains. Local changes to the initial pattern tend to remain local.



(a) Evolution according to CA rule 126 [19]

(b) Evolution of a class 4 CA [19]

Figure 3.16: Comparison of the evolution for two completely different rules

- Class 3: Nearly all initial patterns evolve in a pseudo-random or chaotic manner. Any stable structures that appear are quickly destroyed by the surrounding noise. Local changes to the initial pattern tend to spread indefinitely.
- Class 4: Nearly all initial patterns evolve into structures that interact in complex and interesting ways. Eventually a class 4 may become a class 2 but the time necessary to reach that point is very large.

Some persistent structures are found in Fig. 3.17

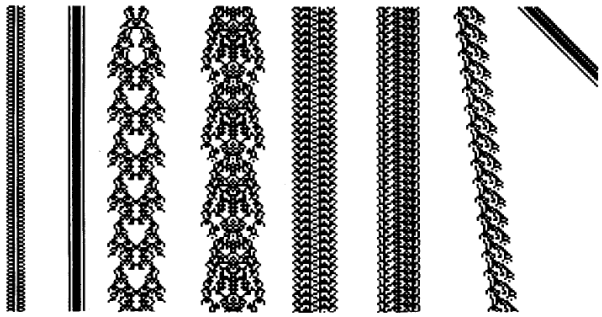


Figure 3.17: Examples of persistent structures [19]

3.10.2.2 The Game of Life

Let us consider a square lattice, and let n be the number of nearest and next-nearest neighbors that are 1. We shall then use the following rule:

- if n<2 : 0
- if n=2 : stay as before
- if n=3 : 1
- if n>3 : 0

An illustration of the resulting animation can be found here (Wikipedia, Game of Life). The animation keeps producing "shots" from a "gun" (see Fig. 3.18 and online on Wikipedia)

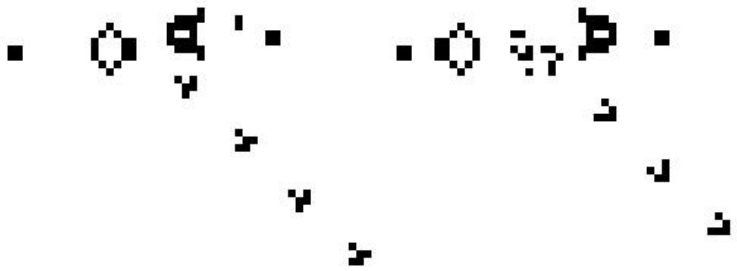


Figure 3.18: Steps 0 and 16 (in a period of 30 time steps) of the Game of Life [1]

Chapter 4

Monte Carlo Methods

4.1 What is "Monte Carlo" ?

The Monte Carlo methods can be characterized by their reliance on repeated random sampling and averaging to obtain results. One of their major advantages is their systematic improvement with the number of samples N, as the error Δ decreases as follows:

$$\Delta \propto \frac{1}{\sqrt{N}}$$

A good example of this is the computation of π which we shall consider in a minute. Monte Carlo methods appear frequently in the context of simulations of physical and mathematical systems. They are also very popular when an exact solution to a given problem cannot be found with a deterministic algorithm. They are particularly popular in the context of higher dimensional integration (see section 4.4).

4.2 Applications of Monte Carlo

Monte Carlo methods have a broad spectrum of applications, including the following:

- Physics: They are used in many areas in physics, some applications include statistical physics (e.g. Monte Carlo molecular modeling) or in Quantum Chromodynamics. In the much talked about Large Hadron Collider (LHC) at CERN, Monte Carlo methods were used to simulate signals of Higgs particles, and they are used for designing detectors (and to help understand as well as predict their behavior)

- Design: Monte Carlo methods have also penetrated areas that might - at first sight - seem surprising. They help in solving coupled integro-differential

equations of radiation fields and energy transport, which is essential for global illumination in photorealistic images of 3D models.

- Economics: Monte Carlo models have also found their place in economics where they have been used e.g. for financial derivatives.

Let us consider an example that we can directly implement ourselves to become more familiar with this new method.

4.2.1 Computation of π

A good illustration of this method of repeated random sampling and averaging is the computation of the number π.

The basic idea is rather simple: We consider the unit area ($x \in [0,1]$ and $y \in [0,1]$) and compare the area within the quarter circle, $P(x,y)$, to the area of the unit square (see Fig. 4.1). This will give $\frac{\pi}{4}$.

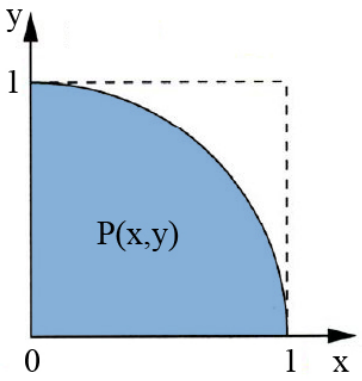


Figure 4.1: Illustration of the areas considered in the computation of π.[20]

This relation is mathematically exact and can be expressed as an integral in the following way:

$$\pi = 4 \int_0^1 \sqrt{1 - x^2} dx$$

Another way to compute π though (and learn something about Monte Carlo at the same time) goes as follows: We consider N random points in the unit square that are characterized by their x and y coordinates x_i, y_i. Then, the number of points N_c lying within the quarter circle (i.e. fulfilling the relation $x^2 + y^2 \leq 1$) is compared to the total number N of points and the fraction will give us an approximate value of π:

$$\pi(N) = 4 \frac{N_c(N)}{N}$$

Of course, the more points we consider, the better the approximation will become. In fact, the error is $\Delta = \pi(N) - \pi \propto \frac{1}{\sqrt{N}}$ and thus really does decrease with the number N of points. This is easy to understand. Imagine you chose just two points, one of which is lying inside the quarter circle, the other outside. This will give you $\pi(2) = 2$, a rather atrocious approximation to 3.14159265... but if you pick 10 points, they will start to approximate the real value much better. For

$N = 10,000$ for instance you may find that $N_c = 7,854$ giving $\pi(10,000) = 3.1416$. It is however not obvious from the start that the error should decrease $\propto \frac{1}{\sqrt{N}}$ which is why we shall look into this in a minute.

4.3 Computation of Integrals

Another well-known application of Monte Carlo is the computation of integrals, particularly higher dimensional integrals (we shall see later on that Monte Carlo is in fact the most efficient way to compute higher dimensional integrals).

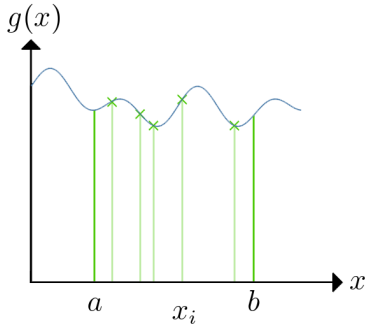


Figure 4.2: Simple sampling for a smooth function [20]

Let us consider the integral of a function $g(x)$ in an interval given by $[a, b]$. We may approximate the integral by choosing N points x_i on the x-axis with their corresponding values $g(x_i)$, summing and averaging over these sampled results and multiplying the resulting expression with the length of the interval:

$$\int_a^b g(x)dx \approx (b-a)\left[\frac{1}{N}\sum_{i=1}^{N} g(x_i)\right]$$

We now need to say a word or two about the nature of these randomly chosen points x_i on the x-axis. If we choose them completely at random, the process is called "**simple sampling**" which works very well if $g(x)$ is smooth.

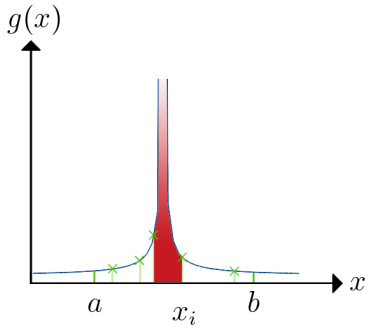


Figure 4.3: Sampling for a function with singularity [20]

But what if we cannot make the assumption that $g(x)$ is smooth? Let us consider a less "cooperative" function, for instance one featuring a singularity at a certain value. Due to the limited number of points that we would usually choose around the singularity (in simple sampling), we would not be able to really appreciate and understand its behavior. Furthermore, the integral would be a very rough approximation. We thus need more precision which goes beyond the possibilities of simple sampling – this additional precision is provided by a second function $p(x)$ which makes our condition for successful sampling less strict: only $\frac{g(x)}{p(x)}$ needs to be smooth. The sampling points are now distributed according to

$p(x)$ and we have

$$\int_a^b g(x)dx = \int_a^b \frac{g(x)}{p(x)}p(x)dx \approx (b-a)\left[\frac{1}{N}\sum_{i=1}^{N}\frac{g(x_i)}{p(x_i)}\right]$$

We have changed our way of sampling by using the distribution function $p(x)$, reducing the requirement on $g(x)$. This manifests itself in the summand above. One could state that $p(x)$ helps us to pick our sampling points according to their importance (e.g. select more points close to a singularity) - in fact, this kind of sampling is called **importance sampling**.

4.3.1 Integration Errors

We have been insisting that these are approximations and not analytical solutions. Naturally, the question arises what error one has to expect in the course of such an approximation.

Error in Conventional Methods

Let us first consider the error in conventional methods (you might have already seen this in a different class). We are going to use the trapezium rule.

Consider the Taylor Series expansion integrated from x_0 to $x_0 + \Delta x$:

$$\begin{aligned}\int_{x_0}^{x_0+\Delta x} f(x)dx &= f(x_0)\Delta x + \frac{1}{2}f'(x_0)\Delta x^2 + \frac{1}{6}f''(x_0)\Delta x^3 + ... \\ &= \left[\frac{1}{2}f(x_0) + \frac{1}{2}(f(x_0) + f'(x_0)\Delta x + \frac{1}{2}f''(x_0)\Delta x + ...) + ...\right]\Delta x \\ &= \frac{1}{2}(f(x_0) + f(x_0+\Delta x))\Delta x + O(\Delta x^3)\end{aligned}$$

This approximation, represented by $\frac{1}{2}[f(x_0) + f(x_0 + \Delta x)]\Delta x$, is called the *Trapezium Rule* based on its geometric interpretation (see Fig. 4.4) We can now see that the error is $\propto (\Delta x)^3$. Suppose we take Δx to be one third its previous value, then the error will decrease by a factor of 27! At the same time though the size of the domain would also be divided by this factor. The net factor is thus only 9 (i.e. $(\Delta x)^2$) and not 27 (i.e. $(\Delta x)^3$) as originally conjectured.

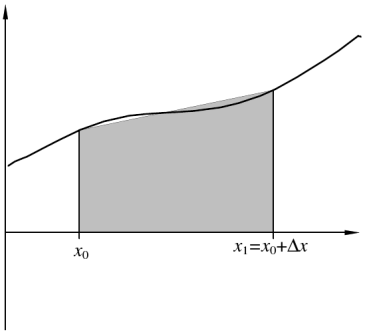


Figure 4.4: Graphical interpretation of the trapezium rule [13][21]

We now subdivide our interval $[x_0, x_1]$ into N subintervals of size $\Delta x = \frac{x_1 - x_0}{N}$. The Compound Trapezium Rule approximation to the integral is therefore

$$\begin{aligned}\int_{x_0}^{x_1} f(x)dx &\approx \frac{\Delta x}{2}\sum_{j=0}^{N-1} f(x_0 + j\Delta x) + f(x_0 + (j+1)\Delta x) \\ &= \frac{\Delta x}{2}[f(x_0) + 2f(x_0 + \Delta x) + 2f(x_0 + 2\Delta x) + ... \\ &\quad +2f(x_0 + (N-1)\Delta x) + f(x_1)]\end{aligned}$$

We thus see that while the error for each step is $O((\Delta x)^3)$, the cumulative error is N times this or $O((\Delta x)^2) \propto O(N^{-2})$.

The generalization to 2 dimensions is rather straightforward (the dimension refers to the domain[1]). Instead of each interval contributing an error of $(\Delta x)^3$ we now have a two dimensional "interval" and instead of the error being $\propto N(\Delta x)^3 \propto (\Delta x)^2$ (since $N \propto \frac{1}{\Delta x}$), we now have for each segment an error $\propto (\Delta x)^4$. The cumulative error is then $\propto N(\Delta x)^4 \propto (\Delta x)^2$ again as for a two dimensional domain we have $N \propto \frac{1}{(\Delta x)^2}$.

We can now do the same again for d dimensions (i.e. the dimension of the domain is d-dimensional). The "intervals" will now be d dimensional, and the error for each segment will be $(\Delta x)^{d+2}$ with the sum over all "subintervals" giving us $N(\Delta x)^{d+2}$ but $N \propto \frac{1}{(\Delta x)^d}$ so that the error will be $\propto N(\Delta x)^{d+2} \propto (\Delta x)^2$. The error is thus independent of the dimension.

We can conclude that the error in conventional methods goes as $(\Delta x)^2$ and given that $T \propto N \propto \frac{1}{(\Delta x)^d}$, we see that $\Delta x \propto T^{-\frac{1}{d}}$ and the error is $\propto (\Delta x)^2 \propto T^{-\frac{2}{d}}$.

Monte Carlo Error

Let us first consider a simplified case of a one dimensional function of one variable, $g : [a, b] \to \mathbb{R}$. If we pick N equidistant points in the interval $[a, b]$ we have a distance of $h = \frac{b-a}{N}$ between each of these points.
The estimate for the integral is

$$I = \int_a^b g(x)dx \approx \frac{b-a}{N}\sum_{i=1}^{N} g(x_i) = (b-a) < g > \equiv Q$$

[1] Quick recapitulation: A function $f : X \to Y$ takes an argument x from the *domain* X and assigns it a value $y = f(x)$ in the *range* Y of the function.

where $< g >$ stands for the sample mean of the integrand. The variance of the function can then be estimated using

$$var(g) \equiv \sigma^2 = \frac{1}{N-1}\sum_{i=1}^{N}(g(x_i) - < g >)^2$$

where the denominator is $N-1$ as we want to obtain the unbiased estimate of the variance. Using the central limit theorem, the variance of the estimate of the integral can be estimated as

$$var(Q) = (b-a)^2\frac{var(g)}{N} = (b-a)^2\frac{\sigma^2}{N}$$

which for large N decreases like $\frac{1}{N}$. Thus, the error estimate is

$$\delta Q \approx \sqrt{var(Q)} = (b-a)\frac{\sigma}{\sqrt{N}}$$

We can now generalize this to multidimensional integrals:

$$I = \int_{a_1}^{b_1} dx_1 \int_{a_2}^{b_2} dx_2 ... \int_{a_n}^{b_n} dx_n f(x_1, ..., x_n)$$

The previous interval $[a, b]$ becomes a hypercube V as integration volume, with

$$V = \{x : a_1 \leq x_1 \leq b_1, ...a_n \leq x_n \leq b_n\}$$

so of course the formulae are adapted accordingly.
Instead of the interval $[a, b]$ appearing in the variance, we now use the hypercube V:

$$var(Q) = V^2\frac{var(g)}{N} = V^2\frac{\sigma^2}{N}$$

and the error estimate becomes

$$\delta Q \approx \sqrt{var(Q)} = V\frac{\sigma}{\sqrt{N}}$$

but as we can see from this, the proportionality to $\frac{1}{\sqrt{N}}$ still remains.

Comparison of the Errors

We have seen that in conventional methods, the error of computing integrals goes with $T^{-\frac{2}{d}}$ and thus depends on the dimension, while the error in Monte Carlo methods is independent of the dimension. There is a crucial point at which Monte Carlo methods become more efficient,

$$T^{-\frac{2}{d}} \overset{crit}{=} \frac{1}{\sqrt{T}} \qquad \rightarrow d_{crit} = 4 \tag{4.1}$$

We can thus conclude that for $d > 4$, Monte Carlo becomes more efficient and is therefore used in areas where such higher dimensional integrals with $d > 4$ are commonplace.

4.4 Higher Dimensional Integrals

Let us now look at an example of higher dimensional integration: Consider N hard spheres of radius R in a 3D box of volume V. Our points are characterized by their position vector $\vec{x}_i = (x_i, y_i, z_i), 1 \leq i \leq N$. We define the distance between two such points as

$$r_{ij} := \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2}$$

where we have to take into consideration that the spheres are hard, i.e. cannot overlap. Thus the minimal distance between two neighboring spheres is the distance when the spheres are in contact, which is equal to $2R$. This translates to the following condition on r_{ij} :

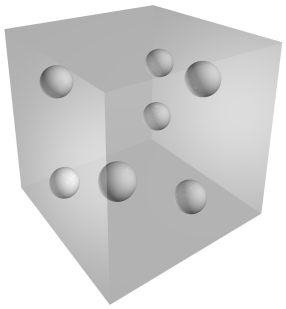


Figure 4.5: A cube filled with spheres [20]

$$r_{ij} \overset{!}{>} 2R$$

So far this does not seem to have too much to do with integration, we are simply placing some spheres in a volume V and measuring some distance r_{ij}. Let us say we are interested in the average distance between the centers of spheres in the box then we need to consider the following integral to reach an analytical result:

$$< r_{ij} >= \frac{1}{Z} \int \frac{2}{N(N-1)} \sum_{i<j} r_{ij} d^3r_1 ... d^3r_N \quad , \text{ where } \quad Z = \int d^3r_1 ... d^3r_N$$

Let us stay with this formula for a moment and have a look at the different factors. The first factor, $\frac{1}{Z}$, is a normalization factor. The following factor is of combinatory origin, in fact it stems from random drawing from a finite population without replacement. We start out with N choices (the whole population) and pick one sphere at random. There are only $(N-1)$ choices left for the second one (as it cannot be the first one again). This gives a total number of $N \cdot (N-1)$ possible combinations for picking two spheres. As they are indistinguishable, there is an additional factor of $\frac{1}{2}$ and we thus divide by this whole combinatory expression. Note that we would need to correct with a factor of $\frac{1}{2}$ if the sum were over $i \neq j$ (since, in that case, we would have counted each volume twice) which was avoided by simply summing over $i < j$.

The Monte Carlo approach to this formula is relatively simple:

- Choose a particle position (i.e. the center of the new sphere)
- Make sure that the new sphere does not overlap with any pre-existing spheres (see condition on r_{ij}). If it does overlap, reject the position and try again

- Once all the spheres have been placed, calculate the distances r_{ij}.

We then use these distances r_{ij} to compute the average. This procedure will converge and give a value that approximates the integral under the right conditions. It is important to note that our result will of course depend on the number of spheres positioned:
Imagine we had taken a ridiculously small number of spheres (let us say only two!) in a rather large volume and we carried out the distance measurement. There would not be much to average over and on top of that, due to the relative liberty of positioning the two spheres in the volume, the distance (and with it the average distance) would fluctuate wildly depending on the setup. If we repeat this n times and average over it, the result will thus not converge as nicely as it would with a relatively large number N of spheres where the space left would be small. Placing many spheres in the volume would help the convergence but will slow down the algorithm as with more spheres comes a higher rate of rejected positions. For large N it is even possible that the last few spheres cannot be placed at all! You'll be able to analyze this in the exercises.

4.5 Canonical Monte Carlo

4.5.1 Recap: What is an Ensemble?

This section briefly explains what an ensemble is. It is a short recapitulation of some of the things you most likely already know from your class on thermodynamics. This section is supposed to quickly refresh your memory.

An ensemble is a set of a large number of identical systems. In the context of physics, one usually considers the phase space (which for N particles is $6N$ dimensional) of a given system. The selected points are then regarded as a collection of representative points in phase space. An important and ever-recurring topic is the probability measure (on which the statistical properties depend). Let us say that we have two regions R_1 and R_2 with R_1 having a larger measure than R_2. Consequently, if we pick a system at random from our ensemble, it is more probable that it is in a microstate pertaining to R_1 than R_2. The choice of this probability measure depends on the specific details of the system as well as on the assumptions made about the ensemble.
The normalizing factor of the measure is called the partition function of the ensemble. An ensemble is said to be stationary if the associated measure is time-independent. The most important ensembles are the following:

- Microcanonical ensemble: A closed system with constant number of particles N, constant volume V and constant inner energy E

- Canonical ensemble: A closed system in a heat reservoir with constant N, constant temperature T and constant V
- Grand canonical ensemble: An open system where the chemical potential μ is constant along with V and T.

The ensemble average of an observable (i.e. a real-valued function f) defined on phase space Λ with the probability measure $d\mu$ (restricting to μ-integrable variables) is defined by

$$< f >= \int_{\Lambda} f d\mu$$

The time average is defined in a different way. We start out with a representative starting point $x(0)$ in phase space. Then, the time average of f is given by

$$\bar{f}_t = \lim_{T\to\infty} \frac{1}{T} \int_0^T f(x(t))dt$$

These two averages are connected by the Ergodic hypothesis. This hypothesis states that for long periods of time, the time one particle spends in some region Λ of phase space of microstates with the same energy is directly proportional to the volume $V(\Lambda)$ of that region. One can thus conclude that all the accessible microstates are equiprobable (over a long period of time). In statistical analysis, one often assumes that the time average $\bar{Q}_t$ of some quantity Q and the ensemble average $< Q >$ are the same.

4.5.2 Back to Canonical Monte Carlo

Let the energy of configuration X be given by $E(X)$, then the probability (at thermal equilibrium) for a system to be in X is given by the Boltzmann distribution:

$$p_{eq}(X) = \frac{1}{Z_T} e^{-\frac{E(X)}{k_B T}}$$

where Z_T is the partition function (the normalizing factor of the measure) :

$$Z_T = \sum_X e^{-\frac{E(X)}{k_B T}}$$

When we go from the "individual" probability (at equilibrium) $p_{eq}(X)$ to give us some specific X over to the "global" (overall) probability to give us just any X we want to obtain probability 1 (normalization) :

$$\sum_X p_{eq}(X) = 1$$

Let us now consider an ensemble average which for X discrete becomes

$$< Q >= \sum_X Q(X) p_{eq}(X)$$

Let us say we want to calculate the ensemble average for a given property, e.g. the energy. Unfortunately, the distribution of energy around the time average $\bar{E}_t$ gets sharper with increasing system size (the peak width increases with the system size as $\sqrt{L^{dim}}$ while the system increases with L^{dim} so the relative width decreases as $\frac{1}{\sqrt{L^{dim}}}$).
Consequently, it is inefficient to pick equally distributed configurations over the energy (this is similar to the situation with singularities that we encountered when comparing simple sampling to importance sampling). We are thus still (at least) one central ingredient short for the computation of the ensemble average.

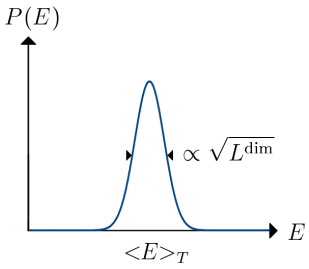


Figure 4.6: Energy distribution [20]

4.5.3 $M(RT)^2$ Algorithm

This algorithm was first described by N.C. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller and E. Teller in 1953. It is a special case of a Markov chain, where the ith element X_i only depends on the previous one, X_{i-1}. More precisely, the idea is that we carry out the previously established method of importance sampling through a Markov chain, going from X_1 to X_2 and so on, where the probability for a given configuration X_j is $p_{eq}(X_j)$.

Properties of Markov Chains

As the $M(RT)^2$ algorithm heavily relies on Markov chains, we should first look at their properties.

We start in a given configuration X and *propose* a new configuration Y with a probability $T(X \to Y)$. Again, we want the sum over all possible new configuration to be unity (normalization).
Furthermore, the probability associated with the transition from a configuration X to a configuration Y needs to be the same as the transition probability of the inverse process, $T(Y \to X)$ ("reversibility").
Last but not least, we recall from thermodynamics the ergodic hypothesis, which states that thermodynamical systems are generally very "chaotic" (molecular chaos) so that all phase space volume elements corresponding to a given energy are equiprobable. More precisely, for "almost all" measurable quantities M the time averaged value $\bar{M}_t$ is equal to the ensemble average $< M >$. In the context of Markov chains this means that a state i is ergodic if it is aperiodic and positive recurrent[2]. If all states in a Markov chain are ergodic, then the chain is called

[2]positive recurrent: If we start in a state i and define T_i to be the first return time to our state i ("hitting time"), $T_i = \inf\{n \geq 1 : X_n = i | X_0 = i\}$, then the state i is **recurrent** if and only if $P(T_i = \infty) = 0$. (i.e. the hitting time is always finite). Even though the hitting time is

ergodic. Consequently, one must be able to reach any configuration X after a finite number of steps.

Let us summarize these properties:

- **Ergodicity:** One must be able to reach any configuration X after a finite number of steps.
- **Normalization:** $\sum_Y T(X \to Y) = 1$
- **Reversibility:** $T(X \to Y) = T(Y \to X)$

Markov Chain Probability

Now that we have established the properties of Markov chains, we want to understand the second step in the process; we recall that the first one is *proposing* a new configuration Y. However, not every new configuration Y is also *accepted*. To illustrate this, let us consider the Ising model again: We may propose a new configuration which will increase the energy of the lattice, but depending on the temperature it is not necessarily likely that the new configuration will also be accepted. At low temperatures, it is unlikely that a spin flip will occur, so we'll handle this with the acceptance probability.

Now that we have a better grasp of the concept, let us formalize these ideas: In analogy to the spin flip in the Ising model of ferromagnetism, we propose a new configuration and define an acceptance probability which will make sure that the model behaves the way we want. We denote the probability of acceptance of a new configuration Y, starting from a configuration X, by $A(X \to Y)$. In practice, we might be interested a bit more in the overall probability of a configuration actually making it through these two steps; this probability is the product of the transition probability $T(X \to Y)$ and the acceptance probability $A(X \to Y)$ and is called the **probability of a Markov chain**:

$$W(X \to Y) = T(X \to Y) \cdot A(X \to Y)$$

If we are interested in the evolution of the probability $p(X, t)$ to find X in time, we can derive this with a logical gedankenexperiment: there are two processes changing this probability:

- A configuration X is produced by coming from Y (this will contribute positively)
- A configuration X is destroyed by going to some other configuration (this will decrease the probability)

finite, the state i does not need to have a finite expectation $M_i = Exp[T_i]$. If however the state is **positively recurrent**, M_i is finite.

The first of these two is proportional to the probability for a system to be in Y, $p(Y)$, while the second one needs to be proportional to the probability for a system to be in X, $p(X)$. When we combine all of this we obtain the so-called **Master equation** :

$$\frac{dp(X,t)}{dt} = \sum_Y p(Y)W(Y \to X) - \sum_Y p(X)W(X \to Y)$$

We can at this point conclude the properties of $W(X \to Y)$:
Similarly to $T(X \to Y)$, the first property is ergodicity, but it manifests itself in a somewhat different way. Given that any configuration X must be reachable after a finite number of steps, the transition probability $T(X \to Y)$ cannot be zero (otherwise a state Z with $T(X \to Z)$=0 would not be reachable) , and acceptance is non-zero as well, so we have $\forall X, Y \quad W(X \to Y) > 0$.
Again, we sum over all possible "target configurations" to obtain unity, giving us the second property (normalization).
The third property, homogeneity, is new. It tells us that if we sum over all "initial configurations", the product of the probability for a system to be in Y multiplied by its Markov probability for the transition to X is given by the probability for a system to be in X. More comprehensibly, the probability for a system to be in X is simply a result of systems coming from other configurations over to X.
Quickly summarized:

- **Ergodicity:** $\forall X, Y \quad W(X \to Y) > 0$
- **Normalization:** $\sum_Y W(X \to Y) = 1$
- **Homogeneity:** $\sum_Y p_{st}(Y)W(Y \to X) = p_{st}(X)$

Note: The attentive reader might object that as $W(X \to Y) = A(X \to Y) \cdot T(X \to Y)$, with both $A(X \to Y)$ and $T(X \to Y)$ being probabilities that have $A \leq 1$, $T \leq 1$, W cannot possibly ever reach $W(X \to Y) = 1$. Consider that we are looking at conditional probabilities here, as A is the probability of acceptance *for a given new configuration* Y. First note that the event of choosing a new configuration $t(x \to y)$ is independent of the probability of accepting it, $a(x \to y)$. We have that

$$\sum_Y P(t) = 1 \quad , \quad \sum_Y P(a) = 1$$

and the conditional probability, "a provided that t" (or in plain English, the probability to accept a new configuration produced by t), is

$$P(a|t) = \frac{P(a \cap t)}{P(t)} = \frac{W(X \to Y)}{T(X \to Y)}$$

where we are now able to identify our original $A(X \to Y)$ easily as the conditional probability $P(a|t)$!

Furthermore, for

$$\sum_i a_i = 1 \quad , \quad \sum_i b_i = 1$$

we have

$$\sum_{ij} a_i b_i = (a_1+a_2+...)(b_1+b_2+...) \stackrel{\sum b_i=1}{=} a_1b_1+a_1(1-b_1)+a_2b_2+a_2(1-b_2)+... = a_1+a_2+... = 1$$

so for two independent events we get 1 again, and thus the expression is normalizable. When we apply this to the example at hand, we find

$$\begin{aligned} \sum_{Y,Z} P(T(X \to Y))P(A(X \to Z)) &= 1 \\ &\stackrel{Y=Z}{=} \sum_Y T(X \to Y)A(X \to Y) = \sum_Y W(X \to Y) \end{aligned}$$

Detailed Balance

Let us now consider a stationary state (i.e. the measure is time independent),

$$\frac{dp(X,t)}{dt} = 0$$

Note that all Markov processes reach a steady state. However, we want to model the thermal equilibrium, so we use the Boltmann distribution,

$$p_{st}(X) = p_{eq}(X)$$

and we obtain

$$\sum_Y p_{eq}(Y)W(Y \to X) = \sum_Y p_{eq}(X)W(X \to Y)$$

and we thus find the **detailed balance condition** (which is a sufficient condition):

$$p_{eq}(Y)W(Y \to X) = p_{eq}(X)W(X \to Y) \tag{4.2}$$

Then, the steady state of the Markov process is the thermal equilibrium. We have achieved this by using the Boltmann distribution for $p(X)$.

4.5.4 Metropolis $M(RT)^2$

In the Metropolis algorithm, the acceptance probability A is defined as

$$A(X \to Y) = \min\left(1, \frac{p_{eq}(Y)}{p_{eq}(X)}\right)$$

We can now insert the Boltmann distribution

$$p_{eq}(X) = \frac{1}{Z_T} e^{-\frac{E(X)}{k_B T}}$$

to find

$$A(X \to Y) = \min\left(1, e^{-\frac{E(Y)-E(X)}{k_B T}}\right) = \min\left(1, e^{-\frac{\Delta E}{k_B T}}\right)$$

Suppose we go to a configuration of lower energy, ΔE will be negative and we would end up with an exponential expression like $\exp(\frac{|\Delta E|}{k_B T})$, at which point the minimum kicks in and sets the expression to one. In other words, the acceptance will be equal to one ("always accept") for transitions to configurations of lower energy.
If we go to a configuration of higher energy, ΔE will be positive and we'll have $\exp(-\frac{|\Delta E|}{k_B T}) < 1$ so the acceptance will increase with the temperature.

Note that a thermal equilibrium is enforced by detailed balance and we impose that the steady state must be a Boltzmann distribution.

Glauber Dynamics

The detailed balance is a very broad condition permitting many approaches. Another such approach was proposed by Glauber (he later obtained the Nobel prize, though not for this contribution but for his work in optics). His proposal for the acceptance probability was

$$A_G(X \to Y) = \frac{e^{-\frac{\Delta E}{k_B T}}}{1 + e^{-\frac{\Delta E}{k_B T}}} \tag{4.3}$$

This also fulfills the detailed balance condition:

$$p_{eq}(X) W(X \to Y) = p_{eq}(Y) W(Y \to X) \tag{4.4}$$

Let us try to prove this.

claim: The acceptance probability (4.3) as proposed by Glauber fulfills the detailed balance condition (4.4).

proof:

- In a first step, let us reduce the condition (4.4) to a simpler expression. We write out
$$W(X \to Y) = A(X \to Y)T(X \to Y)$$
to obtain
$$p_{eq}(X)A(X \to Y)T(X \to Y) = p_{eq}(Y)A(Y \to X)T(Y \to X)$$
where we can use the reversibility (i.e. $T(X \to Y) = T(Y \to X)$) to find
$$p_{eq}(X)A(X \to Y) = p_{eq}(Y)A(Y \to X) \tag{4.5}$$
It is thus sufficient to show that (4.3) fulfills (4.5).

- We can adapt the condition even further by recalling that (see Boltzmann distribution)
$$\frac{p_{eq}(Y)}{p_{eq}(X)} = e^{-\frac{\Delta E}{k_B T}} \tag{4.6}$$
where the partition function Z_T drops out. We can thus rewrite (4.5) by dividing both sides by $p_{eq}(X)$ to obtain
$$A(X \to Y) = \frac{p_{eq}(Y)}{p_{eq}(X)}A(Y \to X) = e^{-\frac{\Delta E}{k_B T}}A(Y \to X) \tag{4.7}$$
It is thus sufficient to show that (4.3) fulfills (4.7).

- To prove that (4.3) does fulfill (4.7), we can simply write
$$\frac{A_G(X \to Y)}{A_G(Y \to X)} = \frac{e^{-\frac{\Delta E}{k_B T}}}{1 + e^{-\frac{\Delta E}{k_B T}}} \cdot \left(\frac{e^{\frac{\Delta E}{k_B T}}}{1 + e^{\frac{\Delta E}{k_B T}}} \right)^{-1} = e^{-\frac{\Delta E}{k_B T}} \cdot \frac{1 + e^{\frac{\Delta E}{k_B T}}}{e^{\frac{\Delta E}{k_B T}} + 1} = e^{-\frac{\Delta E}{k_B T}}$$
and we have thus proven that A_G does in fact fulfill the detailed balance condition. □

The Glauber dynamics have some advantages over Metropolis once you go to low temperatures because of the different formulation of the acceptance.

Literature

- M.H. Kalos and P.A. Whitlock: "Monte Carlo Methods" (Wiley-VCH, Berlin, 2008)
- J.M. Hamersley and D.C. Handscomb: "Monte Carlo Methods" (Wiley and Sons, N.Y., 1964)

- K. Binder and D. Heermann: "Monte Carlo Simulations in Statistical Physics" (Springer, Berlin, B li 1992)
- R.Y. Rubinstein: "Simulation and the Monte Carlo Method" (Wiley and Sons, N.Y., 1981)

4.6 The Ising Model

4.6.1 Introduction

The Ising model is a model that originally aimed at explaining ferromagnetism, but is today used in many other areas such as opinion models and binary mixtures. It is a highly simplified approach to the difficulties of magnetism (e.g. commutation relations of spins).

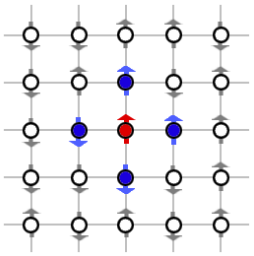


Figure 4.7: Spins on a lattice [20]

We consider a discrete collection of N binary variables called spins, which may take on the values ± 1 (representing the up and down spin configurations). The spins σ_i interact pairwise, and the energy has one value for aligned spins ($\sigma_i = \sigma_j$) and another for anti-aligned spins ($\sigma_i \neq \sigma_j$). The Hamiltonian is given by

$$\mathcal{H} = E = -\sum_{ij} J_{ij}\sigma_i\sigma_j - \sum_i H_i\sigma_i \tag{4.8}$$

where H_i is a (usually homogeneous) external field and J_{ij} are the (translationally invariant) coupling constants. As the coupling constants are translationally invariant, we may drop the indices and simply write $J = J_{ij} \ \ \forall i, j$. The coupling constant J is half the difference in energy between the two possibilities (alignment and anti-alignment).

The simplest example is the antiferromagnetic one-dimensional Ising model, which has the energy function

$$E = \sum_i \sigma_i\sigma_{i+1} \tag{4.9}$$

which can be generalized to the two-dimensional case; we can also add an external field as in equation (4.8).

4.6.2 Monte Carlo of the Ising Model

The Monte Carlo Model of the Ising Model is based on a single flip Metropolis algorithm:

- Choose one site i (having spin σ_i)
- Calculate $\Delta E = E(Y) - E(X) = 2J\sigma_i h_i$
- If $\Delta E < 0$: flip spin: $\sigma_i \to -\sigma_i$
- If $\Delta E > 0$: flip spin with probability $\exp(-\frac{\Delta E}{k_B T})$

where we have introduced h_i which is the approximate local field at site i, given by

$$h_i = \sum_{\text{nn of } i} \sigma_j$$

The sum is characteristic of this approximation; we are only summing over the next neighbors (nn) j of i, in this example we are thus only interested in the local fields of the next neighbors when determining h_i. The notation nn for nearest neighbor is used throughout this chapter.

Note that we evaluate the acceptance probability at each step; given that the acceptance probability is very low, we need a lot of steps for something to change. We will call a group of N steps a "sweep".

Let us consider a magnetic system; we recall the definition of the magnetic susceptibility χ,

$$M = \chi H$$

where M is the magnetization of the material and H is the magnetic field strength. We want to observe its evolution with the ambient temperature T. We find a singularity at the critical temperature T_c, where the magnetic susceptibility χ obeys a power law with critical exponent γ:

$$\chi \propto (T - T_c)^{-\gamma}$$

For finite systems we obtain a maximum instead of a singularity ($\rightarrow$ finite size effects).

A very similar formula is found for magnetization, where we define

$$M(T) = \lim_{H \to 0} \langle \frac{1}{N} \sum_{i=1}^{N} \sigma_i \rangle$$

and find that

$$M \propto |T - T_c|^{\beta}$$

with β depending on the dimension of the model (2D: $\frac{1}{8}$, 3D: ~ 0.326).

As a side comment, note that the binary nature of this model makes it very tempting for a bitwise model (packing 64 spins together). Furthermore, more advanced algorithms actually do not consider locations individually but consider clusters (if you would like to know more, please visit the lecture in the spring semester).

4.6.2.1 Binary Mixtures (Lattice Gas)

A binary mixture is a mixture of two similar compounds. For instance, binary alloys (such as copper nickel alloids) can be described with binary mixtures: Both atoms are roughly the same size but due to energy optimization, the copper and nickel atoms try to form their own, independent clusters. We can denote Cu by $+1$ and Ni by -1 (in analogy to the spins from before). Unlike before, there is one new condition that we did not need to take into consideration in the case of spins: the total number of Cu atoms and the total number of Ni atoms cannot change (while the number of e.g. up spins was free to vary) so there is an additional law of conservation at work.

We include this in the model in the following way:

- E_{AA} is the energy of a A-A bond
- E_{AB} is the energy of a A-B bond
- E_{BB} is the energy of a B-B bond

We set $E_{AA} = E_{BB} = 0$ and $E_{AB} = 1$. Then, the number of each species is constant and our conservation law is hardwired in the model since we only consider pairs of unequal particles (i.e. we can swap but not create/destroy particles). What does this mean in the Ising language? Consider the magnetization:

$$M(T) = \lim_{H\to 0} \langle \frac{1}{N} \sum_{i=1}^{N} \sigma_i \rangle$$

and note that (due to the constant number of atoms of each species) the number of $\sigma_i = +1$ and $\sigma_i = -1$ is constant. To see this, let us divide the indices $\{i\}$ into two subsets:

$$N_a := \{i : 1 \leq i \leq N, \sigma_i = +1\} \quad , \quad N_b := \{i : 1 \leq i \leq N, \sigma_i = -1\} \tag{4.10}$$

so that

$$\begin{aligned} M(T) &= \lim_{H\to 0} \langle \frac{1}{N} \sum_{i=1}^{N} \sigma_i \rangle \\ &= \lim_{H\to 0} \frac{1}{N} \langle \sum_{i\in N_a} \sigma_i + \sum_{i\in N_b} \sigma_i \rangle \\ &= \lim_{H\to 0} \frac{1}{N} \langle \sum_{i\in N_a} (+1) + \sum_{i\in N_b} (-1) \rangle \\ &= \lim_{H\to 0} \frac{1}{N} (|N_a| - |N_b|) \end{aligned}$$

where $|N_a|$ is the number points with $\sigma_i = +1$ (Cu atoms) and $|N_b|$ is the number of points with $\sigma_i = -1$ (Ni atoms). As the number of Cu and Ni atoms is constant, the "magnetization" remains constant and we need new dynamics (Kawasaki dynamics).

4.6.2.2 Kawasaki Dynamics

The Kawasaki dynamics are essentially a Metropolis algorithm that considers the absence of "transfer" between the two species A and B, i.e. their population numbers are constant.

We thus only choose bonds that are on the A-B-boundary and calculate the energy:

- Choose any $(A - B)$ bond
- Calculate ΔE for $(A - B) \rightarrow (B - A)$
- Metropolis: If $\Delta E \leq 0$ flip, else flip with probability $p = \exp(-\beta \Delta E)$
- Glauber: Flip with probability $p = \frac{\exp(-\beta \Delta E)}{1+\exp(-\beta \Delta E)}$ where $\beta = \frac{1}{k_B T}$

We see that this procedure is very similar to the previous one, just with the added condition that the magnetization is constant.

4.7 Interfaces

We have mentioned that the Ising model is not only used for ferromagnetism anymore but has found applications in a variety of other areas.

One such application is the treatment of interfaces. Let us consider two materials A and B with an arbitrary interface (Fig. 4.8). The surface tension γ is then given by the difference in free energy between the compound system $(A+B)$ and the "free" system A:

$$\gamma = f_{A+B} - f_A$$

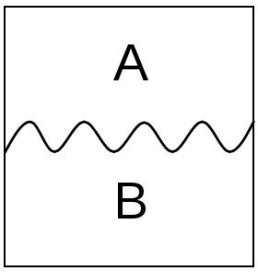


Figure 4.8: Arbitrary interface between two materials A and B [2]

We start with binary variables $\sigma = \pm 1$ again, but we apply fixed boundaries ("+" in the upper half, "-" in the lower half) and populate the lattice. We use Kawasaki

dynamics at temperature T for the simulation as the number of positions (atoms) is constant. The Hamiltonian is given by

$$\mathcal{H} = E = -J \sum_{i,j}^{N} \sigma_i \sigma_j$$

Each point of the interface has a height h measured from the bottom (the average height is denoted by $\bar{h}$) and we can thus define the width W of the interface:

$$W = \sqrt{\frac{1}{N} \sum_{i} (h_i - \bar{h})^2}$$

where N denotes the length of the lattice. In the upcoming sections, we are going to analyze a number of algorithms; we will be particularly interested in the width and the height, as different additional conditions always carry with them a change in the behavior of the width and the height (e.g. we limit the height difference between neighboring positions). This is necessary as the interface will have to vary depending on our simulation needs - a cancer tissue simulation will have to look different from a simulation of a drop on a car windshield.

A special case is the flat surface, where all points at the interface have the same height, $h_i = \bar{h}$ and thus $W = 0$. In general, the width W increases very quickly with the temperature T, exhibiting a singularity at the critical temperature T_c. As we get closer to T_c, the interface becomes more and more diffused and the system starts to "ignore" the boundary conditions, placing "+" entries next to a boundary where we had imposed "−". The transition is known as "roughening transition".

4.7.1 Self-Affine Scaling

We can also analyze the behavior of the width with the variables L (lattice length) and t (time). This behavior is called Family-Vicsek scaling

$$W(L,t) = L^{\xi} f(\frac{t}{L^z})$$

where ξ is the roughening exponent and z the dynamic exponent.
One can now consider two limits: We can either let time go to ∞ or we can consider the limit where $L \to \infty$. To simplify the expression, we substitute the argument of f with $u = \frac{t}{L^z}$. We then find:

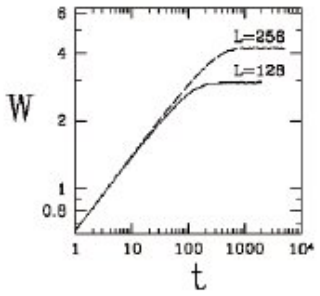


Figure 4.9: Interface width W versus time for two lattice lengths L [16]

- $t \to \infty : W \propto L^{\xi} \Longrightarrow f(u \to \infty) = const$

- $L \to \infty : W \propto t^\beta \Longrightarrow f(u \to 0) \propto u^\beta$

where we have introduced the so-called growth exponent $\beta = \frac{\xi}{z}$. One can numerically verify these laws by observing a data collapse (see the section on scaling laws in the context of percolation).

4.7.2 Expansions of the Model

4.7.2.1 Next-Nearest Neighbors

So far we have used significant simplifications, we have, for instance, only considered next neighbor interactions. One obvious expansion of this simplified model is to also consider next-nearest neighbor interactions:

$$\mathcal{H} = E = -J \sum_{i,j\ nn} \sigma_i \sigma_j - K \sum_{i,j\ nnn}^{N} \sigma_i \sigma_j$$

where nnn denotes the next-nearest neighbors.

The new term introduces stiffness and thus punishes curvature, so that particularly curved surfaces will suffer from this new term and will make the system relax back to a flatter configuration.

4.7.2.2 Shape of a Drop

We can go even further and also include gravity, which will help us understand drops that are attached to a wall. We start with a block of $+1$ sites attached to a wall of an $L \times L$ system filled with -1. The Hamiltonian is given by

$$\mathcal{H} = E = -J \sum_{i,j,\ nn}^{N} \sigma_i \sigma_j - K \sum_{i,j,\ nnn}^{N} \sigma_i \sigma_j - \sum_j h_j \sum_{\text{line } j} \sigma_i$$

where

$$h_j = h_1 + \frac{(j-1)(h_L - h_1)}{L-1} \quad \text{and} \quad g = \frac{h_L - h_1}{L}$$

We then use Kawasaki dynamics (conservation law) and do not permit disconnected clusters of $+1$ (i.e. isolated water droplets in the air). In Fig. 4.10 we see an example ($L = 257$, $g = 0.001$ after 5×10^7 MC updates averaged over 20 samples). We can then define the contact angle Θ, which is a function of the temperature and goes to zero when approaching the critical temperature T_c.

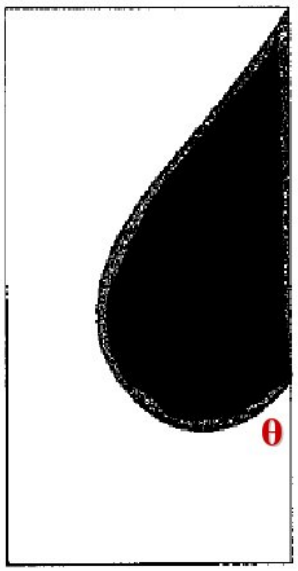


Figure 4.10: Shape of a drop with $L = 257$, $g = 0.001$ [2]

4.7.2.3 Irreversible Growth

For other applications, we can for instance drop the requirement of thermal equilibrium. The range of applications with this additional freedom include:

- Deposition and aggregation points
- Fluid instabilities
- Electric breakdown
- Biological morphogenesis
- Fracture and fragmentation

In the following, we shall get to know some algorithms that help us understand irreversible growth such as those seen in Fig. 4.11.

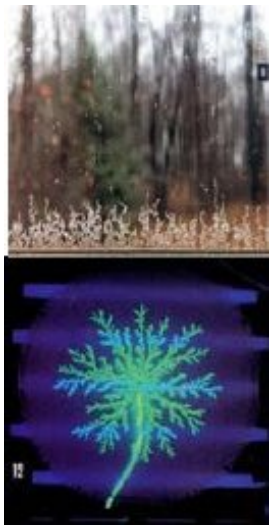


Figure 4.11: Examples of irreversible growth [2]

4.7.2.4 Random Deposition

One of the applications of the methods layed out before with irreversible growth is the random deposition (a good example of this is the growth of ice crystals on a window). The random deposition algorithm represents the simplest possible growth model (which we shall expand later on). The height is directly proportional to the time, $h \propto t$. The width W is proportional to $t^{\frac{1}{2}}$.

The procedure itself is extremely simple as well. It consists but of one step: Pick a randomcolumn and add a particle to that column (see Fig. 4.12).

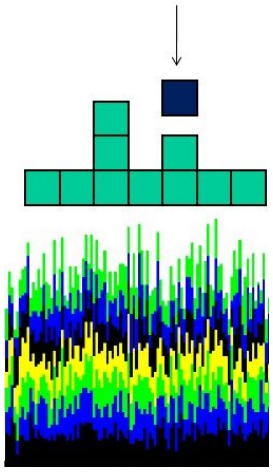


Figure 4.12: Idea and possible result of a random deposition algorithm [2]

The roughening exponent ξ and the growth exponent β are both $\frac{1}{2}$ so the Family-Vicsek scaling law is

$$W(L,t) = \sqrt{L} f(\frac{t}{L})$$

A possible result is illustrated in Fig. 4.12. One immediately notices the sharp spikes which are completely independent from one another, the height varies wildly between columns.

4.7.2.5 Random Deposition with Surface Diffusion

The random deposition model we considered was oversimplified. If we would like to get something more realistic (e.g. less spiky), we need to give the particles a bit more freedom. This is done by permitting the particles to move a short distance to find a more stable configuration.

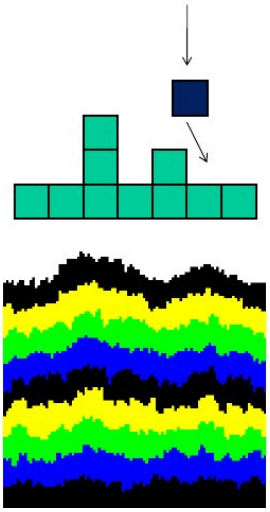


Figure 4.13: Idea and possible result of a random deposition with surface diffusion algorithm [2]

The procedure now has the added complexity of the possible movement:

- Pick a random column i
- Compare the heights $h(i)$ and $h(i+1)$.
- The particle is added to whichever is smaller. For equal heights, the new particle is added to i or $i+1$ with equal probability

This procedure is illustrated (along with a possible result) in Fig. 4.13. With this alternative procedure, the average height now increases with $\sqrt{t}$ while the width W increases with $t^{\frac{1}{4}}$.

The roughening exponent ξ is $\frac{1}{2}$ and the growth exponent β is $\frac{1}{4}$, so the Family-Vicsek scaling law is

$$W(L,t) = \sqrt{L} f(\frac{t}{L^2})$$

From the result we can see that the spikes are much "smoother" and not as jagged anymore as before. The height per column varies more smoothly as well. Nonetheless, the height per column still varies a lot, which we can further restrict by e.g. requiring that it may only differ by a given number of particles. This is exactly the idea behind the next algorithm we are going to look at.

4.7.2.6 Restricted Solid on Solid Model

We add some additional complexity by requiring that neighboring sites may not have a height difference larger than 1, further smoothing out the interface.

The procedure then needs to reflect this latest change with an additional condition (which can lead to the rejection of a particle, if it does not meet the condition).

The new recipe incorporating this change is as follows:

- Pick a random column i
- Add a particle only if $h(i) \leq h(i-1)$ and $h(i) \leq h(i+1)$.
- If the particle could not be placed, pick a different column.

The procedure is illustrated, again along with a possible result, in Fig. 4.14. Due to the rather strict requirement introduced in the RSOS, the height now varies only very little, the interfaces have become very smooth. Again, we may ask for the roughening exponent ξ which is $\frac{1}{2}$ and the growth exponent β which is $\frac{1}{3}$. The Family-Vicsek scaling law for RSOS is thus

$$W(L,t) = \sqrt{L} f(\frac{t}{L^{\frac{3}{2}}})$$

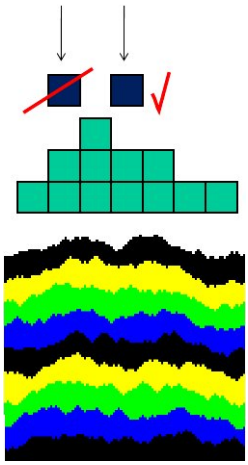


Figure 4.14: Idea and possible result of a RSOS algorithm [2]

4.7.2.7 Eden Model

The Eden model is used in tumor growth and epidemic spread. The idea is that each neighbor of an occupied site has an equal probability of being filled.

The procedure is as follows:

- Make a list of neighbors (red in Fig. 4.15)
- Pick one at random. Add it to the cluster (green)
- Neighbors of this site then become red

There may be more than one growth site in a column, so there can be overhangs. The idea as well as a possible result can be found in Fig. 4.15. The roughening exponent ξ is $\frac{1}{2}$ again while the growth exponent β is $\frac{1}{3}$. The Family-Vicsek scaling is the same as for RSOS.

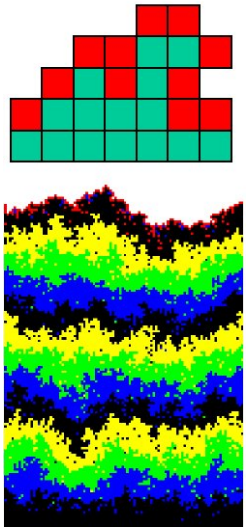


Figure 4.15: Idea and possible result of an Eden algorithm [2]

When we actually grow a cluster (with $\beta = \frac{1}{3}$ and $\xi = \frac{1}{2}$) according to the Eden model (a so-called Eden cluster), which can be used for simulating cancer growth, we might find something like Fig. 4.16. You can see that it is essentially just a big blob with rough edges, which is due to the rule for adding the next particle. Thanks to the relative complexity in comparison to the very oversimplified model of random deposition there are no spikes though (such behavior is always a result of the recipe used for placing a particle).

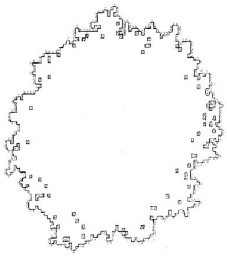


Figure 4.16: Eden Cluster [2]

4.7.2.8 Ballistic Deposition

In yet another model we drop particles from above onto the surface; they stick when they touch a particle below or a neighbor on one side.

Procedure:

- Pick a column i
- Let a particle fall until it touches a neighbor (either below or on either side)

The idea and an example are illustrated in Fig. 4.17. By now we know the drill and directly ask for the roughening exponent which is $\xi = \frac{1}{2}$ and the growth exponent $\beta = \frac{1}{3}$ (these two remain unchanged).

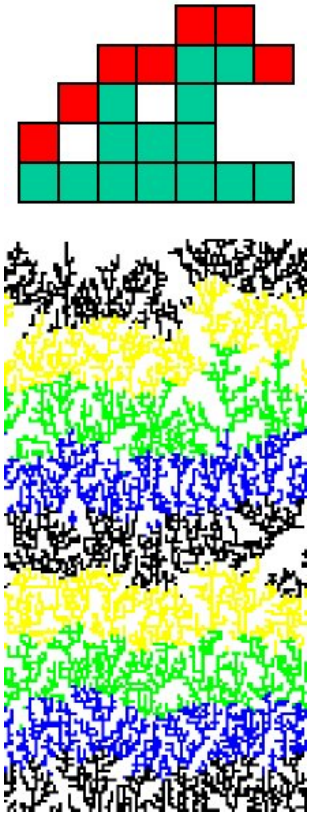


Figure 4.17: Ballistic Deposition, idea and example [2]

4.7.3 Growth

We have seen a handful of algorithms that deal with how to grow patterns according to a given set of rules, simulating e.g. cancer.

Let us formulate an equation for the surface height $h(x,t)$ using symmetry:

1. Invariance under translation in time $t \to t+\delta t$, where δt is a constant
2. Translation invariance along the growth direction: $h \to h + \delta h$, where δh is a constant
3. Translation invariance along the substrate: $x \to x + \delta x$, where δx is a constant

4. Isotropic growth dynamics: e.g., $x \to -x$

5. Conserved relaxation dynamics: The deterministic part can be written as the gradient of a current

Of course not every algorithm is going to respect every single one of these symmetries (nor would we want them to!). Depending on the growth process we envision, a given set of symmetries will need to be respected.

For a growth process respecting all five symmetries, the Edwards-Wilkinson (EW) equation is fulfilled:

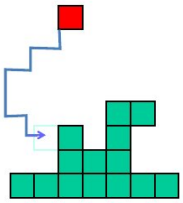
$$\frac{\partial h(x,t)}{\partial t} = \nu \nabla^2 h + \eta(x,t)$$

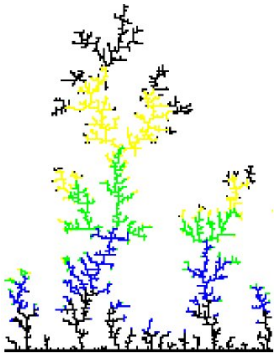
where ν is the surface tension and η is the Gaussian white noise. The corresponding coefficients from before are $\xi = \frac{1}{2}$ and $\beta = \frac{1}{4}$

When we consider nontrivial equilibration (nonlinear equations) we drop the 5^{th} requirement (conserved relaxation dynamics) and find the following equation

$$\frac{\partial h}{\partial t} = \nu \Delta h + \frac{\lambda}{2}(\nabla h)^2 + \eta(x,t)$$

This equation was first proposed by M. Kardar, G. Parisi and Y.-C. Zhang (in 1986). We find $\xi = \frac{1}{2}$ and $\beta = \frac{1}{3}$.





4.7.3.1 Diffusion Limited Aggregation

A particle starts a long way from the surface, it diffuses until it first touches the surface. If it moves too far away, another particle is started. The idea and an example can be found in Fig. 4.18. When you go back to Fig. 4.11 you see the resemblance is rather striking (to the first image). The pattern also resembles electrodepositions and DLA clusters.

Figure 4.18: Diffusion Limited Aggregation, idea and example [2]

4.7.4 Dielectric Breakdown Model (DBM)

As the second last example of algorithms simulating irreversible growth, we want to try to explain the dielectric breakdown.

We start out by solving the Laplacian field equation for ϕ:

$$\triangle \phi = 0$$

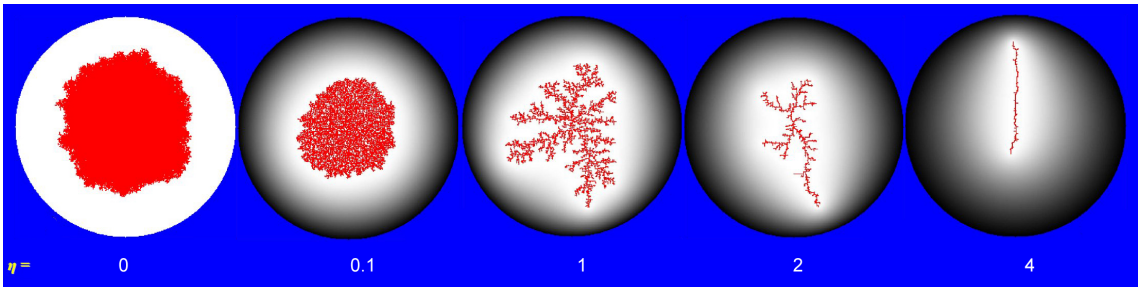
and occupying sites at boundaries with probability

$$p \propto (\nabla \phi)^{\eta}$$

We are now going to consider some special values of η:

- For $\eta = 1$ we find the DLA
- For $\eta = 0$ we find the Eden model (our blob with rough edges)

To see the evolution with η and see the breakdown:



We can identify the similar shape of the third blob with the Diffusion Limited Aggregation, and the very first one with the Eden blob from Fig. 4.16.

4.7.5 Simulated Annealing

Simulated annealing, or SA for short, is a stochastic optimization technique. One often uses SA when the space one considers is discrete. This method can be more effective than exhaustive enumeration (if we are only looking for an acceptable approximation and not the exact solution).

Consider for instance a given set S of solutions and a cost function $F : S \to \mathbb{R}$. We seek a global minimum:

$$s^* \in S^* := \{s \in S : F(s) \leq F(t) \forall t \in S\}$$

Finding the solution will become increasingly difficult for large S, particularly when e.g. $|S| = n!$.

The name of this technique stems from annealing in metallurgy (a technique which involves heating and controlled cooling of a material to increase the size of its crystals and reduce their defects).
In each step of the SA algorithm, one replaces the current solution by a random "nearby" solution which is chosen with a given probability which depends on the difference between the corresponding function values and on a global parameter T (temperature) which is gradually decreased during the process. The current solution changes almost randomly for high temperatures but go more and more "downhill" as we let T approach zero. We permit nonetheless "uphill" movement in order to avoid that the algorithm gets stuck in a local minimum (this will all be explained in the following example).

4.7.5.1 Traveling Salesman

Let us consider an example where we can use simulated annealing. Of course we shall look for a discrete search space, say some cities on a map. Let us thus consider n cities σ_i and the traveling cost from city σ_i to city σ_j given by $c(\sigma_i, \sigma_j)$.

We now look for the cheapest trip through all these cities in order to minimize our costs (and maximize profits). The set S of solutions is given by

$$S = \{\text{permutations of} \{1, ..., n\}\}$$

and the cost function (for a given order, characterized by the permutation π) is

$$F(\pi) = \sum_{i=1}^{n-1} c(\sigma_{\pi_i}, \sigma_{\pi_{i+1}}) \quad \text{for} \quad \pi \in S$$

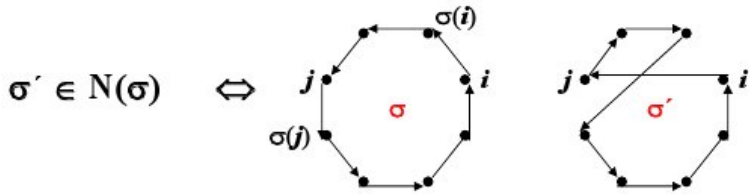
A quick word about permutations: Let us say we have 8 cities, then one possible permutation (the trivial one) is $\{1, 2, 3, 4, 5, 6, 7, 8\}$, and a second one may be $\omega = \{1, 8, 3, 5, 4, 2, 7, 6\}$ (this describes the order of the cities on the trip). For the second permutation, we have $\omega_1 = 1, \omega_2 = 8, \omega_3 = 3$ etc. so that the first three summands in $F(\omega)$ will be $c(\sigma_1, \sigma_8) + c(\sigma_8, \sigma_3) + c(\sigma_3, \sigma_5)$.

Finding the best trajectory is a NP-complex problem, i.e. the time to solve grows faster than any polynomial of n.

We thus make some local changes: we define closed configurations on S:

$$N : S \to 2^S \text{ with } i \in N(j) \leftrightarrow j \in N(i)$$

This can be illustrated in the following way: [2]



Traditionally, one tries to systematically improve costs by exploring close solutions. If $F(\sigma') < F(\sigma)$, $\sigma \mapsto \sigma'$ until $F(\sigma) \leq F(t) \ \ \forall t \in N(\sigma)$. Unfortunately, this poses a new dilemma. Say we have managed to obtain $F(\sigma)$ in this fashion, and we do not find any σ' in the neighborhood that would minimize F further - this still does not guarantee success. Suppose we are dealing with a function that features a local minimum (or local minima), we might get stuck in them and shall not get out by the logic laid out so far.

Instead of this "traditional" optimization algorithm (which falls prey to local minima), we thus introduce the simulated annealing optimization algorithm:

- If $F(\sigma') < F(\sigma)$: replace $\sigma := \sigma'$
- If $F(\sigma') > F(\sigma)$: replace $\sigma := \sigma'$ with probability $exp(-\frac{\Delta F}{T})$ (where $\Delta F = F(\sigma') - F(\sigma) > 0$)

This second step is referred to as "uphill" movement and does not exist in the traditional formulation. The T appearing in the probability is a constant (e.g. temperature). In the course of the algorithm we slowly let T go to zero in order to find the global maximum.

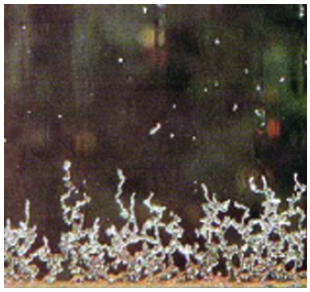
4.7.5.2 Further Examples

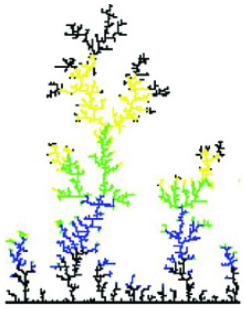
- **Slow cooling:**
  There are different cooling protocols. One common aspect in all of them is that the asymptotic convergence is guaranteed, which leads to an exponential convergence time.
- **Solid on solid model (SSM)**
  Atoms are added from above to a lattice, creating an interface without overhangs or islands. The Hamiltonian is given by

$$\mathcal{H} = E = -\epsilon \sum_{i,j:nn}^{N} |h_i - h_j|$$

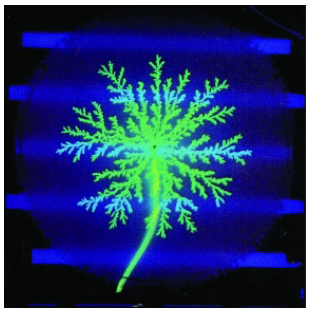
4.8 Simulation Examples

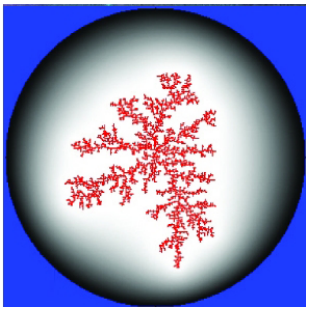
As a visual endpoint to this chapter, here are some of the simulations we have gotten to know and their real life counterparts: [5][4][2]



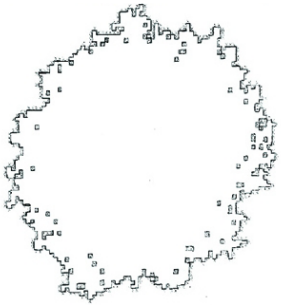


Ice crystals building up on a window; Diffusion Limited Aggregation (DLA)





Vicious fingering; Dielectric Breakdown Model (DBM)



Malignant Melanoma; Eden cluster

Water droplets on a rose bud; shape of a drop

Literature

- M.H. Kalos and P.A. Whitlock: "Monte Carlo Methods" (Wiley-VCH, Berlin, 2008)
- J.M. Hamersley and D.C. Handscomb: "Monte Carlo Methods" (Wiley and Sons, N.Y., 1964)
- K. Binder and D. Heermann: "Monte Carlo Simulations in Statistical Physics" (Springer, Berlin, B li 1992)
- R.Y. Rubinstein: "Simulation and the Monte Carlo Method" (Wiley and Sons, N.Y., 1981)

Part II

Solving Systems of Equations Numerically

Chapter 5

Solving Equations

5.1 One-Dimensional Case

The problem of finding a root of an equation can be written as

$$f(x) = 0$$

where x is to be found. This is equivalent to the optimization problem of finding an extremum of $F(x)$ (where $F'(x) = f(x)$) characterized by

$$\frac{d}{dx}F(x) = 0$$

The function f can be a scalar or a vector function of a vector: $\vec{f}(\vec{x}) \ : \ \mathbb{R}^n \to \mathbb{R}^n$. Let us first look at the one-dimensional case of a scalar function. We shall consider "sufficiently well-behaved" functions throughout the chapter.

5.1.1 Newton's Method

For Newton's method, we need an initial value x_0 ("first guess") and we then linearize the function f around x_0 by only taking the first two terms in the Taylor expansion and dropping all terms of higher order:

$$f(x_0) + (x - x_0)f'(x_0) = 0 \qquad (5.1)$$

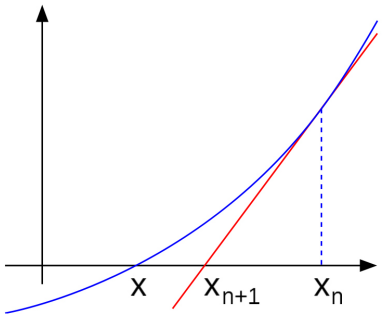


Figure 5.1: Newton iteration [1][21]

or equivalently

$$(5.1) \Rightarrow x - x_0 = -\frac{f(x_0)}{f'(x_0)} \quad \Rightarrow \quad x \ = \ x_0 - \frac{f(x_0)}{f'(x_0)}$$

If we think of x as the next value x_{n+1} of the iteration (and x_0 as the previous one, x_n), we obtain

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{5.2}$$

Newton's method can often converge impressively quickly (in a few steps), particularly if the iteration starts sufficiently close to root. The meaning of "sufficiently close" and "impressively quick" depends on the problem at hand. When one selects an initial value too far away from the root (not "sufficiently close" enough) Newton's method can unfortunately fail to converge completely without the user noticing it.
For an illustration of an iterative step of the Newton method see Fig. 5.1.

5.1.2 Secant Method

The secant method uses a succession of roots of secant lines to approximate the root of a function. It does not require knowledge of the analytical expression of the derivative - we approximate it numerically:

$$f'(x_n) \approx \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}$$

When we insert this into (5.2), we obtain the secant method:

$$x_{n+1} = x_n - (x_n - x_{n-1})\frac{f(x_n)}{f(x_n) - f(x_{n-1})}$$

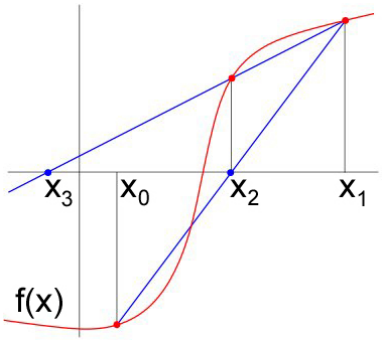


Figure 5.2: Secant method [1][21]

We see that the secant method is Newton's method but with the finite difference approximation instead of the analytical derivative. Newton's method offers faster convergence but requires the evaluation of the function and its derivative at each step, whereas the secant method only requires the evaluation of the function itself. In practice, this may give the secant method an edge, and (depending on the application) can even make it faster in computing time than Newton's method.

The secant method requires two different start values x_0 and x_1 (as opposed to only one initial value for the Newton method). The start values do not have to be as close to the solution as in the Newton method but the secant method does not converge as fast as the Newton method. For an illustration of the secant method see Fig. 5.2.

5.1.3 Bisection Method

Another simple method to solve an equation for its root is the bisection method, which does not require the derivative. The procedure is as follows:

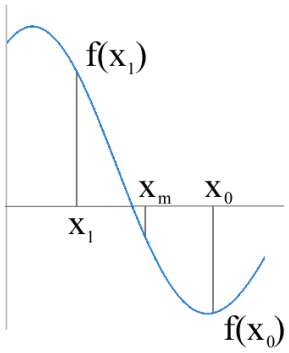


Figure 5.3: Bisection method[20]

1. Take two starting values x_0 and x_1 with $f(x_0) < 0$ and $f(x_1) > 0$.
2. Calculate the mid-point of the two values: $x_m = (x_0 + x_1)/2$.
3. If
$$sign(f(x_m)) = sign(f(x_0))$$
then replace x_0 by x_m, otherwise replace x_1 by x_m.
4. Repeat steps 2-3.

The bisection method is very simple and robust - yet relatively slow. For an illustration of the bisection method see Fig. 5.3.

5.1.4 False Position Method (Regula Falsi)

The false position method is a modification of the bisection method (introducing ideas of the secant method); the procedure is as follows:

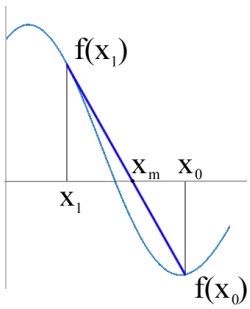


Figure 5.4: False position method [20]

1. Take two starting values x_0 and x_1 with $f(x_0) < 0$ and $f(x_1) > 0$.
2. Approximate f by a straight line between $f(x_0)$ and $f(x_1)$ and calculate the root of this line as
$$x_m = \frac{f(x_0)x_1 - f(x_1)x_0}{f(x_0) - f(x_1)}$$
3. If
$$sign(f(x_m)) = sign(f(x_0))$$
then replace x_0 by x_m, otherwise replace x_1 by x_m.
4. Repeat steps 2-3.

We see that while the secant method retains the last two points, the false position method only retains two points that contain the root. For an illustration of the false position method see Fig. 5.4.

5.2 N-Dimensional Case

As a generalization of the previously discussed methods, we can solve a system of N coupled equations:

$$\vec{f}(\vec{x}) = 0$$

The corresponding N dimensional optimization problem is then

$$\vec{\nabla} F(\vec{x}) = 0$$

where, similar to the aforementioned case, $\vec{\nabla} F = \vec{f}(\vec{x})$. Let us look at the necessary modifications.

5.2.1 Newton's Method

Newton's method in N dimensions replaces the derivative used in the scalar case by the Jacobian matrix:

$$J_{ij}(\vec{x}) = \frac{\partial f_i(\vec{x})}{\partial x_j}$$

Let us consider an example: For a two dimensional function $\vec{f}$, the Jacobian matrix looks as follows:

$$J = \begin{pmatrix} \frac{\partial f_1(\vec{x})}{\partial x_1} & \frac{\partial f_1(\vec{x})}{\partial x_2} \\ \frac{\partial f_2(\vec{x})}{\partial x_1} & \frac{\partial f_2(\vec{x})}{\partial x_2} \end{pmatrix}$$

The Jacobian matrix needs to be non-singular and well-conditioned because it needs to be inverted numerically. The Newton iteration condition in N dimensions is given by

$$\vec{x}_{n+1} = \vec{x}_n - J^{-1}\vec{f}(\vec{x}_n) \tag{5.3}$$

Newton's method in N dimensions can be interpreted as a linearization. We can show this by solving a system of linear equations (due to the nature of the method, we should encounter the solution in exactly one step in the case of a linear equation).

Consider the system of linear equations ($B \in \mathcal{M}_{n\times n}(\mathbb{R})$ and $\vec{x}, \vec{c} \in \mathbb{R}^n$)

$$B\vec{x} = \vec{c} \tag{5.4}$$

whose exact solution is

$$\vec{x} = B^{-1}\vec{c} \tag{5.5}$$

The system (5.4) can also be written as

$$\vec{f}(\vec{x}) = B\vec{x} - c = 0 \quad \Rightarrow \quad J = B \quad \text{(first derivative)} \tag{5.6}$$

Now we apply the Newton method by inserting (5.6) into (5.3):

$$\vec{x}_{n+1} = \vec{x}_n - B^{-1}(B\vec{x}_n - \vec{c}) = B^{-1}\vec{c}$$

and have thus found the exact solution in one step.

5.2.2 Secant Method

The N-dimensional secant method is characterized by the numerically calculated Jacobi matrix (as opposed to the analytical Jacobi matrix from 5.2.1) :

$$J_{i,j}(\vec{x}) = \frac{f_i(\vec{x} + h_j\vec{e}_j) - f_i(\vec{x})}{h_j} \tag{5.7}$$

with $\vec{e}_j$ the unit vector in the direction j. Insert this J into (5.3) and iterate.

h_j should be chosen such that $h_j \approx x_j\sqrt{\epsilon}$, where ϵ is the machine precision (e.g. 10^{-16} for a 64-bit computer)

5.2.3 Other Techniques

We can also solve an N-dimensional system of equations with the relaxation method:

$$\vec{f}(\vec{x}) = 0 \rightarrow x_i = g_i(x_j, j \neq i), \quad i = 1, \ldots, N \tag{5.8}$$

Start with $x_i(0)$ and iterate: $x_i(t+1) = g_i(x_j(t))$.

There are also gradient methods such as the "steepest descent method" or the "conjugate gradient method", which we are not going to go into.

Chapter 6

Ordinary Differential Equations

First order ordinary differential equations (ODE) and initial value problems can be written as

$$\frac{dy}{dt} = f(y,t) \quad \text{with} \quad y(t_0) = y_0 \tag{6.1}$$

If we want to solve an ODE numerically, we have to discretize time so that we can advance in time steps of size Δt.

6.1 Examples

We see examples of ODEs around us every day. The walls around us have slightly radioactive material in them which obeys the equation of radioactive decay

$$\frac{dN}{dt} = -\lambda N$$

with the solution $N = N_0 e^{-\lambda t}$. Another everyday example is the cooling of coffee which obeys the equation

$$\frac{dT}{dt} = -\gamma(T - T_{room})$$

How do we solve such equations numerically? Let us look at some methods, starting out with the Euler method.

6.2 Euler Method

The Euler method is a simple numerical method to solve a first order ODE with a given initial value.

The Euler recipe is as follows:

1. Take the initial value $y(t_0) = y_0$
2. Calculate $\frac{dy}{dt}$.
3. Advance linearly for Δt with the derivative at the initial value as the slope:

$$y(t + \Delta t) = y(t) + \Delta t \cdot y'(t)$$

4. Take the point reached in the previous step as new initial value and repeat steps 2-3.

Let us derive this for finite differences:
We want to approximate the solution of the initial value problem

$$y'(t) = f(t, y(t)) \quad , \qquad y(t_0) = y_0$$

We consider only the first two terms of the Taylor expansion and drop all higher order terms:

$$y(t_0 + \Delta t) = y(t_0) + \Delta t \cdot \frac{dy}{dt}(t_0) + \mathcal{O}((\Delta t)^2) = \underbrace{y(t_0) + \Delta t \cdot f(y_0, t_0)}_{\equiv y(t_1) \equiv y_1} + \mathcal{O}((\Delta t)^2)$$

This corresponds to a linearization around (t_0, y_0). One time step (with $t_{n+1} = t_0 + n \cdot \Delta t$) in the Euler method corresponds to

$$y_{n+1} = y_n + \Delta t \cdot f(t_n, y_n)$$

The solution y_{n+1} is explicit (it is an explicit function of $y_i, 1 \leq i \leq n$). When we start out at y_0 and iterate, we will find y_n; this is the simplest finite difference method. Unfortunately, we also accumulate an error at every time step so we gradually drift away from the exact solution (see Fig. 6.1). The error for each time step is $\propto (\Delta t)^2$.

6.2.1 Interlude: The Order of a Method

In the past, we have analyzed the error for many methods, so before continuing it is helpful to introduce some formalism. We say of a method that it is **locally of order** n if the error at *one time step* is $O((\Delta t)^n)$. However, that is not the whole truth as we are usually interested in the error accumulation over a fixed interval T, consisting of m subintervals Δt, where m is given by

$$m = \frac{T}{\Delta t}$$

The error is thus accumulated additively over T, and we find for the whole interval the global error

$$m \cdot \mathcal{O}((\Delta t)^n) = \frac{T}{\Delta t}\mathcal{O}((\Delta t)^n) = \mathcal{O}((\Delta t)^{n-1})$$

and we therefore say that the method is **globally of order** $(n-1)$. Let us now apply this formalism to the Euler method.

Back to the Euler Method

We have mentioned that the error is locally $\propto (\Delta t)^2$; in the just acquired order language, we say that the Euler method is locally of order two. Globally, the method is not of order two, because one needs $\frac{T}{\Delta t}$ time steps to traverse the whole time interval T:

$$\frac{T}{\Delta t}\mathcal{O}((\Delta t)^2) = \mathcal{O}(\Delta t) \tag{6.2}$$

Therefore, the Euler method is globally of order one. As the error quickly increases with Δt, we need small Δt which is numerically expensive.

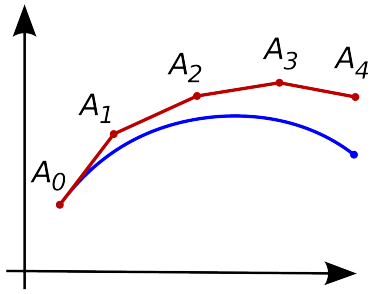


Figure 6.1: Euler method: error accumulation [1]

Second order ODEs can be transformed into two coupled ODEs of first order. Let us consider Newton's equation to illustrate this point:

$$m\frac{d^2x}{dt^2} = F(x) \Rightarrow \begin{cases} \frac{dx}{dt} = v \\ \frac{dv}{dt} = \frac{F(x)}{m} \end{cases} \tag{6.3}$$

This idea can be extended to N coupled first order ODEs:

$$\frac{dy_i}{dt} = f_i(y_1, \ldots, y_N, t), \quad i = 1, \ldots, N \tag{6.4}$$

The Euler algorithm can then be written as

$$y_i(t_{n+1}) = y_i(t_n) + \Delta t f_i[y_1(t_n), \ldots, y_N(t_n), t_n] + \mathcal{O}(\Delta t^2) \tag{6.5}$$

where $t_n = t_0 + n\Delta t$, and the $y_1 \ldots y_N$ are the N functions described by the N ODEs.

6.3 Runge-Kutta Methods

Instead of taking increasingly small time steps to obtain a better approximation to the solution of a given ODE, we can take a completely different approach: We can improve the procedure itself instead of shortening the time steps. Of course the goal of such a change of agenda is that we are not going to need small time steps anymore to get an accurate solution. The way to do this can be seen when going back to the beginning of the Euler method: Instead of dropping all but the first two terms in the Taylor expansion, we could improve the method by getting a better grip on the slope (better than the previous linear approximation).

The Runge-Kutta methods are a generalization of the Euler method, but they achieve the same accuracy as the Euler method for much larger time steps. Therefore, the Runge-Kutta methods are numerically less expensive (yet more complicated to implement).

Formally, the Runge-Kutta methods are derived using a Taylor expansion for $y(t + \Delta t)$ keeping all terms up to order $(\Delta t)^q$:

$$y(t + \Delta t) = y(t) + \frac{(\Delta t)}{1!}\frac{dy}{dt} + \frac{(\Delta t)^2}{2!} \cdot \frac{d^2y}{dt^2} + \cdots + \frac{(\Delta t)^q}{q!} \cdot \frac{d^qy}{dt^q} + \mathcal{O}((\Delta t)^{q+1}) \quad (6.6)$$

We are now going to look at a couple of examples of Runge-Kutta methods and discuss their order later on.

6.3.1 2^{nd} Order Runge-Kutta Method

Let us start with the 2^{nd} order Runge-Kutta method. This is the simplest extension of the Euler method, where we add a term to the previously used linearization. This turns (6.6) into:

$$y(t + \Delta t) = y(t) + \frac{(\Delta t)}{1!}\frac{dy}{dt} + \frac{(\Delta t)^2}{2!} \cdot \frac{d^2y}{dt^2} + \mathcal{O}((\Delta t)^3)$$

We can now formulate the iteration recipe for the 2^{nd} order Runge-Kutta method:

1. Perform an Euler step of size $\frac{\Delta t}{2}$, starting at the initial value $y_i(t_0)$:

$$y_i\left(t + \frac{1}{2}\Delta t\right) = y_i(t) + \frac{1}{2}\Delta t f[y_i(t), t]$$

2. Calculate the derivative at the reached point.

3. Advance a full time step with the calculated derivative as slope.

4. Take the reached point as initial value and repeat steps 1 to 3.

This leads to the following iteration:

$$y_i(t+\Delta t) = y_i(t) + \Delta t f\left[y_i\left(t+\frac{1}{2}\Delta t\right), t+\frac{1}{2}\Delta t\right] + \mathcal{O}((\Delta t)^3) \tag{6.7}$$

This is not the most commonly used Runge-Kutta method though. Most applications using Runge-Kutta have an implementation of the 4^{th} order Runge-Kutta method.

6.3.2 4^{th} Order Runge-Kutta Method

There exist more optimized versions of the Runge-Kutta method than the 2^{nd} order one. A commonly used implementation is the so-called RK4 which is a 4^{th} order Runge-Kutta method.

The procedure is as follows:

1. Define 4 coefficients:

$$\begin{aligned} k_1 &= f(y_n, t_n) \\ k_2 &= f\left(y_n + \tfrac{1}{2}\Delta t \cdot k_1, t_n + \tfrac{1}{2}\Delta t\right) \\ k_3 &= f\left(y_n + \tfrac{1}{2}\Delta t \cdot k_2, t_n + \tfrac{1}{2}\Delta t\right) \\ k_4 &= f(y_n + \Delta t \cdot k_3, t_n + \Delta t) \end{aligned}$$

2. Calculate the next step using a weighted sum of the defined coefficients:

$$y_{n+1} = y_n + \Delta t \cdot \left(\frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4\right) + \mathcal{O}((\Delta t)^5) \tag{6.8}$$

   As you can see, the weighting factors are chosen in such a way that the sum of them is 1: $\frac{1}{6}+\frac{1}{3}+\frac{1}{3}+\frac{1}{6}=1$

This method is globally of order 4 and locally of order 5. The selection of the weighting factors is not unique (see section 6.3.4).

6.3.3 q-Stage Runge-Kutta Method

The definition of the Runge-Kutta methods is more general than what we have seen so far. They are defined by the so called **stage**, which determines the number of terms in the sum of the iteration formula:

$$y_{n+1} = y_n + \Delta t \cdot \sum_{i=1}^{q} \omega_i k_i \tag{6.9}$$

with

$$k_i = f(y_n + \Delta t \cdot \sum_{j=1}^{i-1} \beta_{ij} k_j, \quad t_n + \Delta t \alpha_i) \quad \text{and} \quad \alpha_1 = 0$$

We can conclude that this method uses multiple evaluation points for the function f, leading to a more accurate computation. One might wonder how the weights and the coefficients can be determined. These are described in the Butcher array[1]:

$$\begin{array}{c|ccccc} \alpha_2 & \beta_{2,1} & & & & \\ \alpha_3 & \beta_{3,1} & \beta_{3,2} & & & \\ \vdots & \vdots & & \ddots & & \\ \alpha_q & \beta_{q,1} & \beta_{q,2} & \dots & \beta_{q,q-1} & \\ \hline & \omega_1 & \omega_2 & \dots & \omega_{q-1} & \omega_q \end{array}$$

As you can see, all the parameters of the general Runge-Kutta method are found in this handy table (called Butcher array or Runge-Kutta table).

Let us remain with this array for a second and consider an example for illustration: RK4 with $q = 4$. We know that $\alpha_1 = 0$ (from before) and we recall that in the case of $q = 4$ we have:

- $\omega_1 = \frac{1}{6}$
- $\omega_2 = \frac{1}{3}$
- $\omega_3 = \frac{1}{3}$
- $\omega_4 = \frac{1}{6}$

Furthermore, we remember that $\beta_{ij} = 0$ if $i \neq j + 1$; all of this (and more) is summarized in a few quick lines in the Butcher array for $q = 4$:

$$\begin{array}{c|cccc} 0 & & & & \\ \frac{1}{2} & \frac{1}{2} & & & \\ \frac{1}{2} & 0 & \frac{1}{2} & & \\ 1 & 0 & 0 & 1 & \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array}$$

The Runge Kutta method of stage 4 is of order 4. However, for higher stages, the order does not necessarily correspond to the stage (this is the reason RK is only commonly used up to order 4).

[1]Note that in implicit methods, the coefficient matrix β_{ij} is *not* necessarily lower triangular

6.3.4 Order of the q-Stage Runge-Kutta Method

To construct a Runge-Kutta method of order p, we take the Taylor expansion and calculate the coefficients α_i, β_{ij} and ω_i for $i, j \in [1, p]$ by requiring that in (6.10) the right hand side is zero in $\mathcal{O}((\Delta t)^m)$ for all $m \leq p$:

$$y(t+\Delta t) - y(t) = \underbrace{\sum_{m=1}^{p} \frac{1}{m!} \Delta t^m \cdot \left[\frac{d^{m-1} f}{dt^{m-1}}\right]_{y(t),t}}_{=0} + \mathcal{O}((\Delta t)^{p+1}) \tag{6.10}$$

If we can find such coefficients, the order of the Runge-Kutta method is at least p.

Up to $\mathcal{O}((\Delta t)^{p+1})$ we thus have

$$\sum_{i=1}^{q} \omega_i k_i = \sum_{m=1}^{p} \frac{1}{m!} (\Delta t)^{m-1} \left[\frac{d^{m-1} f}{dt^{m-1}}\right]_{y(t),t}$$

Example: $q = p = 1$

This should lead to the Euler method. There is only one weighting factor, which obviously must be 1:

$$\omega_1 f(y_n, t_n) = f(y_n, t_n) \Rightarrow \omega_1 = 1 \tag{6.11}$$

Example: $q = p = 2$

$$\omega_1 k_1 + \omega_2 k_2 = f_n + \frac{1}{2} \Delta t \left[\frac{df}{dt}\right]_n \tag{6.12}$$

with

$$\left[\frac{df}{dt}\right]_n = \left[\frac{\partial f}{\partial t}\right]_n + \left[\frac{\partial f}{\partial y}\right]_n \cdot \left[\frac{\partial y}{\partial t}\right]_n$$

The subscript n means the evaluation at the point (y_n, t_n). Now insert the coefficients retrieved from (6.9):

$$k_1 = f(y_n, t_n) = f_n$$

$$\begin{aligned} k_2 &= f(y_n + \Delta t \beta_{21} k_1, \quad t_n + \Delta t \alpha_2) \\ &= f_n + \Delta t \beta_{21} \left[\frac{\partial f}{\partial y}\right]_n f_n + \Delta t \alpha_2 \cdot \left[\frac{\partial f}{\partial t}\right]_n + \mathcal{O}((\Delta t)^2) \end{aligned}$$

We can define rules for the weighting factors to derive their values:

- The first rule holds for all Runge-Kutta methods: the sum of the weighting factors must be 1.
$$\omega_1 + \omega_2 = 1$$

- We obtain the second rule by comparing the coefficients in (6.12): we notice that $\left[\frac{\partial f}{\partial t}\right]_n$ shows up with a prefactor of $\frac{1}{2}$. In order to get the same factor on the left hand side, we require
$$\omega_2 \cdot \alpha_2 = \frac{1}{2}$$

- In a similar way we can show that
$$\omega_2 \cdot \beta_{21} = \frac{1}{2}$$

We have found three rules for four parameters. Therefore, the selection of parameters fulfilling all conditions is not unique. However, all choices deliver a method of the demanded order, by construction.

6.3.5 Error Estimation

The estimation of the error of a method is essential if one wants to achieve an accurrate result in the shortest time possible.

However, that is not all that can be done with errors: Imagine we knew the exact error (i.e. the difference between our computed result and the true result). We would then be able to solve the problem exactly after only one step! Unfortunately, we cannot calculate the real error, but we can try to estimate it and adapt our method using the estimated error. The predictor-corrector method discussed in this section will implement this idea to improve the previously discussed methods.

6.3.5.1 Improve Methods Using Error Estimation

As a first estimate, we define the difference between the value after two time steps Δt (called y_2) and the value obtained by performing one time step of size $2\Delta t$ (called y_1):
$$\delta = y_1 - y_2 \tag{6.13}$$

If we consider a method or order p, this definition leads to

$$y(t + 2\Delta t) = \begin{cases} y_1 + (2\Delta t)^{p+1}\Phi + \mathcal{O}((\Delta t)^{p+2}) \\ y_2 + 2(\Delta t)^{p+1}\Phi + \mathcal{O}((\Delta t)^{p+2}) \end{cases} \tag{6.14}$$

$$\Rightarrow \delta = (2^{p+1} - 2)(\Delta t)^{p+1}\Phi + \mathcal{O}((\Delta t)^{p+2}) \tag{6.15}$$

where Φ denotes the evolution operator of the method. By inserting (6.15) into (6.14) we get

$$y(t + \Delta t) = y_2 + \frac{2\delta}{2^{p+1} - 2} + \mathcal{O}((\Delta t)^{p+2}) \tag{6.16}$$

which is a better method of getting the next point, because the error is one order higher than before.

We can improve RK4 using this estimation of the error:

$$y(t + \Delta t) = y_2 + \frac{\delta}{15} + \mathcal{O}((\Delta t)^6)$$

6.3.5.2 Example: Lorenz Equation

The Lorenz equation is a highly simplified system of equations describing the two dimensional flow of a fluid of uniform depth in the presence of an imposed temperature difference taking into account gravity, buoyancy, thermal diffusivity and kinematic viscosity (friction). We introduce the Prandtl number $\sigma = 10$, choose $\beta = \frac{8}{3}$, introduce the Rayleigh number ρ (which is varied). We find chaos for $\rho = 28$. The equations are as follows:

$$\begin{aligned} y_1' &= \sigma(y_2 - y_1) \\ y_2' &= y_1(\rho - y_3) - y_2 \\ y_3' &= y_1 y_2 - \beta y_3 \end{aligned}$$

Interestingly, chaotic solutions of the Lorenz equation do exist and are not simply a result of numerical artefacts (this was proven in 2002 by W. Tucker).

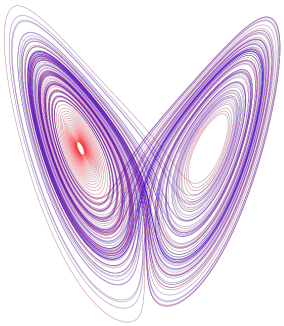


Figure 6.2: Lorentz attractor, calculated by octave (trace starts in red and fades to blue as t progresses) [1]

6.3.5.3 Adaptive Time Step Δt

We could also improve the performance, if we use error estimates to adapt our time step:

- We use large time steps in domains where the solution varies only slightly (i.e. the error is small)
- We use small time steps in domains where the solution varies a lot (i.e. the error is large)

First, we define the largest error that we want to accept, $\delta_{expected}$ and we then measure the real error $\delta_{measured}$. The time step can then be defined as

$$\Delta t_{new} = \Delta t_{old} \left(\frac{\delta_{expected}}{\delta_{measured}} \right)^{\frac{1}{p+1}} \tag{6.17}$$

because $\delta \propto (\Delta t)^{p+1}$

6.3.6 Predictor-Corrector Method

Starting from the error estimation, we can define a new method of improvement for the solution of an ODE: the predictor-corrector method. The idea is to carry out an Euler step with the mean value of the function value at time t and the function value at time $t + \Delta t$:

$$y(t + \Delta t) \approx y(t) + \Delta t \cdot \frac{f(y(t)) + f(y(t + \Delta t))}{2} \tag{6.18}$$

This is an implicit equation, which we cannot solve directly. We make a prediction of $y(t + \Delta t)$ using the Taylor expansion of the function:

$$y^p(t + \Delta t) = y(t) + \Delta t \cdot \frac{dy}{dt}(t) + \mathcal{O}((\Delta t)^2) \tag{6.19}$$

We can compute $y(t + \Delta t)$ in (6.18) using (6.19):

$$y^c(t + \Delta t) = y(t) + \Delta t \cdot \frac{f(y(t)) + f(y^p(t + \Delta t))}{2} + \mathcal{O}((\Delta t)^3) \tag{6.20}$$

The corrected value $y^c(t + \Delta t)$ can itself be inserted into the corrector as the predicted value for a better result. In fact, this can be done several times.

6.3.6.1 Higher Order Predictor-Corrector Methods

The predictor-corrector method can be extended to higher orders if we take more terms of the Taylor expansion for the predictor, for example 4 terms for the 3^{rd} order predictor-corrector method:

$$y^p(t + \Delta t) = y(t) + \frac{(\Delta t)}{1!}\frac{dy}{dt}(t) + \frac{(\Delta t)^2}{2!}\frac{d^2y}{dt^2}(t) + \frac{(\Delta t)^3}{3!}\frac{d^3y}{dt^3}(t) + \mathcal{O}((\Delta t)^4) \tag{6.21}$$

The computation of the corrector then becomes a bit more complicated as we need to define one corrector for the function and two for its derivatives.

Instead of (6.20) we use

$$\left(\frac{dy}{dt}\right)^c (t + \Delta t) = f(y^p(t + \Delta t)) \tag{6.22}$$

The error is then defined as

$$\delta = \left(\frac{dy}{dt}\right)^c (t+\Delta t) - \left(\frac{dy}{dt}\right)^p (t+\Delta t) \tag{6.23}$$

We have to adapt the function itself and its second and third derivative in order to get a complete corrector:

$$\begin{aligned} y^c(t+\Delta t) &= y^p + c_0\delta \\ \left(\frac{d^2y}{dt^2}\right)^c (t+\Delta t) &= \left(\frac{d^2y}{dt^2}\right)^p + c_2\delta \\ \left(\frac{d^3y}{dt^3}\right)^c (t+\Delta t) &= \left(\frac{d^3y}{dt^3}\right)^p + c_3\delta \end{aligned}$$

with the so called Gear coefficients

$$c_0 = \frac{3}{8}, \quad c_2 = \frac{3}{4}, \quad c_3 = \frac{1}{6}$$

These are obtained in a similar way to the RK coefficients in 6.3.4 by requiring the method to be of a certain order. There are also higher order predictor-corrector methods, but the complexity increases very quickly. To get an impression of the ensuing complexity, let us quickly look at the 5^{th} order predictor-corrector method:

Of course, what we know so far is still true even for the 5^{th} order; We have

$$\begin{aligned} \vec{r}_1 &= (\delta t)\left(\frac{d\vec{r}_0}{dt}\right) \\ \vec{r}_2 &= \frac{1}{2}(\delta t)^2\left(\frac{d^2\vec{r}_0}{dt^2}\right) \\ \cdots & \\ \vec{r}_n &= \frac{1}{n!}(\delta t)^n\left(\frac{d^n\vec{r}_0}{dt^n}\right) \end{aligned}$$

and the predictor is then given by

$$\begin{pmatrix} \vec{r}_0^{\,p}(t+\delta t) \\ \vec{r}_1^{\,p}(t+\delta t) \\ \vec{r}_2^{\,p}(t+\delta t) \\ \vec{r}_3^{\,p}(t+\delta t) \\ \vec{r}_4^{\,p}(t+\delta t) \\ \vec{r}_5^{\,p}(t+\delta t) \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 0 & 1 & 3 & 6 & 10 \\ 0 & 0 & 0 & 1 & 4 & 10 \\ 0 & 0 & 0 & 0 & 1 & 5 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \vec{r}_0(t) \\ \vec{r}_1(t) \\ \vec{r}_2(t) \\ \vec{r}_3(t) \\ \vec{r}_4(t) \\ \vec{r}_5(t) \end{pmatrix}$$

with the 1^{st} order equation

$$\frac{dr}{dt} = f(r) \quad \Rightarrow \quad r_1^{\,c} = f(r_0^{\,p}) \quad \Rightarrow \quad \Delta r = r_1^{\,c} - r_1^{\,p}$$

and the 2^{nd} order equation

$$\frac{d^2r}{dt^2} = f(r) \quad \Rightarrow \quad r_2^{\,c} = 2f(r_0^{\,p}) \quad \Rightarrow \quad \Delta r = r_2^{\,c} - r_2^{\,p}$$

The corrector is

$$\begin{pmatrix} \vec{r}_0^{\,c}(t+\delta t) \\ \vec{r}_1^{\,c}(t+\delta t) \\ \vec{r}_2^{\,c}(t+\delta t) \\ \vec{r}_3^{\,c}(t+\delta t) \\ \vec{r}_4^{\,c}(t+\delta t) \\ \vec{r}_5^{\,c}(t+\delta t) \end{pmatrix} = \begin{pmatrix} \vec{r}_0^{\,p}(t+\delta t) \\ \vec{r}_1^{\,p}(t+\delta t) \\ \vec{r}_2^{\,p}(t+\delta t) \\ \vec{r}_3^{\,p}(t+\delta t) \\ \vec{r}_4^{\,p}(t+\delta t) \\ \vec{r}_5^{\,p}(t+\delta t) \end{pmatrix} + \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \end{pmatrix} \cdot \Delta\vec{r}$$

and we see that we need more coeffcients. (Please note that the dot product is to be interpreted as a multiplication which pulls the $\Delta\vec{r}$ vector into each entry of the Gear coefficient vector, as we are dealing with vectors with vector entries). These Gear coefficients are found in the following table:

1st order equation:

Value	c_0	c_1	c_2	c_3	c_4	c_5
3	5/12	1	1/2			
4	3/8	1	3/4	1/6		
5	251/720	1	11/12	1/3	1/24	
6	95/288	1	25/24	35/72	5/48	1/120

2nd order equation:

Value	c_0	c_1	c_2	c_3	c_4	c_5
3	0	1	1			
4	1/6	5/6	1	1/3		
5	19/120	3/4	1	1/2	1/12	
6	3/20	251/360	1	11/18	1/6	1/60

Function	c_0	c_1	c_2	c_3	c_4	c_5
$\dot{\mathbf{r}} = f(\mathbf{r})$	95/288	1	25/24	35/72	5/48	1/120
$\ddot{\mathbf{r}} = f(\mathbf{r})$	3/20	251/360	1	11/18	1/6	1/60
$\ddot{\mathbf{r}} = f(\mathbf{r}, \dot{\mathbf{r}})$	3/16	251/360	1	11/18	1/6	1/60

As we can see from this example of a 5^{th} order PC method, the complexity increases impressively fast.

6.3.6.2 Comparison of Predictor Corrector Methods

We have seen that there are different order predictor-corrector methods, with increasing complexity. Of course, one will immediately ask if this additional complexity, which costs not only computing time but also during the implementation additional debugging time, is worth the hassle. To be able to answer at least the first question, we consider a comparison of several order PC methods for a fixed number of iterations n (for an illustration of the result see Fig. 6.3)
We conclude that it does make a difference for larger time steps, whether we implement a lower order or higher order predictor-corrector. However, this advantage vanishes somewhere between 10^{-2} and 10^{-1}. When we try to choose a time step, we are always making a trade-off:

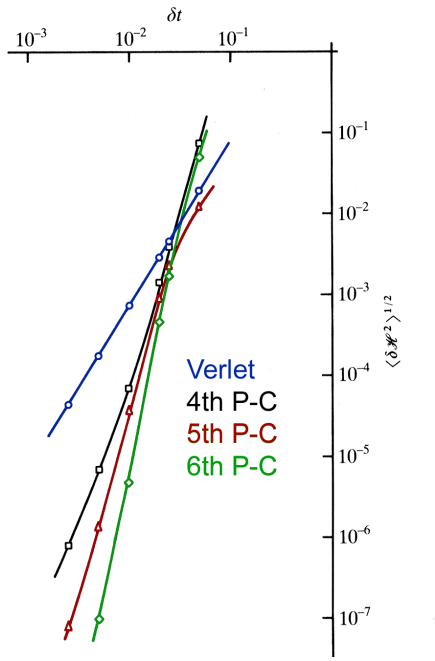


Figure 6.3: Error with respect to size of timestep (log/log scale) [6][21]

- If we choose the time step to be small, the computation will become needlessly slow
- If we choose the time step to be large, the errors will result from approximations

If we choose the time step "just right", then the errors are acceptable while the speed does not needlessly suffer.

6.3.7 Sets of Coupled ODE

We have seen a first example of a coupled ODE with Newton's equation. We can now generalize Runge-Kutta and predictor-corrector methods straight-forwardly to a set of coupled first order ODEs:

$$\frac{dy_i}{dt} = f_i(y_1, ..., y_N, t) \quad , \quad i \in \{1, ..., N\}$$

This is done by inserting simultaneously all the values of the previous iteration.

6.3.8 Stiff Differential Equation

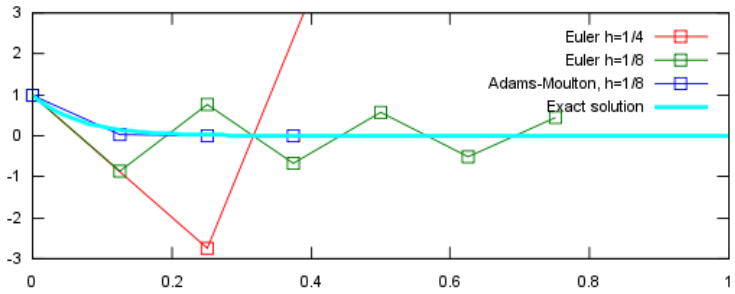
So far, we have considered many methods and improved the error. We have seen that the error depends on the method and the time step, but there is one additional ingredient that can make things worse: stability.

A stiff equation is a differential equation for which certain numerical methods for solving the equation are numerically unstable, unless the step size is taken to be extremely small. It has proven difficult to formulate a precise definition of stiffness, but the main idea is that the equation includes some terms that can lead to rapid variation in the solution.

Let us start by considering a simple example:

$$y'(t) = -15y(t)\ , \quad t \geq 0, y(0) = 1$$

The solution is obviously $y(t) = \exp(-15t)$, and for $t \to \infty$, we have $y(t) \to 0$. When we carry out a numerical calculation with several different methods (Euler with a time step of $\frac{1}{4}$, Euler with a time step of $\frac{1}{8}$, and Adams-Moulton with a step of $\frac{1}{8}$) we observe the following behavior: [1]



We see that Adams-Moulton approximates the solution very well while Euler with a step size of $\frac{1}{8}$ oscillates in a way that is in no way characteristic of the solution. Even worse, Euler with a step size of $\frac{1}{4}$ leaves the range of the graph for good.

6.3.8.1 Stiff Sets of Equations

Of course, this idea generalizes to sets of equations. A system is called "stiff" if a matrix K has at least one large eigenvalue in

$$\vec{y}'(t) = K \cdot \vec{y}(t) + \vec{f}(t)$$

If we have a system

$$\vec{y}'(t) = \vec{f}(\vec{y}(t), t)$$

then the system is called stiff if the Jacobi matrix has at least one large eigenvalue. We then need to solve with an implicit method to avoid instabilities.

Literature

- G.E. Forsythe, M.A. Malcolm and C.B. Moler, "Computer Methods for Mathematical Computations" (Prentice Hall, Englewood Cliffs, NJ, 1977), Chapter 6
- E.Hairer, S.P. Norsett and G. Wanner, "Solving Ordinary Differential Equations I" (Springer, Berlin, 1993)
- W.H. Press, B.P.Flanery, S.A. Teukolsky and W.T. Vetterling, "Numerical Recipes" (Cabridge University Press, Cambridge, 1988) Sect. 16.1 and 16.2
- J.C. Butcher, "The Numerical Analysis of Ordinary Differential Equations" (Wiley, New York, 1987)
- J.D. Lambert, "Numerical Methods for Ordinary Differential Equations" (John Wiley & Sons, New York, 1991)
- L.F. Shampine, "Numerical Solution of Ordinary Differential Equations" (Chapman and Hall, London, 1994)

Chapter 7

Partial Differential Equations

A partial differential equation (PDE) is a differential equation involving an unknown function of several independent variables (including the unknown function's derivatives with respect to the parameters).

To warm up, let us consider a very simple example:

$$\frac{\partial}{\partial x}u(x,y) = 0$$

We immediately see that the function $u(x,y)$ has no x-dependence and can thus be written as $u(x,y) = f(y)$, where $f(y)$ is just an arbitrary function of y. Of course, the same idea can also be done with swapped variables,

$$\frac{\partial}{\partial y}v(x,y) = 0$$

with the solution $v(x,y) = f(x)$ as in this example $v(x,y)$ has no y-dependence. Keeping this rather trivial example in mind, we would like to find a more general expression to characterize partial differential equations so that we are able to work with just about any PDE.

7.1 Types of PDEs

In the introductory example, we considered only first derivatives with respect to the position. In general, time may also be a variable and we are not limited to first derivatives. In fact, we are also going to allow second derivatives and an additional function. In such a more general expression (with up to second derivatives), all of these six terms ($\partial_x u$, $\partial_x^2 u$, $\partial_t u$, $\partial_t^2 u$, $\partial_x \partial_t u$, f) appear with a coefficient (which can be position and time dependent). We can now write out the general

form of the definition of such a PDE with two variables:

$$\begin{aligned} a(x,t)\frac{\partial^2 u(x,t)}{\partial x^2} + b(x,t)\frac{\partial^2 u(x,t)}{\partial x \partial t} + c(x,t)\frac{\partial^2 u(x,t)}{\partial t^2} + \\ d(x,t)\frac{\partial u(x,t)}{\partial x} + e(x,t)\frac{\partial u(x,t)}{\partial t} + f(u,x,t) &= 0 \end{aligned}$$

Based on the coefficients, we distinguish three types of PDEs:

1. A PDE is elliptic, if

$$a(x,t)c(x,t) - \frac{b(x,t)^2}{4} > 0 \tag{7.1}$$

2. A PDE is parabolic, if

$$a(x,t)c(x,t) - \frac{b(x,t)^2}{4} = 0 \tag{7.2}$$

3. A PDE is hyperbolic, if

$$a(x,t)c(x,t) - \frac{b(x,t)^2}{4} < 0 \tag{7.3}$$

Let us apply this knowledge to a couple of examples.

7.2 Examples of PDEs

7.2.1 Scalar Boundary Value Problems (Elliptic PDEs)

The elliptic PDEs can be further subcategorized based on their boundary conditions. There are two boundary conditions:

- The Dirichlet problem has a *fixed value* on the boundary Γ:

$$f(x)|_\Gamma = y_0$$

- The von Neumann problem has a *fixed gradient* on the boundary Γ:

$$\nabla f(x)|_\Gamma = \tilde{y}_0$$

To illustrate this point, let us try to identify the Poisson and the Laplace equation:

- The Poisson equation is

$$\Delta\Phi = \rho(\vec{x}), \quad \Phi(x)|_\Gamma = \Phi_0$$

with the charge distribution $\rho(\vec{x})$ and the fixed value Φ_0 on the boundary Γ. Obviously, the Poisson equation is a Dirichlet problem since the value (not the gradient) is fixed on the boundaries.

- The Laplace equation is

$$\Delta\Phi = 0, \quad \nabla_n \Phi(x)|_\Gamma = \Psi_0$$

with the fixed value Ψ_0 of the gradient of Φ on the boundary Γ. We immediately recognize the von Neumann character of this equation, since the gradient is fixed and not the value.

7.2.2 Vectorial Boundary Value Problem

An example of a vectorial boundary value problem is the Lamé equation of elasticity, which is an elliptic boundary value problem.

Consider a homogeneous isotropic medium $G \in \mathbb{R}^n$ with a boundary Γ, whose state (in the absence of body forces) is given by the Lamé equation:

$$\vec{\nabla}(\vec{\nabla}\vec{u}(\vec{x})) + (1-\nu)\Delta\vec{u}(\vec{x}) = 0 \tag{7.4}$$

where $\vec{u}(x) = (u_1(x_1, ..., x_n), ..., u_n(x_1, ..., x_n))$ is a vector of displacements and ν is the Poisson ratio, which describes how much the material gets thinner in two dimensions if you stretch it in the third one.

7.2.3 Wave Equation

The wave equation describes a wave traveling at a speed c_p; it is a second order linear partial differential equation. One can use it e.g. for sound, light and water waves.

The wave equation is a typical example of a hyperbolic PDE. Let us consider the simplest form, where u is a scalar function $u = u(x,t)$ which satisfies

$$\frac{\partial^2 u(\vec{x},t)}{\partial t^2} = {c_p}^2 \, \Delta u(\vec{x},t) \tag{7.5}$$

with the initial condition

$$u(\vec{x}, t_0) = \tilde{u}_0(\vec{x})$$

The fixed constant c_p corresponds to the propagation speed of the wave, e.g. the speed of sound for acoustic waves. We see that when we consider the coefficients in the general form of a PDE, only the following coefficients are non-zero:

- $c(\vec{x},t)$: the equation contains a $\partial_t^2 u$ expression
- $a(\vec{x},t)$: the equation contains a $\partial_x^2 u$ expression

As all other coefficients are zero, we see that we may arbitrarily choose the sign of a and c, respecting that they appear on opposite sides of the equation and thus have opposite signs. The prefactor $a(x,t)$ (prefactor of Δu) is minus the square of the propagation speed of the wave and the prefactor $c(x,t)$ is one, so we obtain

$$a(\vec{x},t)c(\vec{x},t) - \frac{(b(\vec{x},t))^2}{4} = -c^2 < 0$$

and we have proven that the wave equation really is hyperbolic.

7.2.4 Diffusion Equation

The diffusion equation is a partial differential equation describing density fluctuations in a material undergoing diffusion, such as heat or particle diffusion. The general expression is

$$\frac{\partial \Phi}{\partial t}(\vec{x},t) = \nabla \cdot (\kappa(\Phi,\vec{x})\nabla\Phi(\vec{x},t))$$

where $\Phi(\vec{x},t)$ is the density of the diffusing material at the location $\vec{x}$ at time t and κ is the diffusion coefficient. If we assume that κ is constant, we obtain the simplified expression

$$\frac{\partial \Phi}{\partial t}(\vec{x},t) = \kappa\Delta\Phi(\vec{x},t)$$

Either way, we introduce the boundary condition

$$\Phi(\vec{x},t)|_\Gamma = \Phi_0(t)$$

The diffusion equation is parabolic as can be seen quite easily. We have one first derivative with respect to t (thus $e(\vec{x},t) \neq 0$) and one second derivative with respect to $\vec{x}$ (thus $a(\vec{x},t) \neq 0$). We conclude that $a(\vec{x},t)c(\vec{x},t) = 0$ and $b(x,t) = 0$ so we have directly proven that this equation is parabolic (in accordance with (7.2)).

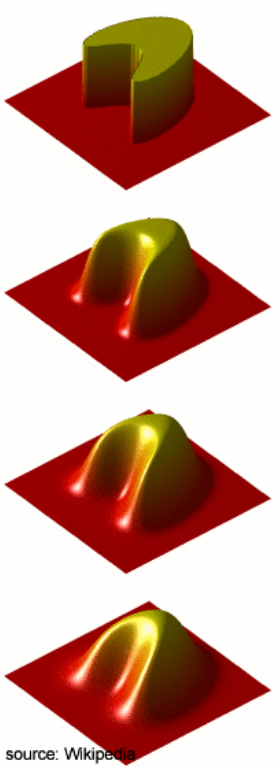


Figure 7.1: A solution of the heat equation (x,y in the plane are spatial directions, the z axis corresponds to Φ and the figures are for increasing t)[1]

To see a solution to the heat equation (where κ is the thermal diffusivity), see Fig. 7.1, where the sequence of images shows the evolution of a solution with time.

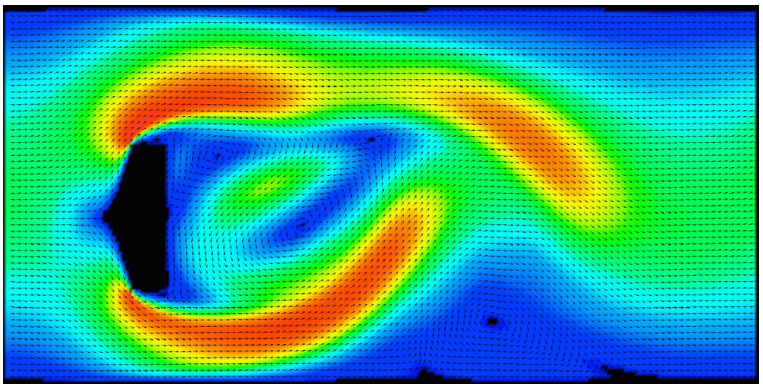


Figure 7.2: Kármán vortex street in a hydrodynamics simulation [7]

7.2.5 Navier-Stokes Equations

The Navier-Stokes equations describe fluid motion. They are commonly used in a wide area of applications, e.g. to model the weather, ocean currents, water flow in a pipe, the air's flow around a wing, and motion of stars inside a galaxy. An example of a solution to the Navier-Stokes equation can be seen in Fig. 7.2 (this example will be explained further towards the end of the chapter).

While the equations give great solutions that accurately describe systems, one needs to take into consideration one of the major disadvantages: they are very hard to solve, particularly for bigger systems.

The Navier-Stokes equations in vector form for an incompressible fluid and constant viscosity (as well as no other body forces) are

$$\rho\left(\frac{\partial\vec{v}}{\partial t}+\vec{v}\vec{\nabla}\cdot\vec{v}\right)=-\vec{\nabla}p+\mu\Delta\vec{v}\ \ ,\qquad \vec{\nabla}\vec{v}=0 \tag{7.6}$$

The condition $\vec{\nabla}\vec{v}=0$ expresses the fact that we are looking at a zero divergence field; as you may remember from electrodynamics, a vector field $\vec{u}$ satisfying $\vec{\nabla}\vec{u}=0$ may be written as $\vec{u}=\vec{\nabla}\wedge\vec{A}$ (since $\vec{\nabla}\cdot\vec{u}=\vec{\nabla}\cdot(\vec{\nabla}\wedge\vec{A})=0$) and thus has neither sources nor sinks (such as the magnetic field for which there are no magnetic monopoles). In the context of fluid motion, the condition $\vec{\nabla}\vec{u}=0$ corresponds to mass conservation (further details are found in section 7.7.4).

The variables appearing in the equation are the following:

$$\begin{aligned}
\vec{v}=\vec{v}(\vec{x},t) &: \text{flow velocity (vector field)}\\
p=p(\vec{x},t) &: \text{pressure}\\
\rho=\rho(\vec{x},t) &: \text{density of the fluid}\\
\mu=\mu(\vec{x},t) &: \text{viscosity of the fluid}
\end{aligned}$$

Furthermore, we need initial conditions and boundary conditions:

$$\left.\begin{array}{rcl} \vec{v}(\vec{x},t_0) & = & \vec{V}_0(\vec{x}) \\ p(\vec{x},t_0) & = & P_0(\vec{x}) \end{array}\right\} \quad \text{defined on the whole domain at time } t_0$$

$$\left.\begin{array}{rcl} \vec{v}(\vec{x},t)|_\Gamma & = & \vec{v}_0(t) \\ p(\vec{x},t)|_\Gamma & = & p_0(t) \end{array}\right\} \quad \text{defined on the boundary } \Gamma \text{ at all times}$$

Later on in this chapter we shall come back to the Navier Stokes equations, but first we need to look a bit more into how to solve PDEs on a computer.

7.3 Discretization of the Derivatives

In order to find the solution of a PDE on a computer, we have to discretize the derivatives on a lattice. Let us denote by x_n the n-th location on the lattice.

7.3.1 First Derivative in 1D

We recall the first derivative in 1D using finite differences:

$$\frac{\partial\Phi}{\partial x} = \frac{\Phi(x_{n+1})-\Phi(x_n)}{\Delta x} + \mathcal{O}(\Delta x) \qquad \text{(two-point formula)} \tag{7.7}$$

$$= \frac{\Phi(x_n)-\Phi(x_{n-1})}{\Delta x} + \mathcal{O}(\Delta x) \qquad \text{(two-point formula)} \tag{7.8}$$

$$= \frac{\Phi(x_{n+1})-\Phi(x_{n-1})}{2\Delta x} + \mathcal{O}((\Delta x)^2) \qquad \text{(three-point formula)} \tag{7.9}$$

One can conclude that the accuracy increases if more points are taken. One can even go so far as to use more than three points to improve the accuracy further.

7.3.2 Second Derivative in 1D

If we apply (7.8) twice to Φ, we obtain the second derivative (using the two-point formula):

$$\begin{aligned} \frac{\partial^2\Phi}{\partial x^2} &= \frac{\partial}{\partial x}\left(\frac{\partial\Phi}{\partial x}\right) \overset{(7.8)}{=} \frac{\partial}{\partial x}\left(\frac{\Phi(x_{n+1})-\Phi(x_n)}{\Delta x} + \mathcal{O}(\Delta x)\right) \\ &= \frac{1}{\Delta x}\left[\left(\frac{\Phi(x_{n+1})-\Phi(x_n)}{\Delta x}\right) - \left(\frac{\Phi(x_n)-\Phi(x_{n-1})}{\Delta x}\right)\right] + \mathcal{O}((\Delta x)^2) \\ &= \frac{\Phi(x_{n+1})+\Phi(x_{n-1})-2\Phi(x_n)}{(\Delta x)^2} + \mathcal{O}((\Delta x)^2) \end{aligned}$$

so we have found the definition of the second derivative in one dimension:

$$\frac{\partial^2\Phi}{\partial x^2} = \frac{\Phi(x_{n+1}) + \Phi(x_{n-1}) - 2\Phi(x_n)}{(\Delta x)^2} + \mathcal{O}((\Delta x)^2) \tag{7.10}$$

A more accurate approximation using the five-point formula for the second derivative is

$$\frac{\partial^2\Phi}{\partial x^2} = \frac{-\Phi(x_{n-2}) + 16\Phi(x_{n-1}) - 30\Phi(x_n) + 16\Phi(x_{n+1}) - \Phi(x_{n+2})}{12(\Delta x)^2} + \mathcal{O}((\Delta x)^4)$$

The derivation is left to the reader as an exercise.

7.3.3 Higher Dimensional Derivatives

In many cases we need derivatives in two or more dimensions. For simplicity, we define $\Delta x = \Delta y = \Delta z$ so that the discretization is the same in all directions. The Laplacian in two dimensions is

$$\begin{aligned}\Delta\Phi = \frac{1}{(\Delta x)^2}[\Phi(x_{n+1}, y_n) &+ \Phi(x_{n-1}, y_n) + \Phi(x_n, y_{n+1}) + \Phi(x_n, y_{n-1}) \\ &- 4\Phi(x_n, y_n)]\end{aligned} \tag{7.11}$$

Where the Δ on the LHS denotes the Laplacian whereas on the RHS, Δ stands for the error. The expression can be seen as a "pattern" that is used on the grid, which creates the value of the derivative at the center point by adding up all the neighbors and subtracting 4 times the value of the center point.

In complete analogy to the 2D case, the Laplacian in three dimensions is

$$\begin{aligned}(\Delta x)^2 \ \Delta\Phi = \Phi(x_{n+1}, y_n, z_n) &+ \Phi(x_{n-1}, y_n, z_n) + \Phi(x_n, y_{n+1}, z_n) \\ &+ \Phi(x_n, y_{n-1}, z_n) + \Phi(x_n, y_n, z_{n+1}) \\ &+ \Phi(x_n, y_n, z_{n-1}) - 6\Phi(x_n, y_n, z_n)\end{aligned}$$

where we need to subtract the center element six times as there are six neighbors (for the sake of readability, the $(\Delta x)^2$ has been multiplied over to the LHS).

7.4 The Poisson Equation

The Poisson equation is

$$\Delta\Phi(\vec{x}) = \rho(\vec{x}) \tag{7.12}$$

where Φ is the scalar electric potential field and ρ is the charge density. Please note that (7.12) is the Poisson equation in SI units; in other sections we will come back to the Poisson equation using cgs units which only changes the prefactor (i.e.

$\Delta\Phi(\vec{x}) = 4\pi\rho(\vec{x})$).

First, we would like to analyze the one-dimensional case with Dirichlet boundary conditions. The Poisson equation in one dimension is

$$\frac{\partial^2\Phi}{\partial x^2} = \rho(x) \tag{7.13}$$

In order to solve this equation, we have to discretize the one-dimensional space; to this end, we pick N points on a given interval: x_n with $n = 1, \ldots, N$. As a short hand, we shall use Φ_n for $\Phi(x_n)$. If we insert the 1D-Poisson equation (7.13) into the definition of the second derivative (7.10) and multiply with $(\Delta x)^2$, we get the discretized equation:

$$\Phi_{n+1} + \Phi_{n-1} - 2\Phi_n = (\Delta x)^2\rho(x_n) \tag{7.14}$$

The Dirichlet boundary conditions are incorporated with

$$\Phi_0 = c_0 \quad \text{and} \quad \Phi_N = c_1.$$

This discretized equation is a set of $(N-1)$ coupled linear equations, which can be written as a matrix in the following way (with $\rho(x) \equiv 0$):

$$\begin{pmatrix} -2 & 1 & 0 & \cdots & 0 \\ 1 & -2 & 1 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & 1 & -2 & 1 \\ 0 & \cdots & 0 & 1 & -2 \end{pmatrix} \cdot \begin{pmatrix} \Phi_1 \\ \\ \vdots \\ \\ \Phi_{N-1} \end{pmatrix} = - \begin{pmatrix} c_0 \\ 0 \\ \vdots \\ 0 \\ c_1 \end{pmatrix} \tag{7.15}$$

This is a system of the form $A\vec{\Phi} = \vec{b}$, which we have to solve for $\vec{\Phi}$.

Poisson Equation in 2 Dimensions

The discretization in two dimensions is rather similar in spirit to the one dimensional case. Of course, instead of a chain with N points x_i we now have an $L \times L$ grid on which we place our points. We assume for simplicity's sake that $\Delta x = \Delta y$ (i.e. quadratic grid). We then have

$$\Phi_{i+1,j} + \Phi_{i-1,j} + \Phi_{i,j+1} + \Phi_{i,j-1} - 4\Phi_{i,j} = (\Delta x)^2\rho_{i,j}$$

and we replace the indices i and j by $k = i + (j-1)L$ to find

$$\Phi_{k+1} + \Phi_{k-1} + \Phi_{k+L} + \Phi_{k-L} - 4\Phi_k = (\Delta x)^2\rho_k$$

These are helical boundary conditions. In fact, we find a system of $N = L^2$ coupled linear equations:

$$A \cdot \vec{\Phi} = \vec{b} \tag{7.16}$$

Let us illustrate this with a simple example. We pick a 5×5 lattice with $\rho = 0$ and $\Phi_m = \Phi_0 \ \ \forall m \in \{n : x_n \in \Gamma\}$ i.e. Dirichlet boundary conditions with fixed Φ_0 on Γ. This will give us an $(L-2)^2 \times (L-2)^2$ matrix, for instance

$$\begin{pmatrix} -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -4 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & -4 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & -4 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & -4 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 \end{pmatrix} \cdot \begin{pmatrix} \Phi_1 \\ \Phi_2 \\ \Phi_3 \\ \Phi_4 \\ \Phi_5 \\ \Phi_6 \\ \Phi_7 \\ \Phi_8 \\ \Phi_9 \end{pmatrix} = - \begin{pmatrix} 2 \\ 1 \\ 2 \\ 1 \\ 0 \\ 1 \\ 2 \\ 1 \\ 2 \end{pmatrix} \Phi_0 \qquad (7.17)$$

We see that this boils down to solving a system of linear equations. The system can be described by

$$A\vec{\Phi} = \vec{b}$$

with the solution

$$\vec{\Phi}^* = A^{-1}\vec{b}$$

and through the Gauss elimination procedure we make the matrix A triangular.

Furthermore, independent of the system size, we notice that each row and column contains only five non-zero matrix elements (see e.g. (7.17)), and is thus not too "crowded" - a sparse matrix. We have seen that in order to find the solution Φ^* we need to invert this matrix; this is done via the LU decomposition.

We have seen that after discretizing a PDE, one is left with a system of linear equations; therefore, we need to have a closer look at how to solve such systems.

7.5 Solving Systems of Linear Equations

After the discretization of a PDE, we are left with a system of coupled linear equations:

$$A\vec{\Phi} = \vec{b} \qquad (7.18)$$

Getting the solution $\vec{\Phi}$ of this system is the most time consuming part - if we were to get the solution of a system of about 10^4 equations, we would have to spend days with the Gauss elimination method. Fortunately, there are other methods to solve systems of this size in less time, though the solution will only be approximate instead of exact. In this section, we shall introduce some of them and consider their advantages and limitations.

7.5.1 Relaxation Methods

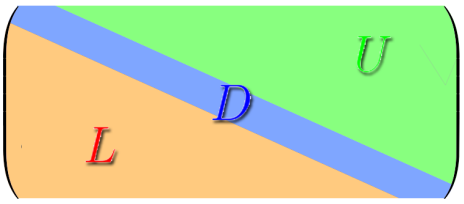
Relaxation methods can be seen as a sort of smoothing operator on the matrices. They provide a realistic behavior in time if we solve for example the heat equation. We can solve even nonlinear problems with these methods if we consider an operator A as a generalization of a matrix. The drawback of these methods is their slow convergence.

7.5.1.1 Jacobi Method

The Jacobi method is the simplest relaxation method for the solution of systems of linear equations. Let us decompose A into the lower triangle L, the diagonal elements D and the upper triangle U:

$$A = D + U + L \tag{7.19}$$

i.e.

$A =$ 

$=$
$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ \vdots & & & & \\ a_{n-1,1} & a_{n-1,2} & a_{n-1,3} & \cdots & a_{n-1,n} \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{pmatrix}$$

For instance, in the case $n = 3$ we have

$$\underbrace{\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}}_{A} = \underbrace{\begin{pmatrix} a_{11} & 0 & 0 \\ 0 & a_{22} & 0 \\ 0 & 0 & a_{33} \end{pmatrix}}_{D} + \underbrace{\begin{pmatrix} 0 & a_{12} & a_{13} \\ 0 & 0 & a_{23} \\ 0 & 0 & 0 \end{pmatrix}}_{U} + \underbrace{\begin{pmatrix} 0 & 0 & 0 \\ a_{21} & 0 & 0 \\ a_{31} & a_{32} & 0 \end{pmatrix}}_{L}$$

Now that we have decomposed A into three different matrices, let us reap the benefits of it!

The Jacobi method is defined as

$$\vec{\Phi}(t+1) = D^{-1}(\vec{b} - (U + L)\vec{\Phi}(t)) \tag{7.20}$$

The Jacobi method is not very fast and the exact solution is only reached for $t \to \infty$. Since we do not need the exact solution but a solution with some error ϵ (called the "precision" or "predefined accuracy"), we can stop if the measured error δ' is smaller than the predefined accuracy ϵ:

$$\delta'(t+1) \equiv \frac{\|\vec{\Phi}(t+1) - \vec{\Phi}(t)\|}{\|\vec{\Phi}(t)\|} \leq \epsilon \tag{7.21}$$

δ' is not the real error, but we will show that it is in most cases a good estimate for it.

Let us start out with the formal definition of the error; quite trivially, the error at a given time step t is the difference between the exact solution $A^{-1}\vec{b}$ and the approximate solution $\Phi(t)$ we have obtained at that time step:

$$\begin{aligned}\vec{\delta}(t+1) &\equiv A^{-1}\vec{b} - \vec{\Phi}(t+1) \\ &= A^{-1}\vec{b} - D^{-1}(AA^{-1})(\vec{b} - (U+L)\vec{\Phi}(t)) \\ &= -D^{-1}(U+L)(A^{-1}\vec{b} - \vec{\Phi}(t)) \\ &= -D^{-1}(U+L)\vec{\delta}(t)\end{aligned}$$

We thus see that from this simple definition (and a bit of plug and play) we have derived an expression that we can rewrite as

$$\vec{\delta}(t+1) = -\Lambda\vec{\delta}(t) \tag{7.22}$$

with the evolution operator Λ of the error (note that Λ is a matrix). The evolution operator establishes a link between the error at one time step and the next (and consequently between the current one and the previous one). We can simply identify Λ in the previous calculation and find

$$\Lambda = D^{-1}(U+L)$$

After n time steps, the error will be the product of evolution operators Λ and the initial "distance"; One can immediately draw some conclusions from this finding:

- As we want to get closer to the solution, the error has to decrease. Consequently, the largest eigenvalue λ of Λ needs to be smaller than one, $|\lambda| < 1$. Generally speaking, λ satisfies $0 < |\lambda| < 1$.
- The smaller the largest absolute eigenvalue λ is, the faster the method will converge.

If we consider enough time steps n, we may write the approximate solution as the sum of the exact solution with an error:

$$\vec{\Phi}_n \approx \vec{\Phi}^* + \lambda^n \cdot \vec{c} \tag{7.23}$$

Of course one issue has not been addressed yet - how do we find λ in the first place? Given that we pick up one factor of λ at each time step, we simply think of λ as the ratio between two time steps and thus conclude that

$$\frac{\|\vec{\Phi}(n+1) - \vec{\Phi}(n)\|}{\|\vec{\Phi}(n) - \vec{\Phi}(n-1)\|} \approx \frac{\lambda^{n+1} - \lambda^n}{\lambda^n - \lambda^{n-1}} = \lambda \tag{7.24}$$

We have mentioned that there is a real error δ and that there is a second error δ' (which was defined in (7.21)). Let us try to find a link between the two. We start out by using the approximation (7.23) in the definition of the error:

$$\delta(n) = \frac{||\vec{\Phi}^* - \vec{\Phi}(n)||}{||\vec{\Phi}(n)||} \overset{(7.23)}{\approx} \frac{||\vec{\Phi}^* - \vec{\Phi}^* - \lambda^n \cdot \vec{c}||}{||\vec{\Phi}(n)||} = \frac{||\vec{c}||}{||\vec{\Phi}(n)||}\lambda^n$$

and we can also use (7.23) to rewrite (7.21) :

$$\delta'(n+1) \equiv \frac{||\vec{\Phi}(n+1) - \vec{\Phi}(n)||}{||\vec{\Phi}(n)||} \overset{(7.23)}{\approx} \frac{||\vec{\Phi}^* + \lambda^{n+1} \cdot \vec{c} - \vec{\Phi}^* - \lambda^n \cdot \vec{c}||}{||\vec{\Phi}(n)||} = \underbrace{\frac{||\vec{c}||}{||\vec{\Phi}(n)||}\lambda^n}_{\delta(n)} |\lambda - 1|$$

so we can link δ and δ'

$$\delta'(n+1) \approx (1-\lambda)\delta(n)$$

i.e.

$$\delta(n) \approx \frac{\delta'(n+1)}{1-\lambda}$$

or we may write

$$\delta(n) \approx \frac{\delta'(n+1)}{1-\lambda} \overset{(7.21)}{=} \left(\frac{1}{1-\lambda}\right) \frac{||\vec{\Phi}(n+1) - \vec{\Phi}(n)||}{||\vec{\Phi}(n)||} \tag{7.25}$$

Of course we can rewrite $1-\lambda$ in a smart way, using (7.24):

$$1 - \lambda = \underbrace{\frac{||\vec{\Phi}(n) - \vec{\Phi}(n-1)||}{||\vec{\Phi}(n) - \vec{\Phi}(n-1)||}}_{1} - \underbrace{\frac{||\vec{\Phi}(n+1) - \vec{\Phi}(n)||}{||\vec{\Phi}(n) - \vec{\Phi}(n-1)||}}_{\lambda}$$

We can use this alternative way of writing $1-\lambda$ to reformulate (7.25) ; we obtain

$$\delta(n) \approx \frac{||\vec{\Phi}(n+1) - \vec{\Phi}(n)||\ ||\vec{\Phi}(n) - \vec{\Phi}(n-1)||}{||\vec{\Phi}(n)|| \left(||\vec{\Phi}(n) - \vec{\Phi}(n-1)|| - ||\vec{\Phi}(n+1) - \vec{\Phi}(n)||\right)} \tag{7.26}$$

Now that we understand the error, let us apply the Jacobi method to our example, the Poisson equation (7.12). We consider the case of a two dimensional grid and start out with any $\Phi_{ij}(0)$. To get from one iteration step n to the next, the procedure is as follows

$$\Phi_{ij}(n+1) = \frac{1}{4}\left(\Phi_{i+1,j}(n) + \Phi_{i-1,j}(n) + \Phi_{i,j+1}(n) + \Phi_{i,j-1}(n)\right) - b_{i,j} \tag{7.27}$$

and the exact solution is given by

$$\Phi^*_{i,j} = \frac{1}{4}\left(\Phi^*_{i+1,j} + \Phi^*_{i-1,j} + \Phi^*_{i,j+1} + \Phi^*_{i,j-1}\right) - b_{i,j}$$

What is crucial in formula (7.27) is its recursive character - we have to save the result at time step n to compute the next result, we cannot simply overwrite its values! It is this characteristic property of the Jacobi method which we will want to correct in the next method (while improving the error, of course).

7.5.1.2 Gauss-Seidel Method

The Gauss-Seidel method does not require keeping a backup of all the data - it just overwrites it. If we consider a grid of $N \times N$ sites, the Gauss-Seidel method will simply calculate the value of the function at a given site using all adjacent sites, independently of whether they have been updated or not (while the Jacobi method computes the new values based on the old ones). To illustrate this point further, consider a small grid such as in Fig. 7.3. All already updated $(n+1)$ sites are red, the un-updated ones (n) are in gray. The value of the site we are looking at (colored yellow) is then given by its neighbors (green), two of which have already been updated.

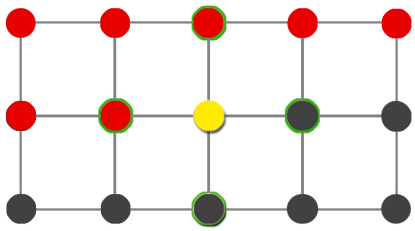


Figure 7.3: Illustration of the Gauss-Seidel method on a grid (red: updated sites, yellow: current site, gray: un-updated sites)[20]

While this may seem somewhat weird at first sight, given that for most elements, the computation will include already updated entries and some that are not updated yet, it does offer the advantage of reduced memory requirements.

Let us formalize the discussion a bit. We decompose our matrix A in the same way as before (see (7.19)) but we combine the elements in a different way:

$$\vec{\Phi}(t+1) = (D+U)^{-1}(\vec{b} - L\vec{\Phi}(t)) \tag{7.28}$$

When we look at the error (and carry out the same calculation as for the Jacobi method) we find that the error evolution operator is given by

$$\Lambda = (D + U)^{-1} L$$

while the stopping criteria is

$$\delta(t) = \frac{\|\vec{\Phi}(t+1) - \vec{\Phi}(t)\|}{(1-\lambda)\|\vec{\Phi}(t)\|} \leq \epsilon$$

Of course we immediately notice that Λ has changed. Most importantly, we see that the denominator has become $(D+U)$ instead of D, making the largest eigenvalue $\lambda_{\max}$ of Λ smaller, consequently decreasing the error at each time step and increasing the convergence speed of the method.

In a nutshell: We have found a method that not only reduces memory requirements (by eliminating the need for a backup) but also converges faster. How can we improve this further?

7.5.1.3 Successive Over-Relaxation (SOR) Method

The successive over-relaxation (SOR) method is a simple generalization of Gauss-Seidel; it tries to improve the convergence of the Gauss-Seidel relaxation method by introducing an over-relaxation parameter ω:

$$\vec{\Phi}(t+1) = (D + \omega U)^{-1} \left(\omega \vec{b} + [(1-\omega)D - \omega L]\vec{\Phi}(t) \right) \tag{7.29}$$

Thanks to the introduction of ω, the denominator in Λ becomes even larger, permitting faster convergence. However, if we push this too far ($\omega > 2$), the algorithm will become unstable (and the solution will blow up).

If we set $\omega = 1$, we are back at the Gauss-Seidel method. Typically, ω is set to a value between 1 and 2 and has to be determined by trial and error.

7.5.1.4 Nonlinear Problems

As previously described, one can solve nonlinear problems with a generalized relaxation. One may consider for example a network of resistors with a nonlinear $I - U$ relation called f (i.e. $I = f(U)$) where each resistor is connected with four neighbors. Kirchhoff's law leads to the following equations:

$$\begin{aligned} & f(U_{i+1,j} - U_{i,j}) + f(U_{i,j} - U_{i-1,j}) \\ + \; & f(U_{i,j+1} - U_{i,j}) + f(U_{i,j} - U_{i,j-1}) = 0 \end{aligned}$$

This means that the current going into the node (i, j) equals the outgoing current. If we reformulate this problem as

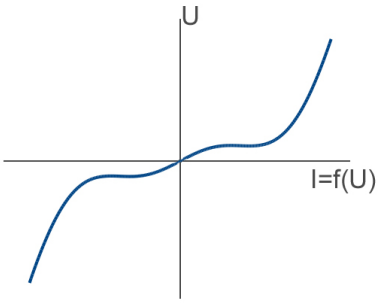


Figure 7.4: Example of a nonlinear relation $I = f(U)$ [20]

$$\begin{aligned} f\left(U_{i+1,j}(t) - U_{i,j}(t+1)\right) \; & + \; f\left(U_{i,j}(t+1) - U_{i-1,j}(t)\right) \\ & + \; f\left(U_{i,j+1}(t) - U_{i,j}(t+1)\right) \\ & + \; f\left(U_{i,j}(t+1) - U_{i,j-1}(t)\right) = 0 \end{aligned}$$

we can solve it for $U_{i,j}(t+1)$ with relaxation.

7.5.2 Gradient Methods

Gradient methods are powerful methods to solve systems of linear equations. They use a functional which measures the error of a solution of the system of equations. If a system of equations has one unique solution, the functional is a paraboloid with its minimum at the exact solution. The functional is defined by the residual $\vec{r}$, which can be seen as an estimate of the error $\vec{\delta}$:

$$\vec{r} = A\vec{\delta} = A(A^{-1}\vec{b} - \vec{\Phi}) = \vec{b} - A\vec{\Phi} \tag{7.30}$$

Clearly, we do not have to invert the matrix A to get the residual. That is precisely why we use it instead of the error. If the residual is small, the error is also going to be small, so we can minimize the functional

$$\mathcal{J} = \vec{r}^T A^{-1} \vec{r} = \begin{cases} 0 & \text{if } \vec{\Phi} = \vec{\Phi}^* \\ > 0 & \text{otherwise} \end{cases} \tag{7.31}$$

where Φ^* is the exact solution. By inserting (7.30) into (7.31), we get

$$\mathcal{J} = (\vec{b} - A\vec{\Phi})^T A^{-1} (\vec{b} - A\vec{\Phi}) = \vec{b}^T A^{-1} \vec{b} + \vec{\Phi}^T A \vec{\Phi} - 2\vec{b}\vec{\Phi} \tag{7.32}$$

Let us denote by Φ_i the ith approximation of the solution and let us define

$$\vec{\Phi} = \vec{\Phi}_i + \alpha_i \vec{d_i}$$

with $\vec{\Phi}_i$ being the start value or the value of the last iteration, $\vec{d_i}$ the direction of the step and α_i the step length. If we insert this into (7.32), we obtain

$$\mathcal{J} = \vec{b}^T A^{-1}\vec{b} + \vec{\Phi}_i^T A\vec{\Phi}_i + 2\alpha_i \vec{d}_i^T A\vec{\Phi}_i + \alpha_i^2 \vec{d}_i^T A\vec{d_i} - 2\vec{b}^T\vec{\Phi}_i - 2\alpha_i \vec{b}^T\vec{d_i}$$

If we minimize the functional $\mathcal{J}$ along the lines given by $\vec{d}$, we will find the best value for α at which $\mathcal{J}$ is minimal: $\bar{\alpha}$. The condition is obviously

$$\frac{\partial \mathcal{J}}{\partial \alpha} = 2\vec{d}_i^t \left(\bar{\alpha}_i A\vec{d_i} - \vec{r}\right) \stackrel{!}{=} 0$$

from which we can conclude that the optimal value is given by

$$\bar{\alpha}_i = \frac{\vec{d}_i^t \vec{r_i}}{\vec{d}_i^t A\vec{d_i}}$$

Of course $\bar{\alpha}_i$ is to be calculated for each step. The most difficult part of the gradient methods is the computation of the direction of the step. Consequently, gradient methods are classified by this feature.

7.5.2.1 Steepest Descent

The most intuitive method of getting the direction is to choose the direction with the largest (negative) gradient. As an analogy, one can think of standing on a mountain top and trying to get down to the valley the quickest way possible, not necessarily the most comfortable way. One would intuitively choose the steepest possible direction. In the case of the steepest descent method, the direction is given by the residual at the point we have reached:

$$\vec{d_i} = \vec{r_i} \tag{7.33}$$

However, the steepest descent method has a major disadvantage which is that it does not take the optimal direction if the functional is not a regular paraboloid. It is a constant "swing and miss" situation as for instance in Fig. 7.5 where the method does lead us to the exact solution in the center, but takes a long time to do so as it keeps changing direction.

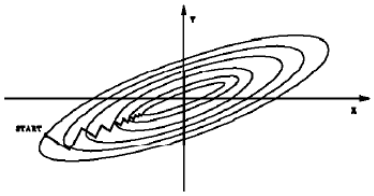


Figure 7.5: An illustration of the gradient descent method on an irregular paraboloid. One clearly sees the "zig-zagging" nature. [14]

The steepest descent algorithm is as follows:

1. Start with $\vec{\Phi}_i$ and choose

$$\vec{d_i} = \vec{r_i} = \vec{b} - A\vec{\Phi}_i$$

2. Evaluate
$$\vec{u}_i = A\vec{r}_i$$
and store the result (we need it twice). Calculate the length of the step:
$$\alpha_i = \frac{\vec{r}_i^2}{\vec{r}_i\vec{u}_i}$$

3. Advance α_i in the direction of $\vec{d}_i$ (which is the same as $\vec{r}_i$) and compute the value of the function at this point:
$$\vec{\Phi}_{i+1} = \vec{\Phi}_i + \alpha_i\vec{r}_i$$

4. Update the residual for the next step:
$$\vec{r}_{i+1} = \vec{r}_i - \alpha_i\vec{u}_i$$

5. Repeat steps 2 to 4 until the residual is sufficiently small.

The vector $\vec{u}_i$ is the temporary stored product $A\vec{r}_i$ so that we need to compute it only once per step since this is the most time consuming part of the algorithm and scales with N^2, where N is the number of equations to be solved. If A is sparse, this matrix-vector product is of order N.

While this method is already very promising, we still want to improve it as the situation as in Fig. 7.5 is somewhat unsettling and inefficient.

7.5.2.2 Conjugate Gradient

We fix the previously observed behavior in the so-called conjugate gradient method. This method takes the functional and deforms it in such a way that it looks like a regular paraboloid - it is only then that it carries out the steepest descent method. In this way, we avoid the behavior we saw previously as we get to root of the problem and get rid of irregular paraboloids.

This can be achieved if we take the new direction conjugate to all previous ones using the Gram-Schmidt orthogonalization process:

$$\vec{d}_i = \vec{r}_i - \sum_{j=1}^{i-1} \frac{\vec{d}_j^T A\vec{r}_i}{\vec{d}_j^T A\vec{d}_j}\vec{d}_j \tag{7.34}$$

Then, all directions $\vec{d}_i$ are conjugate to each other (with respect to A acting as a "metric"):
$$\vec{d}_i^T A\vec{d}_j = \delta_{ij}$$

It needs to be noted at this point that the conjugate gradient method, and gradient methods in general, are only a valuable option for positive and symmetric matrices A.

The algorithm is as follows:

1. Initialize with some vector $\vec{\Phi}_1$ and compute the first residual. Then, take the residual as the direction of your first step:

$$\vec{r}_1 = \vec{b} - A\vec{\Phi}_1, \quad \vec{d}_1 = \vec{r}_1$$

2. Compute the temporary scalar

$$c = (\vec{d}_i^T A \vec{d}_i)^{-1}$$

Note: There is no need for an inversion of the matrix A for this, as we first compute the scalar product which leads to a scalar. However, we need to compute a matrix-vector product, which is of complexity N^2 for dense matrices and N for sparse matrices.

3. Compute the length of the step:

$$\alpha_i = c\vec{r}_i^T \vec{d}_i$$

4. Carry out the step:

$$\vec{\Phi}_{i+1} = \vec{\Phi}_i + \alpha_i \vec{d}_i$$

If the residual is sufficiently small, e.g. $\vec{r}_i^T \vec{r}_i < \epsilon$ (where ϵ is the predefined precision), we can **stop**.

5. Update the residual for the error estimation and for the next step:

$$\vec{r}_{i+1} = \vec{b} - A\vec{\Phi}_{i+1}$$

6. Compute the direction of the next step:

$$\vec{d}_{i+1} = \vec{r}_{i+1} - (c\vec{r}_{i+1}^T A \vec{d}_i)\vec{d}_i$$

7. Repeat steps 2 to 6.

7.5.2.3 Biconjugate gradient

If we are dealing with a matrix that does not fulfill our requirements of a symmetric and positive matrix, we are somewhat outside the possibilities of the conjugate gradient method. Interestingly, we can adapt the gradient method to contain two residuals (hence "biconjugate" gradient method) and the procedure then works beautifully. The residuals are

$$\vec{r} = \vec{b} - A\vec{\Phi} \quad \text{and} \quad \tilde{\vec{r}} = \vec{b} - A^T\vec{\Phi}$$

However, this method does not always converge and can become unstable. The procedure is as follows:

1. Initialize:
$$\begin{aligned} \vec{r_1} &= \vec{b} - A\vec{\Phi} \quad , \quad \vec{d_1} = \vec{r_1} \\ \tilde{\vec{r_1}} &= \vec{b} - A^t\vec{\Phi} \quad , \quad \tilde{\vec{d_1}} = \tilde{\vec{r_1}} \end{aligned}$$

2. Iterate:
$$\begin{aligned} &\vec{r}_{i+1} = \vec{r}_i - \alpha_i A\vec{d}_i \quad , \quad \tilde{\vec{r}}_{i+1} = \tilde{\vec{r}}_i - \alpha_i A^T\tilde{\vec{d}}_i \quad , \quad \alpha = c\vec{r}_i^T\vec{r}_i \\ &\vec{d}_{i+1} = \vec{r}_i + \tilde{\alpha}_i\vec{d}_i \quad , \quad \tilde{\vec{d}}_{i+1} = \tilde{\vec{r}}_i + \tilde{\alpha}_i\vec{d}_i \quad , \quad \tilde{\alpha} = \tilde{c}\tilde{\vec{r}}_i^T\vec{r}_i \\ &\qquad\qquad \text{with} \quad c = (\vec{d}_i^T A\vec{d}_i)^{-1} \quad \text{and} \quad \tilde{c} = (\tilde{\vec{d}}_i^T A\vec{d}_i)^{-1} \end{aligned}$$

3. Stop as soon as
$$\vec{r}_i^T\vec{r}_i < \epsilon \qquad \Rightarrow \qquad \vec{\Phi}_n = \vec{\Phi}_1 + \sum_i^n \alpha_i\vec{d}_i$$

7.5.3 Preconditioning

When using one of the previous methods to solve a system of linear equations, one may notice poor performance because the methods converge very slowly. This is the case if the matrix is badly conditioned: The diagonal elements are almost equal to the sum of the other elements on their row. The solution to this problem is called preconditioning.

The idea is simple: We find a matrix P^{-1} which is cheap to compute and delivers a good approximation of the inverse of the matrix of our problem. (while the idea is simple, the process of finding such a matrix is really not!). We then solve the preconditioned system obtained by a multiplication from the left:

$$\left(P^{-1}A\right)\vec{\Phi} = P^{-1}\vec{b} \tag{7.35}$$

7.5.3.1 Jacobi Preconditioner

Let us consider a first preconditioner, the Jacobi preconditioner. Its name hints at the fact that it shares a core ingredient with the Jacobi method (a member of the group of splitting methods[1]) where we pick the diagonal entries (i.e. in our notation from (7.19), $P = D$). This choice is shared by the Jacobi preconditioner, where we also use the diagonal entries:

$$P_{ij} = A_{ij}\delta_{ij} = \begin{cases} A_{ii} & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

We then invert the obtained diagonal matrix:

$$P_{ij}^{-1} = \delta_{ij}\frac{1}{A_{ii}} \tag{7.36}$$

7.5.3.2 SOR Preconditioner

Another example of a preconditioner is the SOR preconditioner:

$$P = \left(\frac{D}{\omega} + L\right)^{-1} \frac{\omega}{2-\omega} D^{-1} \left(\frac{D}{\omega} + U\right) \tag{7.37}$$

where ω is the relaxation parameter, $0 \leq \omega < 2$. For symmetric matrices with $U = L^T$, the method is called Symmetric Successive Over-Relaxation or SSOR, in which

$$P = \left(\frac{D}{\omega} + L\right)^{-1} \frac{\omega}{2-\omega} D^{-1} \left(\frac{D}{\omega} + L^T\right)$$

[1] The group of splitting methods follows the scheme

$$Ax = b \quad \Rightarrow \quad 0 = -Ax + b \quad \Rightarrow \quad 0 = -(A + P - P)x + b$$

so

$$Px = (P - A)x + b$$

7.5.4 Multigrid Procedure

Let us look at an entirely different approach. When implementing algorithms in the exercises you probably tried out several lattice sizes. If you are given a fixed area (e.g. unit square) for which you are supposed to construct a lattice, increasing the number of points will increase the "resolution". Interestingly, one can combine the advantages of coarse and fine lattices, changing between them - this is the core idea behind the multigrid procedure.

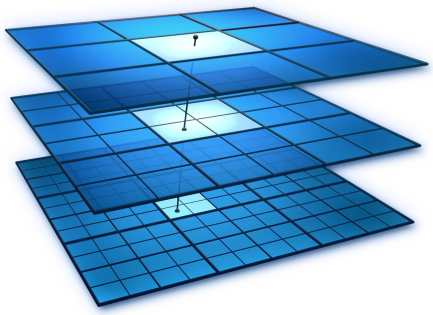


Figure 7.6: Illustration of the multigrid method [8]

If we divide a given area into very few points, we are going to get a bad resolution but the algorithm will finish quickly. For a very fine grid, the algorithm may take ages but the result will be beautiful.

The strategy is to solve the equation for the error on the coarsest grid.

The implementation is a two-level procedure:

1. Determine the residual $\vec{r}$ on the original lattice:

$$\vec{r}_n = \vec{b}_n - A\vec{\Phi}_n \quad , \quad \vec{\delta}_n = A^{-1}\vec{r}_n$$

2. Define the residual on the coarser lattice through a *restriction operator* $\mathcal{R}$:

$$\hat{\vec{r}}_n = \mathcal{R}\vec{r}_n$$

   The restriction operator $\mathcal{R}$ reduces the original system to a smaller system (the coarser lattice).

3. Obtain the error on the coarser lattice solving the equation

$$\hat{A}\hat{\vec{\delta}}_{n+1} = \hat{\vec{r}}_n$$

4. Find the error of the original lattice by using the *extension operator* $\mathcal{P}$:

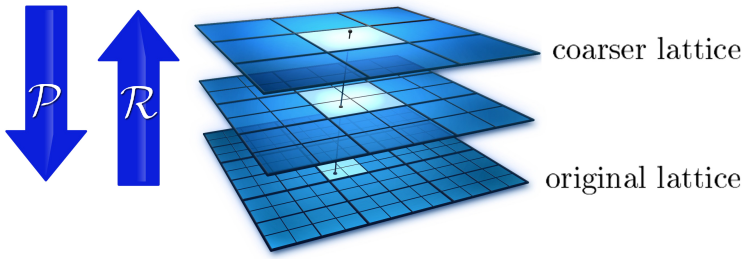
$$\vec{\delta}_{n+1} = \mathcal{P}\hat{\vec{\delta}}_{n+1}$$

   The operator $\mathcal{P}$ approximates the original lattice from the coarser one that we reached using $\mathcal{R}$.

5. Get a new approximate solution

$$\vec{\Phi}_{n+1} = \vec{\Phi}_n + \vec{\delta}_{n+1}$$

The general idea can thus be seen quite easily - we are jumping back and forth between the complex system and the easier system with $\mathcal{P}$ and $\mathcal{R}$



We solve the equations (using e.g. Gauss elimination if the system is small enough) on the coarser lattice and then reproduce the original matrix out of the smaller one, using the extension operator $\mathcal{P}$ (e.g. an interpolation). We can go back and forth several times to reduce a huge system to a system that is solvable by Gauss elimination. In this case, the error of the method is only dependent on the quality of the restriction and extension operators!

We note that

$$\mathcal{R}\mathcal{P}\hat{\vec{r}} = I\hat{\vec{r}} \quad , \quad \mathcal{P}\mathcal{R}\vec{r} = I\vec{r} \tag{7.38}$$

Let us consider an example of such restriction and extension operators:

- The restriction operator $\mathcal{R}$ takes in each direction every second entry (overall thus one entry out of four) and calculates the matrix entry of the coarser grid by adding up the site's value plus the values of its neighbors. The coarser grid has half the side length of the original one. We can write this operator explicitly:

$$\mathcal{R}\vec{r} \mapsto \left\{ \begin{array}{lcl} \hat{r}_{i,j} & = & \frac{1}{4}r_{i,j} + \frac{1}{8}(r_{i+1,j} + r_{i-1,j} + r_{i,j+1} + r_{i,j+1}) \\ & + & \frac{1}{16}(r_{i+1,j+1} + r_{i-1,j+1} + r_{i-1,j+1} + r_{i-1,j-1}) \end{array} \right. \tag{7.39}$$

  The nearest neighbors (nn) are weighed twice as much as the next nearest neighbors (nnn). This is due to the fact that the diagonal entries are counted in four "coarser" sites while the nn only contribute to two such sites.

- The extension operator $\mathcal{P}$ in this example is as follows:

$$\mathcal{P}\hat{\vec{r}} \mapsto \left\{ \begin{array}{lcl} r_{2i,2j} & = & \hat{r}_{i,j} \\ r_{2i+1,2j} & = & \frac{1}{2}(\hat{r}_{i,j} + \hat{r}_{i+1,j}) \\ r_{2i,2j+1} & = & \frac{1}{2}(\hat{r}_{i,j} + \hat{r}_{i,j+1}) \\ r_{2i+1,2j+1} & = & \frac{1}{4}(\hat{r}_{i,j} + \hat{r}_{i+1,j} + \hat{r}_{i,j+1} + \hat{r}_{i+1,j+1}) \end{array} \right. \tag{7.40}$$

The operators $\mathcal{P}$ and $\mathcal{R}$ fulfill one rather interesting condition, but to be able to stomach it well we need to first talk about function spaces. Let us pick any function living on the original grid and denote it by $u(x, y)$. Obviously, a function living on the original grid will take its arguments from the original grid so the arguments are x and y (i.e. the coordinates of the original grid).
If we pick a second function living on the coarser grid and denote it by $\hat{v}(\hat{x}, \hat{y})$ (using $\hat{}$ to refer to elements on the coarser grid), we see that it will take its arguments from coordinates of the coarser grid, i.e. $\hat{x}, \hat{y}$ and not from the original grid (i.e. x, y).
While we already know that we can change between the coarser grid and the original grid with $\mathcal{R}$ and $\mathcal{P}$, there is a special relation that applies to functions living on these two grids: If we consider two functions $u(x, y)$ (on the original grid) and $\hat{v}(\hat{x}, \hat{y})$ (on the coarser grid)as introduced, we can show with a given scalar product that the extension operator and the restriction operator are adjoint:

$$\sum_{x,y} \mathcal{P}\hat{v}(\hat{x}, \hat{y}) \cdot u(x, y) = h^2 \sum_{\hat{x},\hat{y}} \hat{v}(\hat{x}, \hat{y}) \cdot \mathcal{R}u(x, y)$$

Here, h denotes the scaling factor of the finer matrix to the coarser matrix which is given by the coefficient of the respective number of rows. In our example h is 2.

7.6 Finite Element Method

7.6.1 Introduction

The problem of the finite differences method is the regular mesh that cannot be adapted to the problem, so it is not possible to take region-dependent mesh sizes. This is particularly disturbing as we would want to refine the mesh locally in regions of big gradients (the error depends on the ratio of gradient to mesh size) while leaving other regions with small gradients untouched.

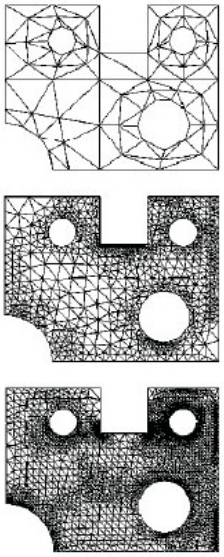


Figure 7.7: Adaptive meshing done with triangulation at three different resolutions [2]

The finite element methods help us out of this dilemma of "rigid mesh sizes". They discretize the PDE by patching together continuous functions instead of discretizing the field on sites. With finite element methods , PDEs can be solved for irregular geometries, inhomogeneous fields (e.g. with moving boundaries) and non-linear PDEs. While the solution is computed, the mesh can even be adapted in order to speed up convergence.

7.6.2 Examples

Let us first consider a couple of examples to get a basic grip of the concepts before introducing the more formal definitions.

7.6.2.1 Poisson Equation in One Dimension

We recall the Poisson equation

$$\frac{d^2\Phi}{dx^2}(x) = -4\pi\rho(x)$$

with the Dirichlet boundary condition

$$\Phi(x)|_\Gamma = 0$$

Earlier on, we solved this equation by discretizing space, which is what we would like to avoid. Instead, we try to expand the field Φ in terms of localized basis functions $u_i(x)$:

$$\Phi(x) = \sum_{i=1}^{\infty} a_i u_i(x) \approx \Phi_N(x) = \sum_{i=1}^{N} a_i u_i(x) \tag{7.41}$$

We approximate the exact solution $\Phi(x)$ with $\Phi_N(x)$, which is the superposition of a finite number of basis functions (instead of an infinite number as in the exact solution). Each basis function has a coefficient a_i. We can obtain the expansion coefficients a_i from (7.41) by introducing so-called "weight functions" $w_j(x)$:

$$-\sum_{i=1}^{N} \int_0^L a_i \frac{d^2 u_i}{dx^2}(x) w_j(x) dx = 4\pi \int_0^L \rho(x) w_j(x) dx \quad , \quad j \in \{1, ..., N\} \tag{7.42}$$

The simplest case is the so-called Galerkin method where we simply pick the weight functions to be the basis functions, i.e. $w_j(x) = u_j(x)$.

We find N coupled equations. When we go from derivatives of Φ to derivatives of u_j, we do not have to find the derivatives numerically (for a sensibly chosen basis of functions), which significantly simplifies the problem. When we separate and insert, we obtain

$$A\vec{a} = \vec{b} \tag{7.43}$$

where $\vec{a}$ is the N-dimensional vector whose entries are the expansion coefficients from (7.41). Let us now try to find an explicit expression for the matrix and vector entries, for which we go back to (7.42):

$$\sum_{i=1}^{N} a_i \underbrace{\left(-\int_0^L \frac{d^2 u_i}{dx^2}(x) w_j(x) dx \right)}_{A_{ij}} = \underbrace{4\pi \int_0^L \rho(x) w_j(x) dx}_{b_j} \quad , \quad j \in \{1, ..., N\}$$

so the matrix element is given by

$$A_{ij} = -\int_0^L u_i''(x) w_j(x) dx \overset{P.I.}{=} \int_0^L u_i'(x) w_j'(x) dx$$

where we have used partial integration, and the vector b_j can be read off as

$$b_j(x) = 4\pi \int_0^L \rho(x) w_j(x) dx \tag{7.44}$$

As a side comment, if we use hat functions as a basis, we obtain matrix elements A_{ij} that look like the discrete elements. We have been talking a lot about basis functions so far, so let us consider an example of basis functions.

7.6.2.2 Example of Basis Functions

An example for basis functions $u_i(x)$ are the hat functions centered around the entry x_i.

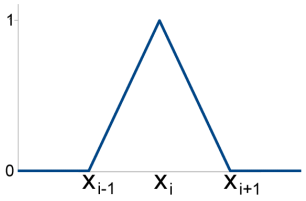


Figure 7.8: Hat basis function in one dimension [20]

We first define the basic element of this basis (after which the method is named), which is the distance between two consecutive elements:

$$\Delta x = x_i - x_{i-1} \tag{7.45}$$

We can then define the hat functions:

$$u_i(x) = \begin{cases} (x_i - x_{i-1})/\Delta x & \text{for } x \in [x_{i-1}, x_i] \\ (x_{i+1} - x_i)/\Delta x & \text{for } x \in [x_i, x_{i+1}] \\ 0 & \text{otherwise} \end{cases} \tag{7.46}$$

from which we can conclude that

$$A_{ij} = \int_0^L u_i^{'}(x) u_j^{'}(x) dx = \begin{cases} 2/\Delta x & \text{for } i = j \\ -1/\Delta x & \text{for } i = j+1 \\ 0 & \text{otherwise} \end{cases}$$

An illustration of the hat function basis is given in Fig. 7.8. In this example, the boundary conditions are automatically satisfied as the basis functions are all zero at both ends.

Let us consider an interval $[0, L]$ on which we are trying to solve the Poisson equation. If we have

$$\Phi(0) = \Phi_0 \quad , \quad \Phi(L) = \Phi_1 \tag{7.47}$$

we can use the decomposition

$$\Phi_N(x) = \frac{1}{L}\left(\Phi_0(L-x) + \Phi_1 x \sum_{i=1}^{N} a_i u_i(x)\right)$$

We are going to look into basis functions more in depth in section 7.6.7.

7.6.2.3 Non-Linear PDEs

As a last example, let us look at a one dimensional non-linear PDE:

$$\Phi(x)\frac{d^2\Phi}{dx^2}(x) = -4\pi\rho(x) \tag{7.48}$$

If we pull the right hand side from (7.42) over (without using the expansion into localized functions), we obtain for (7.48) the relation

$$\int_0^L \left[\Phi(x)\frac{d^2\Phi}{dx^2}(x) + 4\pi\rho(x)\right] w_k(x) dx = 0$$

which we have to solve. This is approximately equivalent to the coupled non-linear system of equations given by

$$\sum_{i,j} A_{ijk} a_i a_j = b_k \quad \text{with} \quad A_{ijk} = -\int_0^L u_i(x) u_j''(x) w_k(x) dx$$

We may solve such a system using the Picard iteration:

Picard Iteration

The Picard iteration consists of three simple steps:

- Start with a guess for Φ_0
- Solve the linear equation for Φ:

$$\Phi_0(x) \frac{d^2\Phi_1}{dx^2}(x) = -4\pi\rho(x)$$

- Iterate:

$$\Phi_n(x) \frac{d^2\Phi_{n+1}}{dx^2}(x) = -4\pi\rho(x)$$

7.6.3 Function on Element

We have so far only considered one dimension; to be useful in practice, we need to go to at least two dimensions (if not more). In two dimensions, we define the function over one element, which is a triangle of the triangulation.

One possible approach is linearization

$$\Phi(\vec{r}) \approx c_1 + c_2 x + c_3 y$$

or approximation by a paraboloid

$$\Phi(\vec{r}) \approx c_1 + c_2 x + c_3 y + c_4 x^2 + c_5 xy + c_6 y^2$$

In the case of the paraboloid fit, we can have smooth transitions between elements (which is not possible for the linear fit for non-trivial cases).

7.6.4 Variational Approach

The basic idea behind the variational approach is the minimization of the function

$$E = \int\int_G \left(\frac{1}{2} (\nabla\Phi)^2 + \frac{1}{2} a\Phi^2 + b\Phi \right) dxdy + \int_\Gamma \left(\frac{\alpha}{2} \Phi^2 + \beta\Phi \right) ds$$

where we have split up the functional into a volume contribution and a surface contribution (represented by the integral over G and Γ, respectively). This definition of E will be motivated later on.

The first step towards minimization of the functional is the determination of its variation δE:

$$\delta E = \int\int_G (\nabla\Phi\delta\nabla\Phi + a\Phi\delta\Phi + b\delta\Phi)\, dxdy + \int_\Gamma (a\Phi\delta\Phi + \beta\delta\Phi)\, ds$$

We recall Green's theorem from calculus

$$\int\int_G \nabla u\nabla v dxdy = -\int\int_G v\Delta u dxdy + \int_\Gamma \frac{\partial u}{\partial n} v ds$$

Of course we can use Green's theorem to modify the first term in δE (both u and v can be identified with Φ). We obtain

$$\delta E = \int\int_G (-\Delta\Phi + a\Phi + b)\, \delta\Phi dxdy + \int_\Gamma \left(a\Phi + \beta + \frac{\partial\Phi}{\partial n}\right) \delta\Phi ds = 0$$

We immediately recognize two different cases for a and b in $\Delta\Phi = a\Phi + b$:

- If we set $a = 0$, we obtain the Poisson equation ($\Delta\Phi = b$ with $b = -4\pi\rho$)
- If we set $b = 0$, we obtain the Helmholtz equation ($\Delta\Phi = a\Phi$ with $a = k^2$)

We consider the first term of the total energy

$$E = \sum_{\text{elements } j} \int\int_{G_j} \left((\nabla\Phi)^2 + a\Phi^2 + b\Phi\right) dxdy$$

which can rewrite as

$$E = \vec{\Phi}^t A\vec{\Phi} + \vec{b}\cdot\vec{\Phi}$$

Minimizing yields

$$\frac{\partial E}{\partial\Phi} = 0 \quad \Rightarrow \quad A\vec{\Phi} + \vec{b} = 0 \tag{7.49}$$

We are going to come back to this equation in sections 7.6.8 and 7.6.9.

7.6.5 Standard Form

We can transform any element j into the *standard form*. In two dimensions we have

$$x = x_1 + (x_2 - x_1)\xi + (x_3 - x_1)\eta$$
$$y = y_1 + (y_2 - y_1)\xi + (y_3 - y_1)\eta$$

with the new variables

$$\eta = \frac{(y-y_1)(x_2-x_1)-(x-x_1)(y_2-y_1)}{D}$$
$$\xi = \frac{(x-x_1)(y_3-y_1)-(y-y_1)(x_3-x_1)}{D}$$

where

$$D = (y_3-y_1)(x_2-x_1)-(x_3-x_1)(y_2-y_1)$$

These new coordinates ξ and η are the coordinates associated with the standard form. The standard form carries an index T. For an illustration of the idea behind standard forms see Fig. 7.9.

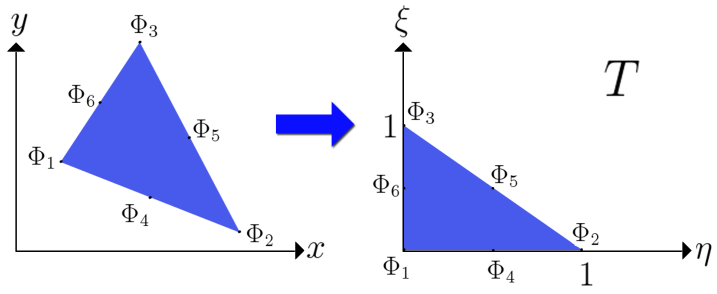


Figure 7.9: Transformation of an arbitrary element to the standard form (denoted by T) [20]

7.6.6 Coordinate Transformation

We can rewrite the derivatives with the new variables:

$$\frac{\partial\Phi}{\partial x} = \underbrace{\frac{\partial\Phi}{\partial\xi}}_{\Phi_\xi}\frac{\partial\xi}{\partial x} + \frac{\partial\Phi}{\partial\eta}\frac{\partial\eta}{\partial x} \quad , \quad \frac{\partial\Phi}{\partial y} = \underbrace{\frac{\partial\Phi}{\partial\xi}}_{\Phi_\eta}\frac{\partial\xi}{\partial y} + \frac{\partial\Phi}{\partial\eta}\frac{\partial\eta}{\partial y}$$

We can plug these derivatives into $\nabla\Phi$ and obtain

$$\nabla\Phi = \left(\frac{\partial\Phi}{\partial x}, \frac{\partial\Phi}{\partial y}\right) = \left(\frac{\partial\Phi}{\partial\xi}\frac{\partial\xi}{\partial x} + \frac{\partial\Phi}{\partial\eta}\frac{\partial\eta}{\partial x}, \frac{\partial\Phi}{\partial\xi}\frac{\partial\xi}{\partial y} + \frac{\partial\Phi}{\partial\eta}\frac{\partial\eta}{\partial y}\right)$$

Of course we need to know how the new variables depend on the previous ones:

$$\frac{\partial\xi}{\partial x} = \frac{y_3-y_1}{D} \quad , \quad \frac{\partial\xi}{\partial y} = -\frac{x_3-x_1}{D}$$
$$\frac{\partial\eta}{\partial x} = \frac{y_2-y_1}{D} \quad , \quad \frac{\partial\eta}{\partial y} = \frac{x_2-x_1}{D}$$

If we would like to know $|\nabla\Phi|^2$ we need to know $(\frac{\partial\Phi}{\partial x})^2$ and $(\frac{\partial\Phi}{\partial y})^2$. Of course, these are simply a combination of our previous results:

$$\begin{aligned}
\left(\frac{\partial\Phi}{\partial x}\right)^2 &= \left(\frac{\partial\Phi}{\partial\xi}\frac{\partial\xi}{\partial x}+\frac{\partial\Phi}{\partial\eta}\frac{\partial\eta}{\partial x}\right)^2 \\
&= \frac{(y_3-y_1)^2}{D^2}\Phi_\xi^2 - 2\frac{(y_3-y_1)(y_2-y_1)}{D^2}\Phi_\xi\Phi_\eta + \frac{(y_2-y_1)^2}{D^2}\Phi_\eta^2 \\
\left(\frac{\partial\Phi}{\partial y}\right)^2 &= \left(\frac{\partial\Phi}{\partial\xi}\frac{\partial\xi}{\partial y}+\frac{\partial\Phi}{\partial\eta}\frac{\partial\eta}{\partial y}\right)^2 \\
&= \frac{(x_3-x_1)^2}{D^2}\Phi_\xi^2 - 2\frac{(x_3-x_1)(x_2-x_1)}{D^2}\Phi_\xi\Phi_\eta + \frac{(x_2-x_1)^2}{D^2}\Phi_\eta^2
\end{aligned}$$

where we have used the notation

$$\Phi_\xi = \frac{\partial\Phi}{\partial\xi} \quad , \quad \Phi_\eta = \frac{\partial\Phi}{\partial\eta}$$

With these results, we know how to deal with derivatives. However, if we would like to use the new coordinates for integrals, we also have to compute the Jacobi matrix

$$J = \begin{pmatrix} \frac{\partial x}{\partial\xi} & \frac{\partial x}{\partial\eta} \\ \frac{\partial y}{\partial\xi} & \frac{\partial y}{\partial\eta} \end{pmatrix}$$

with determinant

$$det(J) = \frac{\partial x}{\partial\xi}\frac{\partial y}{\partial\eta} - \frac{\partial x}{\partial\eta}\frac{\partial y}{\partial\xi} = (x_2-x_1)(y_3-y_1) - (x_3-x_1)(y_2-y_1) = D$$

We also have to transform the integration area; let us call the transformed integration area T. With this transformation, we have all ingredients to carry out the integration with the new variables:

$$\int\int_{G_j} f(x,y)dxdy = \int\int_T \tilde{f}(\xi,\eta)det(J)d\xi d\eta$$

Let us now apply this to the example at hand instead of a general function $f(x,y)$:

$$\int\int_{G_j} \left(\Phi_x^2+\Phi_y^2\right)dxdy = \int\int_T \left(c_1\Phi_\xi^2 + 2c_2\Phi_\xi\Phi_\eta + c_3\Phi_\eta^2\right)d\eta d\xi \tag{7.50}$$

The coefficients are only calculated once for each element:

$$\begin{aligned}
c_1 &= \frac{(y_3-y_1)^2}{D} + \frac{(x_3-x_1)^2}{D} \\
c_2 &= 2\frac{(y_3-y_1)(y_2-y_1)}{D} + 2\frac{(x_3-x_1)(x_2-x_1)}{D} \\
c_3 &= \frac{(y_2-y_1)^2}{D} + \frac{(x_2-x_1)^2}{D}
\end{aligned} \tag{7.51}$$

7.6.7 Basis functions

The discretization in the finite elements method is done by selecting basis functions. As we have seen in the introductory example, the exact solution Φ of a PDE can always be written as a linear combination of infinitly many basis functions:

$$\Phi(x) = \sum_{i=1}^{\infty} a_i u_i(x)$$

with the basis functions u_i and their coefficients a_i. However, the computer neither has infinite memory nor infinite computing speed, so we have to select a finite number of basis functions:

$$\Phi_N(x) = \sum_{i=1}^{N} a_i u_i(x)$$

The basis functions u_i are known and defined for all elements, so we only have to compute their coefficients a_i.

In the one-dimensional case, the linear basis functions are hat functions (as in the introductory example in 7.6.2.2); these are 1 on the element on which they are defined and 0 on every other element.

A clearer example is the two-dimensional case, where we divide the domain into triangles (same size not required). As we have seen in the previous sections, these triangles can be transformed back to the so called standard element, whose corners are located at (0,0), (0,1) and (1,0). The basis functions are defined on this triangle and then transformed back to the original triangle for computation (see previous section on coordinate transformation).

Let us start out by defining the basis functions on the standard element. In the linear case, each basis function has value 1 in one corner and value 0 in the other two. We can thus think of the following three basis functions:

$$\begin{aligned} N_1 &= 1 - \xi - \eta \\ N_2 &= \xi \\ N_3 &= \eta \end{aligned}$$

where we have used the previously mentioned convention that the axes are called ξ and η to distinguish them from the axes of the original triangle (with axes x and y). A graphical representation of the linear basis functions is given in Fig. 7.10.

Instead of using linear basis functions, one may also use quadratic basis functions. One then has six points on each triangle. Again, each basis function has value 1 in one of the points and zero in all others, so we find the following six basis functions:

$$\begin{aligned} N_1 &= (1-\xi-\eta)(1-2\xi-2\eta) \\ N_2 &= \xi(2\xi-1) \\ N_3 &= \eta(2\eta-1) \\ N_4 &= 4\xi(1-\xi-\eta) \\ N_5 &= 4\xi\eta \\ N_6 &= 4\eta(1-\xi-\eta) \end{aligned} \tag{7.52}$$

An illustration of these basis functions is given in Fig. 7.11.

We have used two notations in this section; On the one hand we introduced the prefactors a_i in the decomposition of the field Φ with basis functions u_i; on the other, we just called the basis functions N_i (with prefactors φ_i). In the latter notation, we often use a short hand: we introduce the vectors $\vec{N}$ and $\vec{\varphi}$, whose ith components are the ith basis function (for $\vec{N}$) or the ith coefficient (for $\vec{\varphi}$). In two dimensions we thus write

$$\Phi(\xi,\eta) = \sum_{i=1}^{6} \varphi_i N_i(\xi,\eta) = \vec{\varphi}\vec{N}(\xi,\eta)$$

with

$$\vec{\varphi} = (\varphi_1, ..., \varphi_6) \quad , \quad \vec{N} = (N_1, ..., N_6)$$

The functions N_i are again those of Fig. 7.11. The basis functions also depend on the lattice (i.e. they look different for other lattices such as a square lattice).

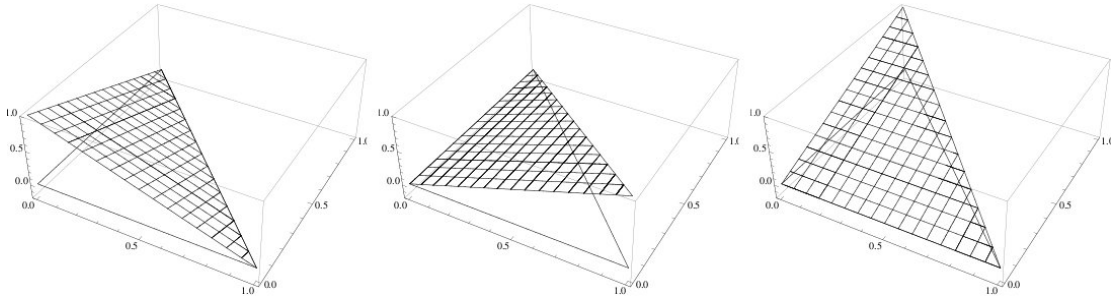


Figure 7.10: The three linear basis functions on the standard element [2]

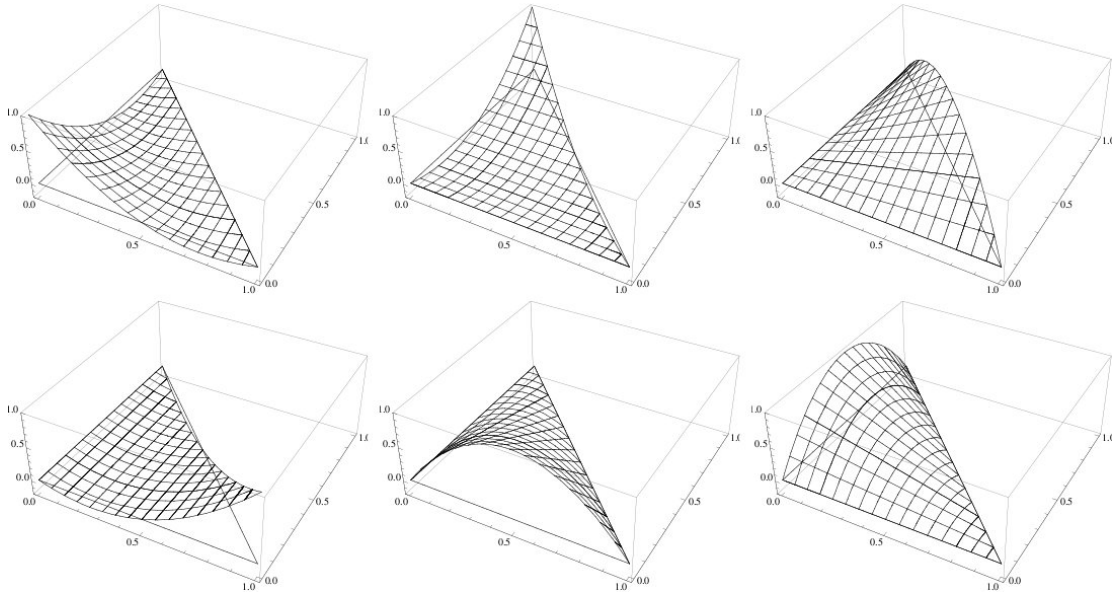


Figure 7.11: The six quadratic basis functions on the standard element [2]

7.6.8 Energy Integrals

Let us calculate the energy integrals on the standard element. We start with

$$\begin{aligned} I_1 &= \int\int_T \Phi_\xi^2 d\xi d\eta = \int\int_T \left(\vec{\varphi}\vec{N}_\xi(\xi,\eta)\right)^2 d\xi d\eta = \int\int_T \vec{\varphi}^{\,T}\vec{N}_\xi\vec{N}_\xi^t\vec{\varphi} d\xi d\eta \\ &= \vec{\varphi}^{\,T}\underbrace{\int\int_T \vec{N}_\xi\vec{N}_\xi^t d\xi d\eta}_{S_1}\vec{\varphi} \end{aligned}$$

and find in a similar fashion

$$\begin{aligned} I_2 &= \int\int_T \Phi_\xi\Phi_\eta d\xi d\eta = \vec{\varphi}^{\,T} S_2\vec{\varphi} \\ I_3 &= \int\int_T \Phi_\eta^2 d\xi d\eta = \vec{\varphi}^{\,T} S_3\vec{\varphi} \end{aligned}$$

where we have introduced the matrices S_1,S_2 and S_3 on the standard triangle. Let us weight the energy integrals I_i with the respective coefficients c_i from the coordinate transformation (see (7.51) in 7.6.6). We see in (7.50) that this is

$$\begin{aligned} c_1I_1 + c_2I_2 + c_3I_3 &= \int\int_T \left(c_1\Phi_\xi^2 + 2c_2\Phi_\xi\Phi_\eta + c_3\Phi_\eta^2\right) d\xi d\eta \\ &= \int\int_{G_j} \left(\nabla\Phi\right)^2 dxdy =: \vec{\varphi}^{\,T} S\vec{\varphi} \end{aligned}$$

This expression defines the rigidity matrix S for any element:

$$S = c_1 S_1 + c_2 S_2 + c_3 S_3 \tag{7.53}$$

Similarly, we define the mass matrix M:

$$\int\int_{G_j} a\Phi^2 dxdy = \int\int_T a\left(\vec{\varphi}\vec{N}(\xi,\eta)\right)^2 Dd\xi d\eta = \vec{\varphi}^{\,T} \underbrace{a\int\int_T \vec{N}\vec{N}^{\,T} Dd\xi d\eta}_{=:M} \vec{\varphi} =: \vec{\varphi}^{\,T} M \vec{\varphi}$$

We thus see that the energy integral has a contribution from both the mass and the rigidity matrix

$$E' = \sum_{\text{elements } j} \int\int_{G_j} \left((\nabla\Phi)^2 + a\Phi^2\right) dxdy = \sum_{\text{elements } j} \vec{\varphi}_j^{\,T} \left(S_j + M_j\right) \vec{\varphi}_j$$

So far we have not considered all terms of E yet (we still have to introduce the so-called field term which we shall do in section 7.6.9), so the current expression is only the "reduced energy integral" E' not yet the energy integral E as we know it.

If we introduce a matrix A_j for each element such that

$$A_j = S_j + M_j$$

we can rewrite the "reduced" energy integral

$$E' = \sum_{\text{elements } j} \vec{\varphi}_j^{\,T} \left(S_j + M_j\right) \vec{\varphi}_j = \sum_{\text{elements } j} \vec{\varphi}_j^{\,T} A_j \vec{\varphi}_j$$

or shorter

$$E' = \vec{\Phi}^{\,T} A \vec{\Phi}$$

where we have introduced the vector $\vec{\Phi}$, which is a vector with vector entries. The j-th element of $\vec{\Phi}$ is $\vec{\varphi}_j$, the coefficient vector for the element j. One often simply writes

$$\vec{\Phi} = (\vec{\varphi}_j) \tag{7.54}$$

Furthermore, we introduced the matrix A which is the more general form of the per-element A_j:

$$A = \otimes_j A_j = \otimes_j (S_j + M_j)$$

As previously mentioned, the expression we have obtained so far is still missing another term, the so-called field term.

7.6.9 Field Term

Let us consider the integral

$$\int\int_{G_j} b\Phi dxdy = \int\int_T b\vec{\varphi}_j \vec{N}(\xi,\eta) D_j d\xi d\eta = \vec{\varphi}_j \underbrace{b\int\int_T \vec{N}(\xi,\eta) D_j d\xi d\eta}_{\vec{b}_j} = \vec{b}_j \vec{\varphi}_j$$

Very similarly to before, we have thus found a second contribution (the field term) to the energy integral:

$$E = \vec{\Phi} A \vec{\Phi} + \vec{b}\vec{\Phi} \quad \text{with} \quad \vec{b} = (\vec{b}_j)$$

We now have found the complete expression for the energy integral, where the first part is the contribution found in the previous section and the second part is the field term.

7.6.10 Variational Approach

As we have shown in the previous sections, the solution of

$$\Delta\Phi = a\Phi + b$$

is a minimum of

$$E = \sum_{\text{elements } j} \int\int_{G_j} \left((\nabla\Phi)^2 + a\Phi^2 + b\Phi \right) dxdy$$

which can be brought into the form

$$E = \vec{\Phi} A \vec{\Phi} + \vec{b}\vec{\Phi}$$

When we minimize this expression, we obtain

$$\frac{\partial E}{\partial \Phi} = 0 \quad \Rightarrow \quad A\vec{\Phi} + \vec{b} = 0$$

This system consists of $6N$ linear equations, where N is the number of elements. The matrix A and the vector $\vec{b}$ depend only on the triangulation and on the basis functions. The unknowns are the coefficients $\vec{\Phi} = (\vec{\varphi}_i)$

So far, we have usually focused on one element j; obviously, we need to generalize this as we do not want an isolated solution on one element but a solution on the whole area. To this end, we need to connect the elements, which can be done with the boundary conditions. This gives rise to off-diagonal terms in the matrix A.

7.7 Time Dependent PDEs

Let us now consider non-elliptic equations; to satify the conditions as previously layed out, such equations must contain both derivatives with respect to time and space. Commonly known (parabolic) examples include the Schrödinger equation and the heat equation whose acquaintance we made earlier in this chapter.

7.7.1 Example: Heat Equation

We have already seen a simple example of a time dependent PDE (in 7.2.4), the heat equation :

$$\frac{\partial T}{\partial t}(\vec{x},t) = \frac{\kappa}{C\rho}\nabla^2 T(\vec{x},t) + \frac{1}{C\rho}W(\vec{x},t)$$

where $T(\vec{x},t)$ is the local temperature, C is the specific heat, ρ is the density (assumed to be homogeneous), κ is the thermal conductivity (assumed to be constant), and W is a term used to include external sources or sinks. In order to solve this equation, one can use the so-called "line method" which we shall employ in two dimensions. Then,

$$T(x_{ij}, t+\Delta t) = T(x_{ij},t) + \frac{\kappa\Delta t}{C\rho(\Delta x)^2}\tilde{T}_{ij}(t) + \frac{\Delta t}{C\rho}W(x_{ij},t)$$

where

$$\tilde{T}_{ij}(t) = T(x_{i+1,j},t) + T(x_{i-1,j},t) + T(x_{i,j+1},t) + T(x_{i,j-1},t) - 4T(x_{i,j},t)$$

When we stare at this equation for a bit, we recognize that there might be a problem if the prefactor $\frac{\kappa\Delta t}{C\rho(\Delta x)^2}$ of $\tilde{T}_{ij}(t)$ is $\frac{1}{4}$, as in that case we get a contribution $-T(x_{i,j},t)$ (from the last summand in $\tilde{T}_{ij}(t)$) which cancels out the leading contribution; even worse, if the prefactor is larger than $\frac{1}{4}$, the temperature will start to jump between positive and negative new values and become unstable. This behavior is characteristic for a parabolic equation. For stability's sake we thus need to avoid

$$\frac{\kappa\Delta t}{C\rho(\Delta x)^2} \geq \frac{1}{4}$$

7.7.2 Crank - Nicolson Method

The Crank-Nicolson method is an implicit algorithm, first proposed by John Crank and Phyllis Nicolson. When we look at

$$\begin{aligned} T(\vec{x},t+\Delta t) = T(\vec{x},t) \quad &+ \quad \frac{\kappa\Delta t}{2C\rho}\Big(\nabla^2 T(\vec{x},t) + \nabla^2 T(\vec{x},t+\Delta t)\Big) \\ &+ \quad \frac{\Delta t}{2C\rho}\Big(W(\vec{x},t) + W(\vec{x},t+\Delta t)\Big) \end{aligned}$$

we notice that $T(\vec{x}, t+\Delta t)$ appears on the RHS as well. This is suboptimal at best as we would like to only have this term appear on the LHS. As we are dealing with an implicit equation, we cannot solve it. We remedy this situation by discretizing the Laplace operator.

First, we define

$$\vec{T}(t) = (T(x_n, t)) \quad , \quad \vec{W}(t) = (W(x_n, t)) \quad , \quad n \in \{1, ..., L^2\}$$

(using the notation previously introduced in (7.54)) and the discretized Laplace operator O

$$OT(x_n, t) = \frac{\kappa \Delta t}{C\rho \Delta x^2}\Big(T(x_{n+1}, t)+T(x_{n-1}, t)+T(x_{n+L}, t)+T(x_{n-L}, t)-4T(x_n, t)\Big)$$

Note that we have included the prefactors of the heat equation in the definition. Then, the Crank-Nicolson formula becomes

$$T(\vec{x}, t+\Delta t) = T(\vec{x}, t)+\frac{1}{2}\Big(OT(\vec{x}, t)+OT(\vec{x}, t+\Delta t)\Big)+\frac{\Delta t}{2C\rho}\Big(W(\vec{x}, t)+W(\vec{x}, t+\Delta t)\Big)$$

where the factor of $\frac{1}{2}$ stems from the averaging over T. One then pulls this to the other side of the equation so that the RHS only contains values at time t and the LHS only contains values at time $(t + \Delta t)$:

$$(2 \cdot I - O)\vec{T}(t + \Delta t) = (2 \cdot I + O)\vec{T}(t) + \frac{\Delta t}{C\rho}\Big(\vec{W}(t) + \vec{W}(t + \Delta t)\Big)$$

where I is the unity operator. We can then only solve the equation by inverting the operator $(2 \cdot I - O)$; we already know how to carry out inversions as we have done it already for elliptic problems. The inverted operator is

$$B = (2 \cdot I - O)^{-1}$$

so that we can rewrite our equation:

$$\vec{T}(t + \Delta t) = B\left[(2 \cdot I + O)\vec{T}(t) + \frac{\Delta t}{C\rho}\Big(\vec{W}(t) + \vec{W}(t + \Delta t)\Big)\right]$$

This is the formal solution of the Crank-Nicolson method which is a solution to the heat equation. This method is of second order; one can go to higher orders by taking more terms in the Taylor expansion.

7.7.3 Wave Equation

In a next step, let us consider an example of a hyperbolic equation. For an equation to be hyperbolic, we need second derivatives with respect to time and

space with opposite signs; a popular example of a hyperbolic equation is the wave equation.

$$\frac{\partial^2 y}{\partial t^2} = c^2 \nabla^2 y \quad \text{with} \quad c = \sqrt{\frac{k}{\rho}}$$

The classical solution is to go to Fourier space (where we find frequencies/modes). We can use a finite difference scheme with finite time steps by discretizing the time derivatives:

$$\frac{y(x_n, t_{k+1}) + y(x_n, t_{k-1}) - 2y(x_n, t_k)}{\Delta t^2} \approx c^2 \nabla^2 y(x_n, t_k)$$

When we insert the discretized Laplacian and separate functions of (t_{k+1}), we find

$$\begin{aligned} y(x_n, t_{k+1}) = \quad & 2 \quad (1 - \lambda^2) y(x_n, t_k) - y(x_n, t_{k-1}) \\ & + \quad \lambda^2 \Big(y(x_{n+1}, t_k) + y(x_{n-1}, t_k) + y(x_{n+L}, t_k) + y(x_{n-L}, t_k) \Big) \end{aligned}$$

where we have introduced

$$\lambda = c \frac{\Delta t}{\Delta x} < \frac{1}{\sqrt{2}}$$

This corresponds to the cut off mode for wave lengths smaller than λ. In other words, the timestep Δt should be small enough to avoid propagations that are faster than one unit per time step. To paraphrase this once more, we are not allowed to leave the light cone (we cannot move faster than the seed propagation).

7.7.4 Navier-Stokes Equations

We have previously mentioned the Navier-Stokes equations as an example of a PDE. It is one of the most important equations in Fluid Dynamics; the branch of computational physics concerned with this field is called Computational Fluid Dynamics, or CFD for short.

The Navier-Stokes equations deal with constant energy (T const) so we still need to conserve mass and momentum. These two conservation laws can be found in the Navier-Stokes equations, and we shall try to find out how they enter and how the equations resemble Newton's equation.

First, let us write out the Navier-Stokes equations:

$$\frac{\partial \vec{v}}{\partial t} + (\vec{v}\vec{\nabla})\vec{v} = -\frac{1}{\rho}\vec{\nabla} p + \mu \nabla^2 \vec{v} \tag{7.55}$$

$$\vec{\nabla}\vec{v} = 0 \tag{7.56}$$

These are the equations of motion for an incompressible fluid. Let us look at the LHS of (7.55): We see that we find the full time derivative of the velocity field:

$$\underbrace{\frac{\partial \vec{v}}{\partial t}}_{A} + \underbrace{(\vec{v}\vec{\nabla})\vec{v}}_{B}$$

We directly make the connection to Newton's equation; instead of $F = ma$ we have brought the mass (in the form of the density ρ) over to the RHS,

$$F = ma \quad \Rightarrow \quad a = \tfrac{1}{m}F$$

use ρ instead of m and find

$$d_t v = -\frac{\text{const}}{\rho}F$$

Clearly, we still still have $a = d_t v$ on the left. There are two important contributions:

- A: We see that there is a direct contribution from the temporal change in the velocity field through the partial derivative
- B: The second contribution is the convective part. This term arises when the velocity field is convected with a fluid, i.e. when mass points take the velocity vectors with them. This contribution is non-linear.

On the RHS of (7.55), there are also two contributions:

$$\underbrace{-\frac{1}{\rho}\vec{\nabla}p}_{C} + \underbrace{\mu\nabla^2\vec{v}}_{D}$$

where μ is the viscosity.

- C: The first contribution stems from the force arising from a pressure gradient; this is the driving force.
- D: Furthermore, there is a second force which is particular to fluids: the viscosity term smooths out the equation. In fact, the absence of this term can cause numerical problems! Viscosity exists only in motion, and expresses the fluid's internal resistance to flow, it can be thought of a kind of "fluid friction". It arises when "layers" of fluids move at a different speed, so the viscosity arises from the shear stress between the layers (which will oppose any applied force)

So far, we have only talked about the velocity; clearly, aside from the velocity field $\vec{v}$ another field makes an appearance in the Navier-Stokes equations: the pressure

field p. We thus have 4 unknowns (v_x, v_y, v_z, p) but only three equations that we have discussed so far. The fourth equation is the condition

$$\vec{\nabla}\vec{v} = 0$$

which expresses the fact that we are looking at incompressible fluids. This equation is not just a "sidekick", it is central as it expresses mass conservation. We thus have four equations for four unknowns. Before continuing, we still need initial values and the values on the boundaries:

$$\begin{array}{lll} \vec{v}(\vec{x}, t_0) = \vec{V}_0(\vec{x}) & , & p(\vec{x}, t_0) = P_0(\vec{x}) \\ \vec{v}(\vec{x}, t)|_\Gamma = \vec{v}_0(t) & , & p(\vec{x}, t)|_\Gamma = p_0(t) \end{array}$$

There exist two limits of the Navier-Stokes equation:

- The large viscosity limit (or small system limit, small velocity limit, small Reynolds number limit): In this first case, we obtain the Stokes equation which describes a Stokes flow (typical for biological systems). It is a linear equation, and neglects term B, i.e. $(\vec{v}\vec{\nabla})\vec{v} \approx 0$
- For large Reynolds numbers, we cannot neglect B anymore but we can neglect D, i.e. $\mu\nabla^2\vec{v} \approx 0$. As previously hinted at, this is numerically suboptimal, as we get turbulence phenomena; these turbulences are due to the fact that we are neglecting viscosity which smoothens our system! We obtain the Euler equation, which is a pure nonlinear equation and impossible to solve numerically.

As the attentive reader might have guessed, we are not going to consider the second limit.

CFD has an impressive spectrum of methods to offer, in fact there are well over 100 methods to solve this kind of equations.

Instead of looking at every single one, let us pick just one, the penalty scheme, and try to solve our equations with it.

Penalty Scheme

We have to solve (7.55) and (7.56) simultaneously. The second equation will be treated as a "penalty" (nomen est omen).

We still have to discretize the derivatives in a smart way:

$$\frac{\vec{v}_{k+1} - \vec{v}_k}{\Delta t} = -\vec{\nabla}p_{k+1} - \mu\nabla^2\vec{v}_k - (\vec{v}_k \cdot \vec{\nabla})\vec{v}_k \tag{7.57}$$

When we apply $\vec{\nabla}$ on both sides, we obtain

$$\frac{\vec{\nabla}\vec{v}_{k+1} - \vec{\nabla}\vec{v}_k}{\Delta t} = -\nabla^2 p_{k+1} - \mu\nabla^2(\vec{\nabla}\vec{v}_k) - \vec{\nabla}(\vec{v}_k \cdot \vec{\nabla})\vec{v}_k$$

and we can use (7.56), or equivalently for the discretized velocity field

$$\vec{\nabla}\vec{v}_{k+1} = \vec{\nabla}\vec{v}_k = 0$$

to simplify the equation as follows

$$\underbrace{\frac{\vec{\nabla}\vec{v}_{k+1} - \vec{\nabla}\vec{v}_k}{\Delta t}}_{0} = -\nabla^2 p_{k+1} - \underbrace{\mu\nabla^2(\vec{\nabla}\vec{v}_k)}_{0} - \vec{\nabla}(\vec{v}_k \cdot \vec{\nabla})\vec{v}_k$$

so (7.57) has been reduced to

$$\nabla^2 p_{k+1} = -\vec{\nabla}\left((\vec{v}_k \cdot \vec{\nabla})\vec{v}_k\right)$$

Fortunately, the RHS does not depend on the next time step. Furthermore, we recognize the Poisson equation: We can find p_{k+1} from velocities at earlier time steps alone. To accomplish this, one needs boundary conditions for the pressure which one obtains by projecting the Navier-Stokes equations on the boundary, which must be done numerically.

7.7.5 Operator Splitting

We introduce an auxiliary variable field $\vec{v}^*$ (which is essentially just an intermediate step)

$$\frac{\vec{v}_{k+1} - \vec{v}^* + \vec{v}^* - \vec{v}_k}{\Delta t} = -\vec{\nabla}p_{k+1} - \mu\nabla^2\vec{v}_k - (\vec{v}_k \cdot \vec{\nabla})\vec{v}_k \tag{7.58}$$

Obviously, (7.58) can be split up into two equations

$$\begin{aligned} \frac{\vec{v}^* - \vec{v}_k}{\Delta t} &= -\mu\nabla^2\vec{v}_k - (\vec{v}_k \cdot \vec{\nabla})\vec{v}_k \\ \frac{\vec{v}_{k+1} - \vec{v}^*}{\Delta t} &= -\vec{\nabla}p_{k+1} \end{aligned}$$

We can determine $\vec{v}^*$ from the first equation and use it in the second one. If we apply $\vec{\nabla}$ to

$$\frac{\vec{v}_{k+1} - \vec{v}^*}{\Delta t} = -\vec{\nabla}p_{k+1}$$

we obtain

$$\nabla^2 p_{k+1} = \frac{\vec{\nabla}\vec{v}^*}{\Delta t}$$

If we project the normal vector $\vec{n}$ to the boundary, one obtains

$$\frac{\partial p_{k+1}}{\partial n} \equiv (\vec{n} \cdot \vec{\nabla})p_{k+1} = \frac{1}{\Delta t}\vec{n}(\vec{v}^* - \vec{v}_{k+1})$$

7.7.6 Spatial Discretization

We still have two "sets" of unknowns: on the one hand the velocity field $\vec{v}$ and on the other the pressure field p. A smart way to solve the equations is to place the velocities and pressures on a staggered lattice (MAC, marker and cell).

One places the pressures at the center of the cell (with lattice spacing h) and the velocity components at the centers of the sides of the cell (see Fig. 7.12).

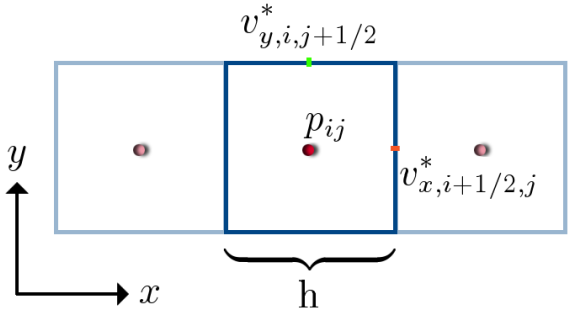


Figure 7.12: Placement of velocity components and pressure on the staggered lattice (MAC) [20]

The velocity in x direction (v_x) is found on the left and right sides of a given cell i, j (on the left $v_{x,i-1/2,j}$ and on the right $v_{x,i+1/2,j}$) while the velocity in y direction (v_y) is located on the top and bottom sides of a given cell i, j (on the top $v_{y,i,j+1/2}$, on the bottom $v_{y,i,j-1/2}$). This is illustrated in Fig. 7.13, where the color code indicates whether the entry corresponds to v_x (orange) or v_y (green). Furthermore, we see in the given examples of the velocities that the indices are adapted. The position in both directions is indicated with respect to the lattice spacing h, i.e. if we move from cell i, j over to the right by h, we arrive at the cell $(i+1), j$; the side separating the two positions obviously has the coordinates $(i+\frac{1}{2}), j$.

The pressure locations are referred to by their indices i, j. Let us consider the pressure gradient:

$$(\vec{\nabla} p)_{x,i+\frac{1}{2},j} = \frac{1}{h}(p_{i+1,j} - p_{i,j})$$

We see that the pressure gradient is defined on the sides of the cells (unlike the pressure itself!) as a function of the adjacent pressures $p_{i+1,j}$ and $p_{i,j}$.

If we consider the gradient of this expression, we are going to move on the lattice

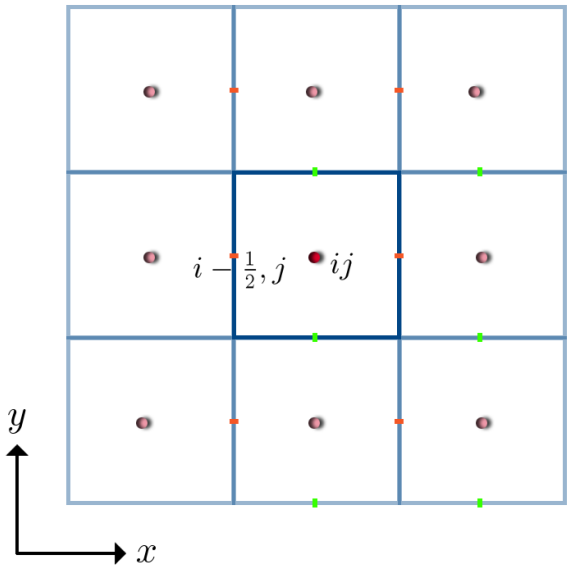


Figure 7.13: Placement of velocity and pressure on a lattice; in this illustration, two sample points are indicated with their coordinates. Green stubs indicate the location of v_x entries whereas orange stubs stand for v_y entries. [20]

again,

$$\nabla^2 p_{i,j} = \frac{1}{h^2}\left(p_{i+1,j} + p_{i-1,j} + p_{i,j+1} + p_{i,j-1} - 4p_{i,j}\right)$$

and we are in the center of a cell again. When we consider

$$\vec{\nabla}\vec{v}^*_{i,j} = \frac{1}{h}\left(v^*_{x,i+1/2,j} - v^*_{x,i-1/2,j} + v^*_{y,i,j+1/2} - v^*_{y,i,j-1/2}\right)$$

we see that this expression also comes to lie in the center of a cell (just like a pressure entry). Similarly, we have

$$\nabla^2 p_{k+1} = \frac{\vec{\nabla}\vec{v}^*}{\Delta t}$$

We can thus conclude that the pressure p_{k+1} is solved in the cell centers.

However, if we consider

$$\vec{v}_{k+1} = \vec{v}_k + \Delta t\left(-\vec{\nabla}p_{k+1} - \mu\nabla^2\vec{v}_k - (\vec{v}_k\cdot\vec{\nabla})\vec{v}_k\right)$$

we can conclude that the equation for the velocity components is solved on the sides.

Going back to the original equation, there is still one contribution that we have not mentioned so far: $(\vec{v}\vec{\nabla})\vec{v}$. If we take its first entry $(\vec{v}\vec{\nabla})v^x$, we have to discretize the following expression:

$$(\vec{v}\cdot\vec{\nabla})v^x = \underbrace{v^x\frac{\partial}{\partial x}v^x}_{\equiv E} + \underbrace{v^y\frac{\partial}{\partial y}v^x}_{\equiv F} \tag{7.59}$$

We discretize (7.59) step by step, starting out with the x coordinate (corresponding to E in (7.59)):

$$\left(v^x\frac{\partial v^x}{\partial x}\right)\Bigg|_{i+1/2,j} = \left(v^x_{i+3/2,j}\right)\cdot\frac{1}{2h}\left(v^x_{i+3/2,j} - v^x_{i-1/2,j}\right)$$

Of course we can do the very same for the y component (i.e. F in (7.59))

$$\left(v^y\frac{\partial v^x}{\partial y}\right)\Bigg|_{i+1/2,j} = \frac{1}{4}\left(v^y_{i,j+1/2} + v^y_{i,j-1/2} + v^y_{i+1,j+1/2} + v^y_{i+1,j-1/2}\right) \cdot\frac{1}{2h}\left(v^x_{i+1/2,j+1} - v^x_{i+1/2,j-1}\right)$$

The prefactor of $\frac{1}{4}$ stems from the fact that we are averaging over all four y components.

Let us summarize the position of the entries we have encountered on our odyssey around the lattice:

Element	Position in cell
p_{ij}	center
v_x	right & left side
v_y	top & bottom side
$(\vec{\nabla}p)$	sides
$\nabla^2 p$	center
$\vec{\nabla}\vec{v}$	center

7.8 Discrete Fluid Solver

There are several other ways of solving this kind of equations. One can go back to the starting point of the equations, the conservation of momentum. A fluid consists of molecules and one can derive the Navier-Stokes equations starting at the molecular level with the Boltzmann equation. One thus considers discrete systems that collide while the momentum is conserved.

Let us consider a couple of examples.

7.8.1 Lattice Gas Automata

One of these new methods are the lattice gas automata (LGA); they are based on the idea of cellular automata methods used to simulate fluid flows. One creates a lattice (as in Fig. 7.14) on which one places particles with certain momenta. When particles meet, one has to define the direction of the particles after the collision. In the collision, mass and momentum need to be conserved, so one arrives at the stochastic rules in Fig. 7.14 (in the case of a hexangular lattice; the $p = \frac{1}{2}$ indicates that both possibilities are equiprobable). One can prove that the continuum limit of these methods are the Navier-Stokes equations.

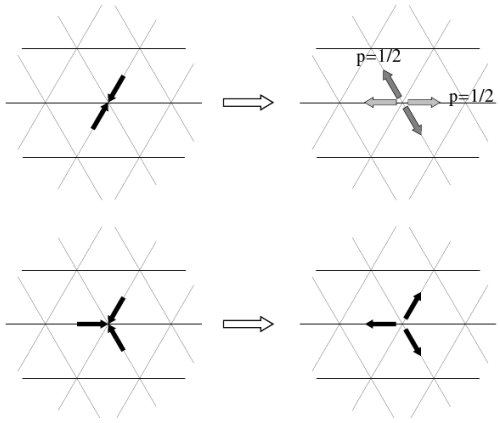


Figure 7.14: Stochastic rules for collisions (respecting mass and momentum conservation) [9]

Aside from academic interest, lattice methods have also found applications in the industry. Car manufactures are for instance interested in the calculation of the air flow around cars. Another less commercial application is illustrated in Fig. 7.15.

Lattice gas automata are particularly popular when one faces messy boundary conditions; obviously, this includes a lot of applications (e.g. the petroleum industry is very interested in how to force oil out of rocks).

7.8.2 Von Karman Street

Another example of computations that are made possible with these new methods is the von Karman street (mentioned earlier on already) which describes the velocity field of a fluid behind an obstacle. In Fig. 7.16, one recognizes several vertices. We recall that one particularity of the methods is that they are able to deal with messy boundary conditions; clearly, the von Karman street is an example of such boundary conditions.

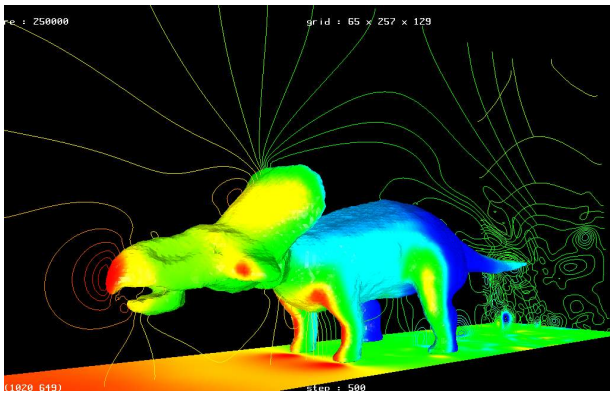


Figure 7.15: Air flow around a given object (dinosaur in this example) [15]

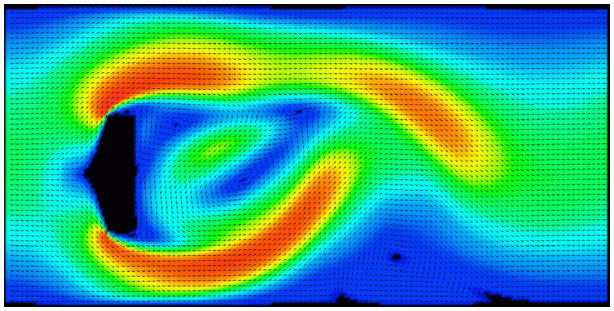


Figure 7.16: Von Karman street simulation [7]

7.8.3 Further Examples

Further examples include the following:

- Sedimentation
- Raising of a bubble
- Simulation of a free surface

Of course this list is not exhaustive - there are many more applications (in fact the range of applications is quite impressive).

Bibliography

[1] From corresponding Wikipedia article

[2] Of unknown source (if you know the source, please let us know!)

[3] Sofia University St. Kl. Ohridski (Physics Department)

[4] Flickr (from photographer Steve Wall)

[5] Semnitz: Cutaneous disorders of lower extremities

[6] University of Calgary (Department of Chemistry)

[7] University of Wuppertal (Physics Department)

[8] robodesign.ro

[9] Université de Genève, Class on Numerical Calculations

[10] Stauffer's book (Introduction to percolation theory)

[11] Herrmann H.J., Landau D.P., Stauffer D., New universality class for kinetic gelation, Phys. Rev. Lett. 49, 412-415 (1982)

[12] P. C. da Silva, U. L. Fulco, F. D. Nobre, L. R. da Silva, and L. S. Lucena, Recursive-Search Method for Ferromagnetic Ising Systems: Combination with a Finite-Size Scaling Approach, Brazilian Journal of Physics

[13] University of Cambridge, Dep. of Applied Mathematics and Theoretical Physics

[14] Trond Hjorteland's page

[15] Computational Fluid Dynamics Visualization Gallery

[16] Brazilian Journal of Physics

[17] Niel Tiggemann : Percolation

[18] Sociedade Brasileira de Física

[19] http://www.stephenwolfram.com/

[20] Figure made by Marco - Andrea Buchmann

[21] Adapted by Marco - Andrea Buchmann