2. Overview of important computational physics methods

Before we go on to discuss Monte Carlo simulations, I give an overview of important computational physics methods. This is in part because of general interest, but in part also because MC methods are (at least sometimes) used in connection to almost all main methods in computational physics. Hence knowing the basic methods helps understanding the subsequent examples of MC.

2.1. What is computational physics?

This is a non-trivial question. The simple definition would of course be any physics which utilizes computers in any form.

But this is not a very good definition. Most present-day measurements use computers, so by this definition even experiments would be computational physics.

The most common understanding of the term is that computational physics is attempting to theoretically solve or understand problems in physics with methods where computers play an essential and irreplaceable role.

Thus experiments are not computational physics, even though sometimes (think LHC) extreme amounts of number crunching may be necessary to get the experimental result.

Computational physics does with this wide definition include topics ranging from numerical integration, numerical solution of differential equations to simulation which attempt to follow the progress of a physical system as it happens in reality (example: star motion in a galaxy)

Since numerical methods of solving mathematical problems are very commonplace nowadays, many people tend to exclude at least the simplest numerical methods from the definition of computational physics.

What is clear, however, is that all aspects of **computer simulation** clearly are counted as computational physics. By computer simulations people mean using computers to mimic a real physical process. This can be either in the form of trying to simulate something as it happens in reality, but also using artificial dynamics in a manner which tries to give an answer corresponding to that in a real system.

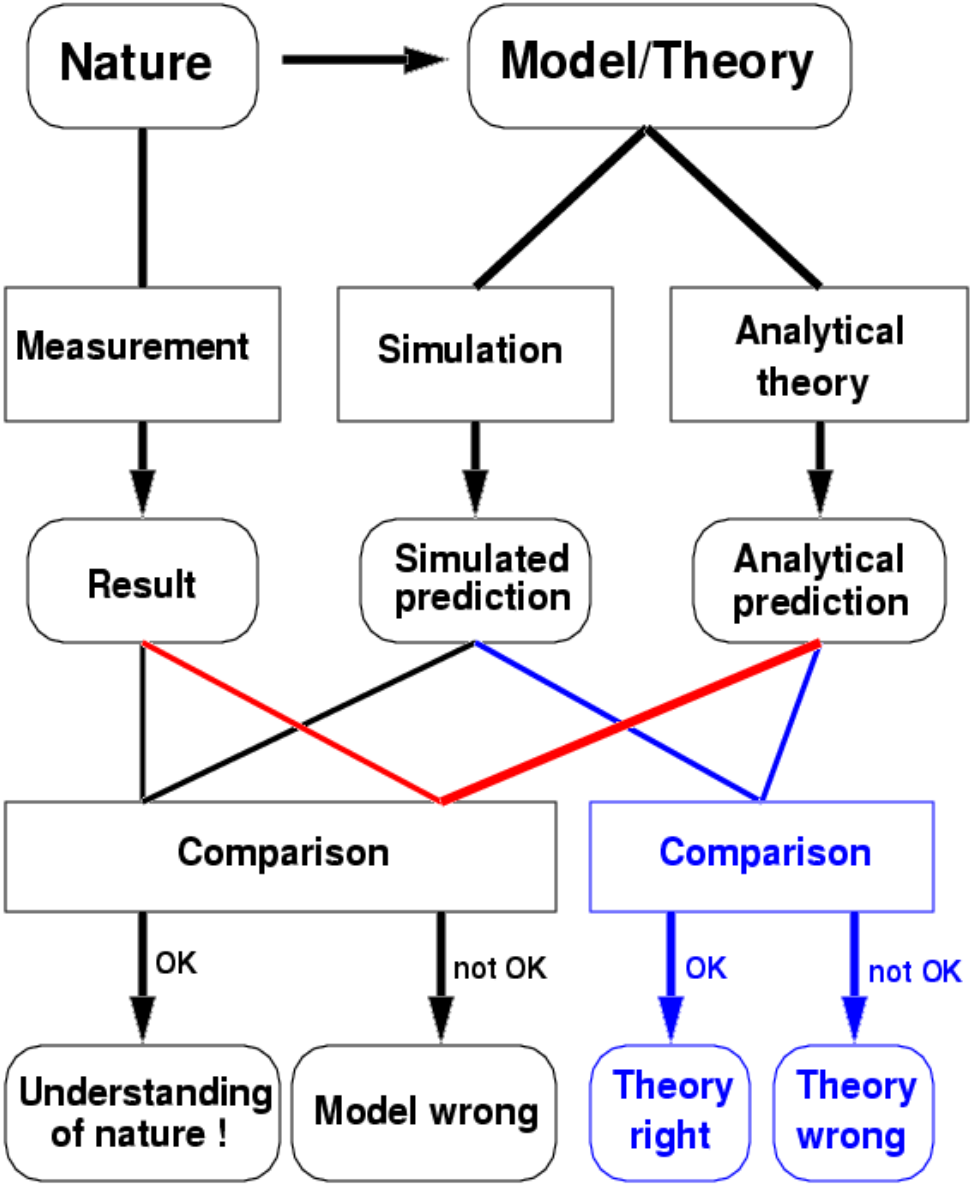
Another topic of frequent debate is whether computational physics should be counted equal to experimental and theoretical, or as a part of theoretical physics.

In practice, in many cases doing simulation work is more similar to doing experiments than theory. Typical aspects of experiments are rare, irrelevant events, equipment error, having to wait long for results, having to collect statistics, unexplained glitches and not understanding what is going on inside the systems that produces a given result. These can all be part of complex simulations as well, while they would never appear while doing analytical theory!

This in my personal opinion justifies placing computational methods on equal standing to theory an experiment as an approach to solve problems in physics.

But on the other hand, this is primarily a semantic and hence not all that important question.

One way to present the role of simulations in physics is as follows:


Nature
Model/Theory
Measurement
Simulation
Analytical theory
Result
Simulated prediction
Analytical prediction
Comparison
Comparison
OK
not OK
OK
not OK
Understanding of nature !
Model wrong
Theory right
Theory wrong

- Note that not only can simulations be used to compare to experiments, but also to test theories.

- In case the basic assumptions behind a theory and simulation are the same, but the theory uses more approximations, the simulation can sometimes provide a perfect test of the validity of the approximations in the theory.

Thus defining computational physics is somewhat arbitrary, but for working purposes we will understand it as non-experimental physics carried out with methods which require so high computational capacity that they can not be solved in reasonable time without computers. This rules out some of the simplest numerical techniques for e.g. solving the roots of an equation, since in most cases there exist clever analytical schemes (exact or approximative) to do such things using just pen and paper.

In the remainder of this section, we will review some common computational physics methods, attempting to give some flavour of what they do without going into almost any details. Since a huge number of these methods exist, the list is by no means exhaustive, and since many methods have heavy overlap, it is not even perfectly well-defined.

2.2. Numerical methods

[Numerical recipes, own knowledge]

The use of numerical methods is extremely wide-spread in the sciences today. A few examples of the most common methods in use are:

2.2.1. Interpolation

Since physics data often is in numerical form, and it is not known what function it exactly corresponds to, numerical interpolation is important for handling the data. One main reason is that using an interpolation scheme it is possible to treat a discrete set of data as if it were a continuous function, and then use the ordinary ways to handle continuous functions on it (derivation, integration etc.)

There are two dominating ways of doing this. One is direct linear interpolation, which is trivial, but not very accurate unless the data exist on a very dense grid. The other is various forms of spline interpolation. The most common is probably cubic spline interpolation. In this method a group of third-order polynomials is fit to the data, ensuring that at least the function and its derivative are continuous everywhere. It is almost as fast as linear interpolation, but much more accurate, and not hard to code (especially since subroutines are easily available in numerous text books, e.g. Numerical Recipes).

2.2.2. Parameter fitting

Often the motivation for doing this derives directly from the previous point. Although using interpolation on a discrete set of data enables doing continuous operations on the data, all of these still have to be also carried out numerically. If one wants to get from the numerical data to an analytical form, one can first derive or guess a functional form which should fit the data, leaving free parameters in it, and then fit these parameters to the data.

A one-dimensional example: given a discrete set of data

$$x_i, y_i, \delta y_i$$

where δy_i is the uncertainty of the data point y_i, one first has to select a function

$$y(x, a, b, c, d)$$

then fit the parameters a, b, c, d to the data so that (ideally) the function $y(x)$ will reproduce all the points y_i within the uncertainty δy_i.

2.2.3. Optimization

Parameter fitting is one variety of optimization. In there, one wants to optimize the function parameters $a, b, c, \ldots$ to reproduce the data as well as possible. This can also be stated as minimizing a function which describes the difference between the fitted function and the data. In general, there is a very wide range of methods which deal with minimizing and maximizing functions. The simplest are straightforward iteration routines, but even many of the most advanced simulation schemes in existence (e.g. molecular dynamics, Monte Carlo, genetic algorithms) are often used for minimization.

Some Monte Carlo minimization schemes are presented later during this course.

2.2.4. Numerical integration.

Here the problem is simply that we want to know the value of a definite integral

$$I(a) = \int_0^a y(x)dx$$

Used pretty much anytime analytical integration can not be carried out (and unfortunately to an increasing extent also when people are too lazy to even try to do the integral analytically).

This is also done very commonly when the function $y(x)$ does not exist in analytical form, but only as data points x_i, y_i.

2.2.5. Solving equations

The simple problem of finding the roots of an equation

$$f(x) = 0$$

is often carried out numerically. Most of you are probably familiar with Newton's method of doing this, and despite its simplicity it still sometimes is a good choice, even in multidimensions. But of course much better methods exist for many kinds of equation solving.

2.2.6. Solving differential equations

Another very important group of numerical methods are those used for solving differential equations, partial differential equations and sets of differential equations, because of the central role these play in many parts of physics.

An example could be solving the heat conduction equation

$$\frac{\partial T}{\partial t} = \kappa \nabla^2 T$$

where κ is the heat conduction coefficient. That is, we have an object of known shape, a heat source and heat sinks somewhere on it, and we want to know the time-dependent or steady-state temperature distribution on the object.

There is a spectacularly easy way of solving this problem for the steady-state case, $\partial T/\partial t = 0$. EXPLAINED DURING THE LECTURE.

This kind of finite-differencing methods are quite common for solving partial differential equations on computers. But often they are also very inefficient compared to more advanced methods (often related to FEM, see below).

2.2.7. Fourier transforms

Doing Fourier transformation of functions or data sets is very important in many branches of physics. In many cases the basic problem can be stated as: The real data has the form of a function in the time domain $h(t)$, and it can be sampled on discrete time intervals Δt in an experiment. We want to know the Fourier transform

$$H(f) = \int_{-\infty}^{\infty} h(t) e^{2\pi i f t} dt$$

of this function. This can obtained numerically directly, using the direct equation for the discrete Fourier transform. But this is very inefficient, it requires a number of operations $O(N^2)$, where N is the number of sampled points. Fortunately there is a smarter method called the Fast Fourier Transform (FFT) which reduces the computational complexity to $O(N \log_2 N)$.

2.2.8. Where to find out more about numerical methods?

Purely numerical methods are taught at the physics department on at least 2 courses, "Tieteellisen laskennan peruskurssi" and "- jatkokurssi". Hence this course will not delve deeper into them.

The by far most used books on the subject are those in the "Numerical Recipes" series. They are well written, and have a pragmatic viewpoint, emphasizing well-tested, easy-to-implement solutions. It is also available free online, http://www.ulib.org/webRoot/Books/Numerical_Recipes

But it has received some criticism on being unreliable (http://math.jpl.nasa.gov/nr/) and sometimes giving inefficient solutions, although most of the criticism is probably unwarranted or relevant only to the first edition (http://www.nr.com/bug-rebutt.html or http://www.ifir.edu.ar/ hnavone/docencia/apuntes/biblios/wnotnr.htm).

In my personal experience, Numerical Recipes is excellent to use when you need to quickly implement something which works, and do not care if it is top efficient or state-of-the-art. Of course, even then you have to test that the method actually works right in your problem. When, however, you need to find a routine as efficient as possible, or deal with a mission-critical part of your project, it is certainly better to delve deeper into the literature to see if something better than the Numerical Recipes solutions exist.

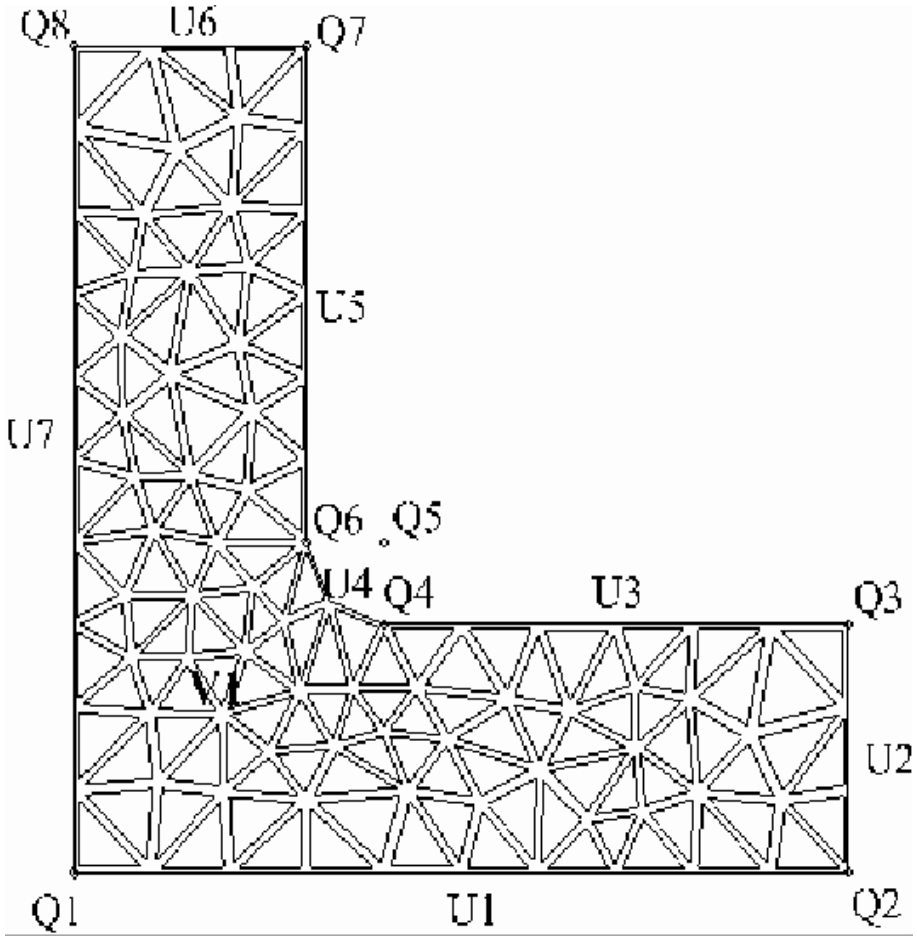
2.3. Finite element modeling

[http://care.seas.ucla.edu/niki/feminstr/introfem/introfem.html, private communication with P. Sofronis]

Finite element modeling (FEM) deals primarily with the analysis of elastic deformations of solids, but is also used in the modeling of e.g. fluid dynamics, electromagnetics and heat transfer. Elastic deformation means deformations which do not cause permanent change in a material. Imagine taking a long metal bar and bending it in one end, then watching how it vibrates and slowly returns to the starting point. FEM can model all of this.

As long as everything occurs within the elastic limit, and the characteristic lengths scales are much longer than interatomic distances, FEM works excellently at least in materials with well-known elastic properties. FEM can also be extended to handle plastic (permanent) deformation to some extent, but has great trouble in handling fracture.

The basic idea in FEM is divide an object into a 2D or 3D mesh of geometrical shapes (in 2D usually triangles):

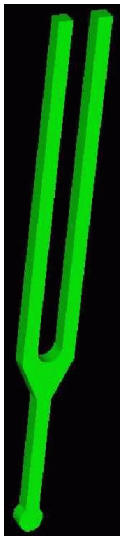


The points where the triangles intersect define completely the shape of the object. The FEM equations then allow applying a stress on one or more of the triangles, and calculating how the applied stress affects the shape of the entire object (the triangles are handled in a way such that the points where they meet always intersect,

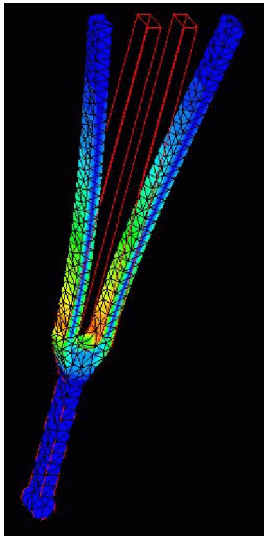
Usually the applied stress takes the form of an external boundary condition, since this of course is what usually happens in reality as well. A simple boundary condition for the figure above could e.g. be pressing the side U6 down while fixing the side U1, leaving all other sides open, and see how the object deforms.

An illustration of how the result of a deformation calculation can be: vibration modes in a tuning fork.

Original object:



One of the vibration modes:



Of course the vibrations are very strongly exaggerated in the figure, which is quite typical in illustrating FEM results.

FEM is widely used in the mechanical manufacturing industry, since it is quite possible to build up a FEM model of e.g. an entire car or airplane. There are several commercial FEM programs, one of the better known is Abaqus. But FEM is also used in the engineering and physics research communities in a wide range of problems. In Finland CSC is developing a multipurpose FEM code called ELMER.

FEM is not taught in the University of Helsinki, but certainly at the Helsinki University of Technology.

Their courses can almost certainly be accepted as part of your degree here, if it otherwise fits in your study line.

2.4. Molecular dynamics simulations

[see e.g. Allen-Tildesley]

Although the name "molecular dynamics" sounds very much like a topic specific to molecules, it is actually a very general method. Conceptually what MD does is something very simple: it describes the motion of a group of particles in space under the influence of forces acting between the particles, and possibly external forces as well.

The particles in question can be atoms, either as part of molecules or in a solid or liquid, molecules, microparticles in a fluid suspension, asteroids and planets or even parts of a galaxy. For the last two examples, the method is not usually called MD, but the basic algorithm is the same.

The simulation of the particle motion is achieved by solving the N-body equations of motion numerically. As you probably know, in classical mechanics only the 2-body problem can be easily handled analytically. The 3-body problem can in principle be handles analytically, but the solution is so impractical that virtually no-one uses it. For $N > 3$ an analytical solution is impossible, but MD gives a numerical solution. If the forces are known accurately, then the MD solution can be made arbitrarily accurate.

The basic MD algorithm can be illustrated as follows, in a slightly simplified manner (some tricks which speed up the solution have been left out):

The basic molecular dynamics algorithm

0. Set initial conditions $\boldsymbol{r_i}\,(t_0)$ and $\boldsymbol{v}_i(t_0)$

1. Solve equations of motion acting over a short time step Δt (predictor phase)

$$\boldsymbol{r_i}\,(t_n) \rightarrow \boldsymbol{r_i}^{pred}\,(t_{n+1})$$
$$\boldsymbol{v_i}\,(t_n) \rightarrow \boldsymbol{v_i}^{pred}(t_{n+1})$$

3. Calculate new forces $\boldsymbol{F}_i\,(\boldsymbol{r}_i^{pred})$

4. Solve equations of motion over the short time step Δt (corrector phase)

$$\boldsymbol{r_i}^{pred}\,(t_{n+1}) \rightarrow \boldsymbol{r_i}\,(t_{n+1})$$
$$\boldsymbol{v_i}^{pred}\,(t_{n+1}) \rightarrow \boldsymbol{v_i}(t_{n+1})$$

5. Set $t = t + \Delta t$, $n=n+1$

6. If $t < t_{\max}$, return to phase **1.**

7. Calculate final results and end simulation

The forces acting between atoms can either be derived classically or quantum mechanically. In the classical formalism, the algorithm can be (when cleverly written) virtually always made into an $O(N)$ form. The quantum mechanical formalisms are usually $O(N^3)$ or worse, although $O(N)$ methods are now starting to become common in a few applications. As a rule of thumb, classical methods can hence treat millions of atoms, while the quantum mechanical ones are limited to a few hundred.

The other main limitation of MD is the time scale. The time step Δt in the algorithm has an inherent limitation in that if it is too large, the numerical solution becomes completely unstable. For atoms this limit typically is of the order of 1 fs. This severely limits the time scales which can be handled by MD - to simulate one second one would need 10^{15} iterations of the basic MD loop, which is easily understood to be completely impossible for any system of real interest.

MD is taught in detail on the course "Introduction to atomistic simulations".

2.5. Genetic algorithms

[Deaven, Ho, Phys. Rev. Lett. 75 (1995) 288; Xiao and Williams, Chem. Phys. Lett. 215 (1993) 19]

Genetic algorithms is a group of algorithms which can be used for minimization in a wide range of problems, some of which are also used in physics.

The name derives from the idea in the algorithm to mimic Darwinian evolution. One forms a gene set which describes the real system of interest, then applies a process of natural selection of this to find the best adapted states. In algorithm form this can look e.g. as follows:

Simple form of a genetic algorithm:

0. Start.

Select a population size, e.g. 2. Generate DNA strings describing initial state in discrete form, e.g.

P1 = (1001:1010:1110:0110:0101:0011)
P2 = (1001:1010:1110:0100:1011:1110)

1. Mating and breeding.

From the parent state, exchange the string starting from a random bit:

C1 = (1001:1010:1110:0110:0|011:1110)
C2 = (1001:1010:1110:0100:1|101:0011)

to create two children.

2. Mutation. With a given probability μ/bit exchange the state of a bit (0 -> 1 or 1->0) for all bits in all individuals.

3. Natural selection. Select the 2 members in the new population (P1, P2, C1, C2) which are best adapted to the environment, and kill all the others.

4. If convergence has not been reached, return to stage **1.**

5. End

The "DNA" gene sequence codes the state of a system in binary form. If we, say, are looking at

minimizing the energy in a a molecules, and the system has 6 degrees of freedom: three distances and 3 angles, we could code the system state something like follows:

(4.5 Å,5.0 Å,9.0 Å, 120° , 100° ,60°)=(1001:1010:1110:0110:0101:0011)

where we have used crude 0.5 Å resolution in distances and 20° resolution in angles.

It is further necessary to have a criterion describing how well an individual is "adapted to the environment". For the molecular system example, this would probably simply be the system potential energy. In a fitting problem, it could be the χ^2 measure of the quality of the fit.

Note that despite its elegance, the genetic algorithm method by no means is good for minimization of all kinds of systems. For instance since the differences between the states tend to be big, it is extremely inefficient for finding the nearest local minimum efficiently. But in some systems with many deep local minima it can be absolutely the best for finding the global minimum [an example is: Morris, Phys. Rev. B 53 (1996) R1740].

GA for atoms is described on the course "Introduction to atomistic simulations".

2.6. Electronic structure calculations

The calculation of the electronic structure of atomic and molecular system is an extremely widespread and important part of modern physics, chemistry and biochemistry. Calculating the electron structure (i.e. solving the Schrödinger or Dirac equation) can be performed exactly analytically only for extremely small systems, more or less only for one-electron systems. Hence for any many-electron system numerical methods have to be used.

But even using numerical methods becomes difficult and very timeconsuming if no approximations are made. The main reason can be explained as follows. Given a set of nuclei at positions, one want to calculate the electronic ground state. The Schrödinger equation for a set of N nuclei M_n and I electrons m_i can be written as

$$\left(\overbrace{-\sum_{n=1}^{N} \frac{\hbar^2}{2M_n} \frac{\partial^2}{\partial \mathbf{r}_n^2}}^{T_n} \overbrace{-\sum_{n=1}^{I} \frac{\hbar^2}{2m_e} \frac{\partial^2}{\partial \mathbf{r}_i^2}}^{T_e} + \overbrace{\sum_{i=1}^{I} \sum_{j=1}^{I} \frac{e^2}{r_{ij}^2}}^{V_{ee}} + \overbrace{\sum_{n=1}^{N} \sum_{i=1}^{I} \frac{Z_n e^2}{r_{ni}^2}}^{V_{en}} \right.$$

$$+\overbrace{\sum_{n=1}^{N}\sum_{l=1}^{N}\frac{Z_n Z_l e^2}{r_{nl}^2}}^{V_{nn}}\Bigg)\Psi = E\Psi \tag{1}$$

Here T_n is the kinetic energy of nuclei, T_e the kinetic energy of electrons, V_{ee} the potential energies between electrons and electrons, V_{ne} the potential energies between nuclei and electrons, V_{nn} the potential energies between nuclei and nuclei. We will not here delve into the deeper meaning of these.

Even with modern computers it is practically impossible to handle the electron-electron term in almost any system of practical interest. So most electron-structure calculations approximate this term in a simpler form. The two main approaches are Hartree-Fock (HF) and density-functional theory (DFT).

A note on the terminology. Very frequently the terms "first principles" and *ab initio* are used to describe both HF and DFT methods, to signify that these methods are only based on the Schrödinger equation (which is assumed to be correct in the non-relativistic case) and use no empirical input anywhere. This is done so widely that many people tend to forget that "first principles" is a very general term which does not necessarily have anything to do with electron structure. Moreover, it

is debatable whether DFT should be called $ab\ initio$ at all, since most modern DFT methods use approximations which are used "because they seem to work well", i.e. are empirically motivated.

The third very common method for electron structure calculation is the tight-binding methods. These are similar to DFT, but only deal with the outermost electrons, and correct for the error in this with a pair potential. They contain fitted parameters so they are not $ab\ initio$.

Eletronic structure calculations are taught on "Introduction to electronic structure calculations".

2.7. Monte Carlo simulations/stochastic simulations

Monte Carlo (MC) or stochastic simulations is a common name which can be used for almost any simulation method which uses random numbers. In fact, most of the other simulation groups mentioned in this overview also often use random numbers, and hence could be considered MC, so there is considerable overlap in terminology.

But some simulation methods are conventionally considered genuine MC simulations, usually those where the basic algorithm itself relies on randomness. Since the remainder of this course deals with such methods, we will not say anything more here to avoid repetition.

2.8. Lattice QCD

[http://www.npac.syr.edu/copywrite/pcw/node34.html; http://members.tripod.com/ Igorlvanov/physics/hep-qcd.html]

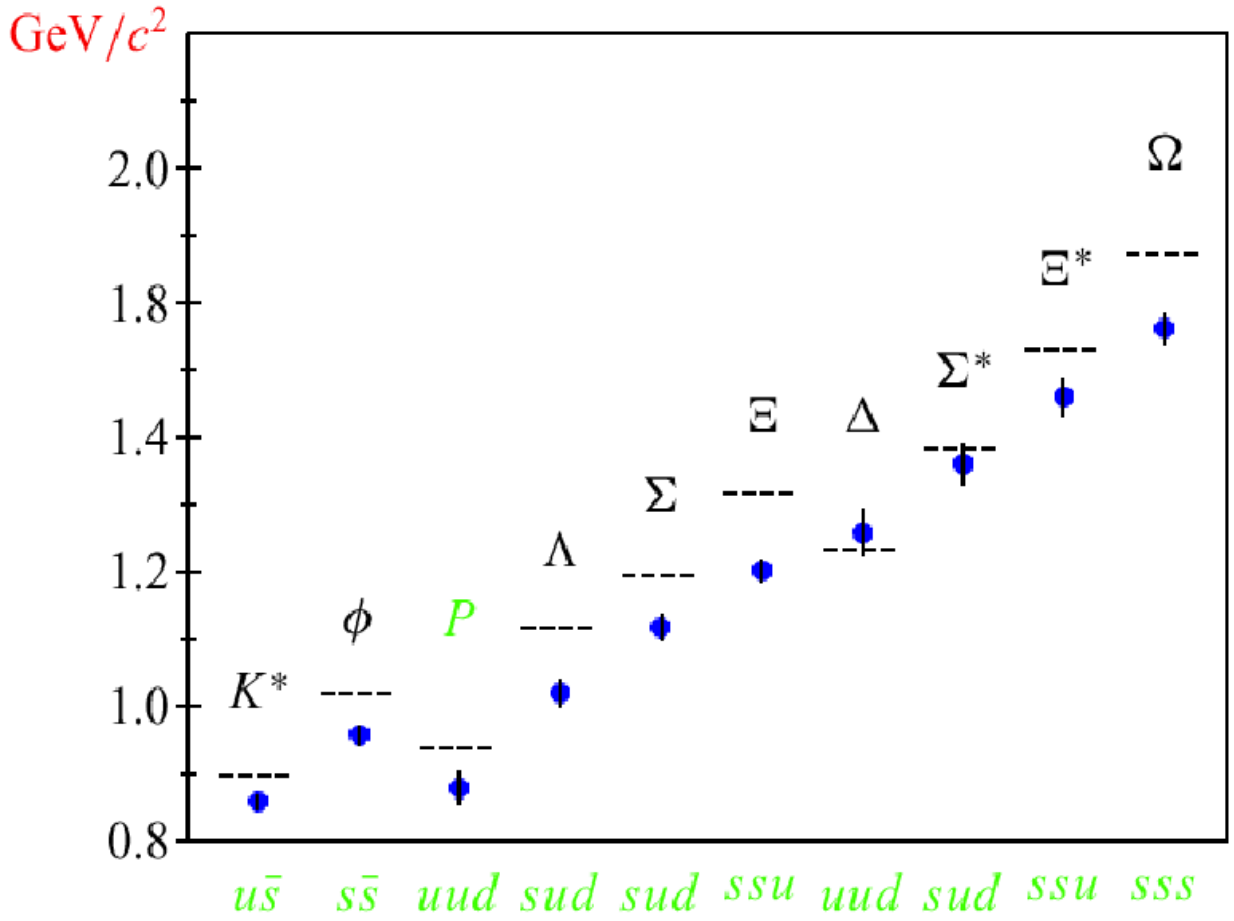
Quantum chromodynamics (QCD) is the theory which describes how gluons glue together quarks to form any heavier particles. The problem is that QCD is a nonlinear theory that is not analytically solvable. Moreover, perturbative approaches also do not work well because the QCD interaction is so strong. This has led to the introduction of non-perturbative approximations based on discretizing four-dimensional space-time onto a lattice of points, giving a theory called lattice QCD, which can be simulated on a computer.

Most of the work on lattice QCD has been directed towards deriving the masses (and other properties) of the large number of hadrons which have been observed experimentally. Also e.g. calculations of the conditions of matter in the early stages of the evolution of the universe and of hypothetical 4-quark systems have been performed (some of these in Helsinki in the group of Tony Green).

Lattice QCD requires enormous computing power. Already in 1995 (see web address below) QCD calculations were performed on lattices of size $24^3 \times 48$, which involves the numerical solution of a 21,233,664 dimensional integral. The only way of solving such an integral is by Monte Carlo methods.

Despite the difficulties, good progress has been made. Lattice QCD has been able to predict the masses of several basic hadrons to an accuracy of 10 % or so, a good feat considering that the QCD interaction parameters are not well known at all [http://www.physics.gla.ac.uk/ppt/ResInter/ProtonMass/]

Figure 3: Light hadron masses



Lattice QCD has been taught at our department by Tony Green. Since he is retired, the continuation of this teaching is now not entirely clear.

2.9. Cellular automata

[http://lslwww.epfl.ch/ moshes/ca_main.html; http://www.artificial-life.com/demos/automata/default.asp; http://www.brunel.ac.uk/depts/AI/alife/al ca.htm]

"Cellular automata are discrete dynamical systems whose behaviour is completely specified in terms of a local relation. A cellular automaton can be thought of as a stylized universe. Space is represented by a uniform grid, with each cell containing a few bits of data; time advances in discrete steps and the laws of the universe are expressed in, say, a small look-up table, through which at each step each cell computes its new state from that of its close neighbours. Thus, the system's laws are local and uniform."

They are important in pure computer science and mathematics for the basic theory of computation, but also in many applications in different fields of science. CAs have been applied to the study of general phenomenological aspects of the world, including communication, computation, construction, growth, reproduction, competition and evolution.

In physics they have been used to study a wide range of phenomena. They have a relation with partial differential equations; for instance the Dirac equation in 1+1 dimensions can be represented with a cellular automaton [Int. J. Theor. Phys. 20 (1981) 491]. They can also be used to e.g. model forest

fires [http://www.maths.usyd.edu.au:8000/u/magma/Examples/node8.html]; see the animation on the web page of this course.

At the end of this MC course we will discuss cellular automatons in greater detail.