Mid-term projects for

Advanced Introduction to C++, Scientific Computing and Machine Learning

Claudius Gros
2018/19

December 11, 2018

1 Formalities

The projects should be performed by groups of maximally three people, with the results being handed in as files

- A pdf-report of 5-8 pages containing an introduction to the subject and to the numerical methods used, a description of the programming effort (structure and core code snippets, but not all the code), and the presentation of the results. For the latter we expect both figures of simulated data and a statistical evaluation of any reasonable property that you choose.
- The documented code.
- Additional media files (optional) referred-to in the report. You may also generate gif files from series of images to visualize what is going on, in case your project involves time varying quantities.

Do not use libraries or external codes except for those used and presented in the lecture, if not explicitly asked by the project. We expect you to write the code simple and efficient on your own.

- The date to hand-in is: **January 25, 2019.**
- All groups present their results at the end of the term. If there is time, selected groups may present their results during the lecture hours, otherwise during the tutorials.

You are welcome to consult with the teaching assistant responsible for the project in case you need help.

1.1 Selection / Assignment

Every group must select a different project, no two groups may work on the same project. In need the projects will be assigned using a random number generator. The process will be in two steps.

- Every group must sent an email to Fabian Schubert (email on the webpage) till **10.12.2018**, indicating two or three preferred projects. Please indicate the order of your preferences (1,2,3). The result will be shown on the webpage (anonymised).

  If project has only a single contenter at this stage, it will be reserved.

- In the lecture, on **21.12.2018**, the final assignment will be organized. Groups attending will have preference in case of conflicting priorities.

  You are free to change your preference until then, telling Fabian personally. If you select a project not yet taken, it will be reserved.

2 Projects

You are welcome to propose your own project. This has to be down however well in advance.

- Contact your tutor, discuss content and workload.
- In case, submit an outline following the style of the projects below.

Your proposal will be included in this list (and reserved), if accepted. Proposals not included in this list cannot be selected.

2.1 Dynamic programming: Why non-rational behavior may be rational

(Oren Neumann)

Seemingly non-rational behavior may arise when maximizing the chance to survive in ecological models. Dynamic programming (1) is a method to evaluate optimal strategies in complex decision environments. The final goal is to show that a choice of survival strategy can violate transitivity when the available strategies are limited, see (2). Therefore, implement a class for a choice-making animal that is exposed to predation and starvation risks. The animal has a certain energy level as an internal state, being in the lowest state leads to starvation. The level of energy has an influence on the choice of the animal. Assume that its survival is certain if it is alive on the last day of winter (day 0). Then calculate its probability of survival for every previous winter day (day $-N$) using dynamic programming. Assume the animal always takes the best possible choice available.

Examine the strategies chosen at each energy level. Do they change in time or become constant? Show that the choice of strategy becomes intransitive when only two out of three strategies are available.

(1) Complex decisions made simple: a primer on stochastic dynamic programming

(2) Violations of transitivity under fitness maximization

2.2 Instance complexity, the Knapsak problem and human decision-making

(Oren Neumann)

Complexity theory, (1) and (2), deals with the question of how hard it is to solve a given problem, like the traveling salesman problem. Of key interest is the scaling of the computational effort with the number of parameters, e.g. with the number of cities to be visited. A problem is considered to be 'easy' when the scaling is polynomial, and 'hard' otherwise (non-polynomial hard or NP hard).

The so-called Knapsak problem (3) denotes the situation where one has to select a subset of items, characterized individually by a weight and a utility, such that utility is maximized and the cumulative weight minimized. It is assumed that this type of task is important for understanding the evolution of the human brain. For daily life situations, one is confronted with specific instances of the problem, which may be comparatively hard or easy to solve.

Implement an algorithm of your choice for the Knapsak problem, like dynamic programming, and study the instance complexity. Discuss the implication of the phase transition occurring as a function of weight and utility constraints (4).

(1) A Gentle Introduction to Computational Complexity Theory, and a little bit more

(2) A Gentle Introduction to Algorithm Complexity Analysis

(3) 0-1 Knapsack Problem

(4) Phase transition in the Knapsack problem

2.3 Roche instability and Saturn Rings

(Oren Neumann)

This project aims at simulating many particles under the effect of gravitation. For the sake of simplicity you may consider all problems in two dimensions.

Start by implementing a numerical integration method to solve the equations of motion of point mass particles in a gravitational potential of a star. Compare different integration methods, for instance the velocity verlet (1) and the Runge-Kutta fourth order method (2). Therefore, calculate up to serveral

million orbits and evaluate the accuracy in terms of energy conservation and to the accuracy to which the orbit retraces itself. Check dependency in terms of the time resolution Δt.

To investigate the Roche stability form a gravitational bound asteroid out of N (∼1000) rocks. Form the asteroid by adding a friction term, which is additional to the Newton equations of motion, such that the particles are contracted towards each other but experience a repulsion on short distance. The overall mass and density should correspond to a real asteroid (3).

Once the asteroid has formed, let it pass close to a planet, like Earth or Jupiter. What happens, if the asteroid passes within the Roche radius? Can you observe a ring to be formed?

For the last part of the project consider a large number of small objects in the orbit of a planet. Initialize the particles such that these form a ring like the Saturn ring (4). Let the system evolve, and check the stability of different orbits. Then add a shepherd moon and thereby induce gaps to the ring. Note: For this last part of the project you can neglect the gravitational influence between the small particles and only consider the effect of the planet and the moon on the particles.

(1) Velocity Verlet

(2) Runge-Kutta integration

(3) Asteroids

(4) Computer Simulation of Saturn's Ring Structure

2.4 Custom ring memory management

(Hendrik Wernecke)

Design a ring-memory, which dynamically manages the storage of elements, e. g. integers and integer arrays. Implement the ring-memory as a list (1) with forward and backward links, with each element of the ring being capable of storing data of a certain size, e. g. $N = 10$ integers per element. Futher, create a class for the memory manager, which should:

- initially create a ring of memory elements,
- store elements in the memory ring by finding a sufficiently large chunk of memory in consecutive elements of the ring,
- free the memory chunks, when stored content should be deleted,
- restructure memory elements to avoid memory fragmentation (2),
- add new memory elements to the ring, if and only if new content is too large to be stored in the existing ring.

Try to optimize the trade-off between memory fragmentation and re-allocation time. Measure the performance of the memory manager, when using 50% of your machine's total RAM by filling the memory up to a certain percentage, e. g. 70%, and then randomly delete and add arrays. Monitor the number of operations and the execution time (3) needed.

(1) Data type list

(2) Memory fragmentation

(3) Measuring time in `C++` accurately

2.5 Blockchain transactions in a distributed network

(Hendrik Wernecke)

Implement a small network of agents and a simplified blockchain (1), which records the transactions executed by the agents. Use a cryptographic hash function (2) of your choice (can be imported from external library) to realize the proof-of-work concept (3). The transactions of an agent shall be digitally signed and verifiable for any other agent in the network by using pairs of public/private keys (4). The goal is to perform blocks of transactions between the agents in real time (i. e. in parallel) and to measure the effort and success of agents. The procedure of transaction should follow these steps:

- Nodes send their desired transactions to the network of agents.

- Other nodes verify the transaction with the public key method.
- When a block of transactions is full the network decides by proof-of-work which node appends the current block to the chain. (A reward can be given here.)
- The updated blockchain is distributed in the network.

Try to attack the blockchain by manipulating past blocks.

(1) Blockchain algorithm

(2) Algorithmic hash functions

(3) Proof-of-work concept

(4) Public key method

2.6 Pseudo random number generators (pRNG)

(Hendrik Wernecke)

Computational methods such as the Monte-Carlo algorithm or cryptographical applications require random numbers satisfying the properties of statistical randomness (1). Therefore, one employs so-called random number generators (RNG) (2). Unfortunately, truely random number are scarce, so one is restricted to pseudo-RNG (3).

Implement at a cryptographically safe (4) pRNG of your choice (uniform distributed random numbers, write it on your own), and test it for statistical randomness. Further, perform a statistical analysis on the built-in `C++` pRNG

- `std::rand` from `<cstdlib>` and
- `std::uniform_int_distribution` from `<random>` (requires `C++11`).

Try to identify weak spots of one of the algorithms and simulate an attack by predicting the next number given a sequence of random numbers. Evaluate the success of your attacks statistically.

(1) Statistical randomness

(2) Overview of random number generators

(3) Pseudo random number generators

(4) Cryptographically secure pRNG

2.7 Arbitrary-precision arithmetic

(Anton Motornenko)

Create a type that would be suitable for arbitrary precision calculations for both integers and floats. Store those numbers as arrays where each element of array will contain one (or two, or three, or ... what is the optimal number?) digit. Define algebraic operations for it so you can add, subtract, divide, and multiply those objects as normal numbers (i.e. by just typing `a+b, a-b, a/b, a*b`). Implement elementary mathematical functions for those objects `pow`, e^a, $\sin(a)$, $\arcsin(a)$, $\log(a)$ (this one may be tricky), use Taylor series expansions for it. Create a conversion function to and from standard C++ data types (`int, float, double, long double`). Provide also a nice output of numbers in exponential form (scientific notation).

Try to calculate first 10000 digits of π. Calculate 1000!. Examine performance of the calculations, study how computation time changes while increasing accuracy. Compare with computation times for C++ standard types.

Note: for $\log(a)$ divide a by 2 n times until $a/2^n < 2$ then $\log(a) = \log(a/2^n) + n\log(a)$ where both terms can be calculated by Taylor series expansion. For more advanced algorithms take a look in (3).

(1) Donald Knuth "The Art of Computer Programming", Chapter 4.3

(2) Double factorial formula for π: $\frac{\pi}{2} = \sum_{k=0}^{\infty} \frac{k!}{(2k+1)!!} = \sum_{k=0}^{\infty} \frac{2^k k!^2}{(2k+1)!}$

(3) Elementary Functions: Algorithms and Implementation

2.8 Microscopic simulations of ideal gas (molecular dynamics)

(Anton Motornenko)

Implement a simplified molecular dynamics model (no inter-particle potentials, only elastic collisions). Simulate microscopical dynamics of the one-component gas and study it's thermodynamical properties.

- Initialize a cubic box of a size L;
- Put there a number ($N \sim 10^2$) of particles of the same type and mass (m) with some arbitrary spatial distribution;
- Define boundary conditions of the box: zero (particles are reflected from the boundary) or periodic (if a particle reaches one side of the box it disappears there and appears on the opposite side with the same momenta);
- Particles should have some initial kinetic energy (the total kinetic energy of particles is $E_{\rm kin}^{\rm tot}$ and total momentum is $\sum_N \vec{p} = 0$).
- Define particle interaction in the following way: particles travel along straight-line trajectories until the distance d to the closest particle is $d \leq D$ where D is obtained from interaction cross-section σ as $D = \sqrt{\sigma/\pi}$ (2), (3). Collision occurs in the center of mass frame. After collision particles are scattered at some random angle with some random momentum transfer, then particles move along new straight-line trajectory with the new momenta. Energy-momentum conservation law is fulfilled during the collision.

With the settled microscopic behavior you can study macroscopical properties of the system. Wait some time τ until system reaches thermodynamical equilibrium.

- Calculate single particle energy distribution by averaging over all particles in the box and by the time or/and by the different boxes from ensemble. Compare distribution with Boltzmann distribution, try to extract system temperature by fitting the distribution.
- Calculate the pressure of the system, which is defined as $P = 1/3 \sum_N \langle \vec{p_i} \vec{v_i} \rangle$ (where the summation runs over all particles in the box and $\langle \dots \rangle$ means time/ensemble averaging). Check if ideal gas law is satisfied ($P = nT$ where $n = N/V$ is particle density and T is the temperature).
- Check what changes with the change of collision cross-section. What changes for different initializations of the system (but keeping the total kinetic energy and density of particles the same), e.g. initialize particles in some part of the box, initialize particles with the same momenta, initialize particles with some arbitrary distribution in momenta, initialize particles with some arbitrary spatial distribution.

Suggested scales:

- particles: nucleons (protons and neutrons);
- $\sigma = 40$ mb (4 fm^2) $\Rightarrow D \approx 1.2$ fm;
- densities $n = 0.05 - 0.5$ fm^{-3};
- $E_{\rm kin} = 10 - 50 \cdot N$ MeV ;
- N = 200;
- $L = (N/n)^{1/3}$.

(1) Molecular dynamics

(2) see Sec. 3.3.1 for the description of the collision criteria

(3) see Sec. A. 1), 3), 5), 6); B. 2); for some details regarding molecular dynamics for many-hadron physics.

2.9 Equilibrium properties of neutron stars

(Elias Roland Most)

Neutron stars are among the densest objects in the known universe and are made of matter at densities and pressures too extreme to be produced by any experiment on earth (1). At the same time having a mass of roughly that of the sun they only have sizes comparable to that of Frankfurt. One largely unknown property of neutron stars is their composition in the interior as the behaviour of nuclear matter at such densities is poorly understood. Since this uncertainty is also reflected in the masses and radii of neutron stars studying equilibrium solution for various nuclear physics input is one of the key tools in constraining neutron stars but also nuclear physics alike.

In this project your task will be to study some properties of isolated neutron stars in equilibrium, which are relevant for gravitational wave observation. That is, you will be computing a mass-radius relation and the tidal deformabilities, which govern how much the star can be deformed in a binary system and what imprint it will subsequently have on the gravitational waveform of a neutron star merger.

An equilibrium solution for a spherically symmetric neutron star is described by its pressure p, energy density ϵ, radius R and mass $M = m_r(R)$. The tidal deformability Λ is proportional to the Love number $k_2 = k_2(y)$ for $y = \frac{R\beta(R)}{H(R)}$, where $\beta(R)$ and and $H(R)$ are auxiliary functions.

To compute these quantities you will need to solve the *Tolmann-Oppenheimer-Volkoff* (TOV) equations for p and m_r and the equations for tidal deformability for the auxiliaries H and β (2):

$$\frac{dp}{dr} = -(\epsilon + p)\frac{m_r + 4\pi r^3 p}{r(r-2m_r)}, \tag{1}$$

$$\frac{dm_r}{dr} = 4\pi r^2 \epsilon. \tag{2}$$

$$\frac{dH}{dr} = \beta \tag{3}$$

$$\begin{aligned}\frac{d\beta}{dr} = {} & 2\left(1-2\frac{m_r}{r}\right)^{-1} H\left\{-2\pi\left[5\epsilon + 9p + \frac{\mathrm{d}\epsilon}{\mathrm{d}p}(\epsilon+p)\right]\right. \\ & \left. + \frac{3}{r^2} + 2\left(1-2\frac{m_r}{r}\right)^{-1}\left(\frac{m_r}{r^2} + 4\pi r p\right)^2\right\} \\ & + \frac{2\beta}{r}\left(1-2\frac{m_r}{r}\right)^{-1}\left\{-1 + \frac{m_r}{r} + 2\pi r^2(\epsilon - p)\right\}.\end{aligned} \tag{4}$$

The equations can be solved most efficiently using an adaptive Runge-Kutta scheme, like Dormand-Prince, starting from the center of the star. Starting from a central pressure $p = p_c$, $m_r = 0$ integrate the equations out to the radius R of the star, which is defined as $p(R) = 0$. The initial conditions for H and β are obtained from the series expansion $H(r) = r^2$ and $\beta(r) = 2r$ as $r \to 0$.

In order to close the equation you need to specify an equation of state $p = p(\rho)$. For simplicity you may take $p = K\rho^\Gamma$. In summary, your task will be

1. Implement a C++ object 'TOV' that describes a TOV neutron star and provides all the necessary routines to compute all the quantities described above.
2. Implement an adaptive Runge-Kutta scheme (e.g. Dormand-Prince) to solve the set of coupled ODEs given above
3. Implement the computation of the tidal deformability k_2 , see eq. (14) in PRD 81, 123016 (2010)).
4. For a large variety of K and Γ compute sequences of TOV solutions, i.e. for 200 different values of p_c.
5. Analyse what are good values for K and Γ to get subluminal sound speeds $c_s < 1$ and maximum neutron star masses $M_{\max} \sim 2M_\odot$.
6. Analyse the dependence of $k_2(M)$ and also study the behaviour of $\Lambda = \frac{2}{3}k_2 R^5$. What happens if an upper bound $\Lambda < 700$ is introduced?

(1) Introduction to neutron stars

(2) Tidal deformabilities and TOV equation (PRD 81, 123016 (2010))

(3) Neutron Stars for Undergraduates

2.10 Second order phase transitions within Ginzburg-Landau theory

(Elias Roland Most)

This project aims to simulate a second order phase transition. As a thermodynamic system it can be described terms of the free energy, which in Ginzburg-Landau theory is given by (1)

$$\mathcal{F}([\psi(\mathbf{x})], T) = F_0(T) + \int_{\mathbf{R}^m} \left[A(|\nabla\psi|)^2 + a(T - T_c)\psi^2 + a_0\psi^4 \right] \mathrm{d}^m x. \tag{5}$$

Here $\psi(x)$ is an order parameter that determines which phase the system is in at every point x and T_c is the critical temperature below which two stable phases can exist. To simplify the calculation, we will use a dimensionless, rescaled form of this equation

$$\mathcal{F}([\psi(\mathbf{x})], T) = \int_{\mathbf{R}^m} \left[\frac{1}{2}(|\nabla\psi|)^2 - \frac{1}{2}\psi^2 + \frac{1}{4}\psi^4 \right] \mathrm{d}^m x. \tag{6}$$

Performing a simple variation of this functional you can show that (please derive the equation)

$$-\frac{\delta\mathcal{F}(\psi)}{\delta\psi} = \Delta\psi + \psi - \psi^3. \tag{7}$$

Hence you can see that the two phases are represented by $\psi = \pm 1$, which is an exact minimum of the free energy functional.

Your task will now be to study possible solutions numerically, by finding configurations that minimize the energy functional (6). To do so, you will have to setup an initial configuration before the phase transition and a final configuration after the phase transition. The transition itself will be parameterised by N intermediate states, that will be computed by iteratively solving (7) using the gradient-descent like *string method* proposed in section IV of (2). This amounts to solving the simple ODE

$$\dot{\psi} = -\frac{\delta\mathcal{F}(\psi)}{\delta\psi} \tag{8}$$

for every intermediate state plus an additional reparametrisation after every Runge-Kutta iteration, which will require you to perform a cubic spline interpolation of all N intermediate states. (You may use a library such as the *GNU Scientific Library* (3) for this.) Your tasks will then be

1. Write a C++ code that implements the string method outlined in section IV. J. Chem. Phys. 126, 164103 (2007). This includes a 4th order Runge-Kutta method with an additional reparametrisation.

2. Set up a 1D grid setting half of the values to 1 the other half to -1, that is to two distinct phases as the initial configurate and -1 everywhere as the final configuration. Use a suitable number of ~ 100 intermediate states. Then apply the code to this problem and compute the solution. Additionally, solve the variation equation (7) analytically (the 1D case has a unique solution) and show that your code reproduces it during the transition.

3. Now setup a 2D problem with various phases on the domain and again compute the phase transition to a uniform $\psi = -1$ configuration via minimum energy energy intermediate states closest to your input. What do you find?

(1) Ginzburg-Landau Theory

(2) Simplified String Method

(3) GNU Scientific Library

2.11 A simple SIMD parallel datatype

(Elias Roland Most)

Modern compute architectures allow for different levels of parallelisms. Besides running a programme on multiple cores of the same CPU, e.g. by using thread level parallelism, it is also possible to exploit single-instruction-multiple-data (SIMD) data parallelism.

This means that instead of performing an operation such as $d = a * b + c$ it is possible to promote the operands to vectors and execute $\mathbf{d} = \mathbf{a} * \mathbf{b} + \mathbf{c}$ on up to 16 `float`/ 8 `double` elements at the same time.

This crucial feature for high performance computing applications can sometimes be efficiently exploited by the compiler, especially if explicit Fortran compute kernels are used, but often fails in the presence of more complicated C++ object code, where the interdependencies of the data are less obvious to the compiler. In order to ensure the full utilisation of vector capabilities it is possible to explicitly write the code using *vector intrinsics*, which directly translate to single assembly instructions.

These are C functions that are called on arrays of the underlying datatype, e.g. `float[16]` on the most recent Intel Skylake architecture, and ensure that full vectorisation is achieved.

As an example consider a simple addition of the consecutively stored 4 doubles in arrays `a` and `b` using AVX intrinsics:

```
#include <immintrin.h>
__m256d add(__m256d &a, __m256d &b){
        return _mm256_add_pd(a,b);
}
//This adds four consecutive double elements from a and b
//and stores them in c
void load_and_add( double *a, double *b, double *c){

  __m256d av, bv; // __m256d is a vector register datatype,
                          //  roughly this a double[4] array,
                          // and can be accessed using av[0] etc.

   __m256d av = _mm256_load_pd(a); // Load four consecutive doubles
   __m256d bv = _mm256_load_pd(b);

   __m256d cv =  add(av,bv); // Perform the addition

   _mm256_store_pd(cv);  // Write the result back to the address at c
}
```

Since writing intrinsics by hand for every operation is very cumbersome, your task will be to implement a simple data parallel datatype based on intrinsics, where operators such as $+, -, /, *, +=, -=, ...$ internally translate to intrinsics. You should also write wrapper functions that map your datatype to SIMD versions of the most common math functions (you can use SIMD math functions from a library). Since intrinsics are highly hardware dependent limit yourself to the older Intel architectures and implement SSE and AVX variants allowing for the use of 4 `float` and 8 `float` elements respectively. You will find most of the information needed for this in the references.

Using this parallel datatype implement one algorithm of your choice, that you believe will benefit from vectorisation using the two parallel datatypes you have written. Examples here would be a Runge-Kutta or Gauss-Legendre integration. Compare this with a scalar implementation that you try to autovectorise using the compiler.

Present a study with different data batch sizes and show the benefit of vectorisation on different architectures, i.e. compare the scalar, SSE and AVX versions.

(1) Introduction to SIMD vectorization with SSE

(2) Crunching numbers with AVX

(3) CppCon18 Talk: Compute More in Less Time Using C++ Simd Wrapper Libraries

(4) SIMD math functions

2.12 Expression templates for tensor equations

(Elias Roland Most)

Expressing equations governing physical systems can be a cumbersome and error prone endeavour. In the field of general relativity large systems of partial differential equations govern the evolution of exotic objects such as black holes. With the advent of gravitational wave detections from compact binary mergers, the need for accurate gravitational wave forms requires the numerical evaluation of large tensorial quantities (see (1), (2) and (3)), such as

$$R^{\lambda}{}_{\sigma\mu\nu} = \partial_{\mu}\Gamma^{\lambda}{}_{\nu\sigma} - \partial_{\nu}\Gamma^{\lambda}{}_{\mu\sigma} + \Gamma^{\lambda}{}_{\mu\kappa}\Gamma^{\kappa}{}_{\nu\sigma} - \Gamma^{\lambda}{}_{\nu\kappa}\Gamma^{\kappa}{}_{\mu\sigma}.$$

This expression contains many implied summations of the same indices that need to be efficiently mapped to modern programming languages. While classical approaches consisted in using computer algebra systems such as Maple and Mathematica, it is equally possible to implement them directly in C++ by means of *expression templates* (1). These were one of the first applications to template meta-programming and features the use of an expression tree to map equations to recursively evaluated expressions. This recursive unfolding can be performed at compile time with zero overhead or impact at run-time letting the compiler generate efficient code for simple expressions. To give an example the idea is to map an expression $\lambda A_{ij} + B_{ij}$ to an object

`VectSum< ScalarMult< rank2<low,low> > , rank2<low,low> >`.

Your task will be to implement a simple expression template framework that can perform contractions, scalar multiplications and additions. The expressions should keep track of the rank and dimensions of the tensor, e.g. ensure that $C_{ij} = A^k_i B_{kj}$ is correctly treated as a rank 2 tensor, and should refuse compilation if invalid expressions are entered, e.g. $C_{ij} + B_i$ would be illegal.

Using this framework implement several expression, shorter and longer ones, with also explicitly written out loops for the short expression to evaluate performance. If you need some inspiration for lengthy but physical meaningful applications check out the first pages of (4).

(1) General Relativity

(2) Introduction to Ricci Calculus

(3) Introduction to Expression Templates

(4) Complicated expressions from numerical relativity

2.13 Message Passing Interface (MPI)

(Thomas Mertz)

The task of this project is to get familiar with the Message Passing Interface (MPI) (1), which implements efficient communication between processes on parallel computing architectures. The standard (1) is rather well-written and nicely indexed, however, you can find a simple MPI tutorial at (2). OpenMPI (a particular open source implementation of the MPI standard) is installed on the ITP computers, which you can access via your accounts. If you want to install OpenMP on your home machine, see (6).

1. Warm-up. Get started by writing small programs that send messages from one process to another (`MPI_Send`), from one process to all others (`MPI_Bcast`), from all processes to all others. The sending process should be configurable via a command line argument. Use `MPI_Gather` to parallelize a scalar product of two vectors and send the partial results to one process. Then, modify your code such that the results are sent to all processes (`MPI_Allgather`). Verify the correctness of your results through suitable text output, e.g. "processor N received MESSAGE from processor M".

2. Broadcasting. Implement your own broadcast method `mybroadcast`, sending data from one process to all others, by using `MPI_Send` and `MPI_Recv`. This works well for small networks, however, MPI is built to scale well to large networks of thousands of processors. Benchmark your implementation vs. MPI's `Bcast`. How could your implementation be improved to better utilize the network, where all processors are connected with each other?

3. Ring-Topology. Simulate a bidirectional ring-topology by implementing your own `send_left` and `send_right` methods, which allows each process to only send a message to its left and right neighbors. Assign random (but unique) identifiers to each process, such that at compile time the process with the largest identifier is not known.

4. Leader Algorithm. Motivate why the finding of consensus, e.g. in the election of a leader, is difficult on a ring topology. Implement the Hirschberg-Sinclair algorithm (3) to find a leader (process with the largest unique identifyer UID) on your ring.

5. Distributed Sorting Algorithm. We want to implement the merge sort algorithm on a pool of processors. First, implement the serial merge sort algorithm. Then, use MPI to build a parallel version. You can use (4) and (5) as a guide. Benchmark the serial and parallel implementation using random arrays and analyze the scaling with respect to the length of the array and the number of processors. How large is the performance overhead due to communication? On an ideal system with instantaneous communication, what scaling would your algorithm achieve with the number of processors. On which additional parameters does this scaling depend?

(1) MPI Standard

(2) MPI Tutorial

(3) Hirschberg-Sinclair algorithm

(4) Parallel Mergesort

(5) Parallel Merge

(6) Open MPI

2.14 Harmonic Waves – Linear Algebra

(Thomas Mertz)

We want to determine the vibration modes of a guitar string as a function of the tension T and density μ. In a first order approximation the dynamics of the string is governed by a wave equation, see (4),

$$\frac{\partial^2 y}{\partial x^2} = \frac{\mu}{T}\frac{\partial^2 y}{\partial t^2}, \tag{9}$$

where y is the deviation from the equilibrium position at point x. In order to tackle the problem numerically, discretize space and use the first-order approximation for the second derivative. The Hamiltonian is then represented by a matrix in the discrete space. The problem then reduces to a set of coupled second-order ordinary differential equations, which can be decoupled by diagonalizing the second derivative operator. The subsequent solution of the decoupled second-order equations is readily done analytically.

Implement a numerical solution to the eigenvalue problem

$$Av = \lambda v, \tag{10}$$

where $A \in \mathbb{C}^{n\times n}$, $v \in \mathbb{C}^n$ and $\lambda \in \mathbb{C}$.

You should implement matrix and vector types and a matrix-vector multiplication.

1. Power method. The task of finding the largest eigenvalue of a matrix A can be solved iteratively by realizing that given a random vector v_0 represented in the unknown eigenbasis $\{w|Aw = aw, a \in \mathbb{C}\}$ of A

$$v_0 = \sum_i c_i w_i, \tag{11}$$

successive multiplication of A with v_0 yields

$$v_n = A^n v_0 = \sum_i c_i a_i w_i. \tag{12}$$

If the largest eigenvalue is non-degenerate, the relative overlap $\langle v_n, w\rangle / \langle v_n, v_n\rangle$ will approach one for large enough n, where w is the eigenvector associated to the largest eigenvalue. In practice, one renormalizes the vector v_n in every iteration, i.e. after every multiplication with A.

2. Finding the smallest eigenvalue. Since the power method can only find the largest eigenvalue, we have to do a little more work to obtain the ground state. Let $A' = A - \mu I$, where $\mu \in \mathbb{R}$ is larger than the largest eigenvalue of A. A' has the same eigenvectors as A, but its eigenvalues are shifted by μ, such that the smallest eigenvalue is $\lambda_0' = \lambda_0 - \mu$ and $|\lambda_0'| > |\lambda_i|\ \forall i > 0$. Therefore, the smallest eigenvalue can be found by two successive power iterations, first with A to find μ, then with A'.

3. System of linear equations. Implement the Gaussian elimination algorithm to find the solution to a system of linear equations.

4. Matrix inverse. The inverse of a matrix can be obtained by solving n separate systems of equations. Use your Gaussian elimination method to obtain the inverse of a matrix.

5. Inverse power method. The power method can only compute the largest eigenvalue. Using a small modification, arbitrary eigenvalues can be computed. Instead of through multiplication with the matrix A, the vector v_n in each iteration is defined as

$$v_n = \frac{(A-\mu I)^{-1} v_{n-1}}{\| (A-\mu I)^{-1} v_{n-1} \|}, \tag{13}$$

where μ is an initial guess for an eigenvalue. The algorithm will converge to the eigenvector associated to the eigenvalue closest to μ. Is it worthwhile to compute the inverse instead of solving a system of linear equations? Analyze the performance of the two approaches.

(1) Power iteration

(2) Gaussian elimination

(3) Inverse power iteration

(4) String vibration

2.15 Poisson Equation – Stencil Codes

(Thomas Mertz)

The task of this project is to analyze optimization strategies for so-called "stencil" codes (1). These types of codes appear in all types of relaxation algorithms, where an update operation is performed on a subset of grid points. The shape of the subset of relevant grid points is called a "stencil", which is moved across the grid in each iteration. Here, we are interested in the solution of the electrostatic Poisson equation in two spatial dimensions

$$\Delta\phi = \rho, \tag{14}$$

where $\phi(x)$ is the electrostatic potential and $\rho(x)$ is the charge density. In this case we study both the Dirichlet problem with given potential and the von Neumann problem, where the density on a boundary is known.

Solve the Poisson equation for the surroundings of a homogeneously charged ring ($\rho_{\text{boundary}} = \rho_0$) and an H-shaped metal plate at constant potential ϕ_0.

0. Why iterative methods. Explain, by analyzing the performance in terms of e.g. the number of floating point instructions, why a direct solution of the Laplace equation, for example by Gaussian elimination, is practically infeasible for large systems. Compare with the scaling of iterative methods.

1. Jacobi method. Implement the Jacobi method (2) to solve the Poisson equation iteratively. Plot the execution speed (site updates per second) as a function of the grid size.

2. Gauß-Seidel method. Implement the Gauß-Seidel method (3) as a more memory-efficient version to solve the Poisson equation iteratively. Compare with the Jacobi method. How does the convergence speed compare?

3. Optimization. Optimize your Jacobi code to attenuate the memory bandwidth bottleneck by introducing blocks of fixed size $b \leq L$ (L = linear system size) in one spatial direction, which are updated one after the other. Investigate the performance upon decreasing b starting from L. Make use of the shared memory architecture of modern processors by parallelizing the block updates with OpenMP (4).

4. Performance analysis. Find a condition for optimal performance based on the memory bandwidth bottleneck of the Jacobi algorithm. (Hint: why are site updates faster on smaller blocks?) Taking measurements with your code for different system sizes, can you estimate the size of your processor's L3 cache? How does the performance scale with the number of threads? Investigate for different system and block sizes.

(1) Definition: Stencil code

(2) Jacobi Method

(3) Gauß-Seidel Method

(4) Some information on OpenMP, including minimal examples

2.16 Two dimensional XY Model

(Fabian Schubert)

Introduction. In the two dimensional (2D) XY model, classical spins of unit length in a two dimensional lattice can rotate in the plane of a lattice. The Hamiltonian for nearest neighbor interaction is given by

$$H = 1/2 \sum_{i,j} H_{i,j} \tag{15}$$

$$H_{i,j} = -J\left[\cos(\theta_{i,j} - \theta_{i+1,j}) + \cos(\theta_{i,j} - \theta_{i-1,j}) + \cos(\theta_{i,j} - \theta_{i,j+1}) + \cos(\theta_{i,j} - \theta_{i,j-1})\right] \tag{16}$$

where $\theta_{i,j}$ are angles denoting the spin orientation at the lattice sites and J is coupling constant.

The state of minimal energy is given by a perfect alignment of all spins and it is invariant under a global rotation. Interestingly, in two spatial dimensions, the Mermin-Wagner theorem shows rigorously that such a continuous symmetry cannot be broken spontaneously at any finite temperature. Thus the XY-model cannot have an ordered phase at low temperature like the Ising model does.

Kosterlitz-Thouless Transition. Even though no phase transition occurs in the model, it can be shown that there exist two "phases"

- A low-temperature-phase, where most spins are aligned and the correlation-function decays with a power law.
- An unordered high-temperature-phase where the correlation-function decays exponentially.
- Additionally, the correlation length also diverges as a power law when the critical temperature is approached from above.
- This should be verified in the numerical simulation.

Vortices. The existence of the Kosterlitz-Thouless (KT) transition is linked to be "vortices". A vortex is a topologically stable spin-formation, which lowers the correlation of the system. In the low temperature phase, there are few vortices, closely coupled as vortex-antivortex pairs. In the high temperature phase, many vortices are present.

What to Do in the Project.

1. Instantiate a two-dimensional array of size $N \times N$, where N is e.g. 1000. This array holds the spin angles on the lattice.
2. Subsequently update spin angles at randomly chosen sites using the Metropolis-Hastings algorithm (1) given some temperature T. Use periodic boundary conditions for next neighbors at the borders of the lattice.
3. Investigate the KT transition by estimating the spin angle correlation $C_\theta = \langle \cos((\theta_{i,j} - \theta_{k,l}) \rangle$ as a function of distance d and Temperature. Fit both a power-law and an exponential function to $C_\theta(d)$, sweeping through T from low to high temperatures and discuss the resulting goodness of fit as a function of T.
4. In the high temperature regime, the exponential function should provide a good fit to the correlation length. If your exponential function is of the form $\exp(-d/l)$, the fitting parameter l defines the *correlation length* fo your system. Plot this parameter as a function of T. You should see a divergence at a critical temperature T_c.
5. Implement a method that finds spin vortices in your lattice. For this you can make use of the fact that for a single vortex
$$\oint \nabla\theta dl = 2\pi k \ , k = \pm 1, 2, 3, ... \tag{17}$$
where k is the winding number of the vortex. Plot the number of vortices in the system as a function of temperature. Can you relate the shape of this function to the value of T_c?

(1) Metropolis Algorithm

(2) Classical XY Model

(3) The Metropolis-Hastings algorithm

(4) The XY Model

2.17 Finding Structure in Time – The Simple Recurrent Network

(Fabian Schubert)

Introduction. Many applications of machine learning are concerned with the extraction of patterns within static pieces of data, such as images, where the information is presented to the system in parallel. However, structure in data can also be found in the time domain, the most obvious example being spoken

or written language. Research on time-based learning algorithms exists since the 1980s and is an ongoing field of research.

One approach to this problem is the notion of recurrence: artificial neural networks feeding signals back into them self can store and process information provided in a time-dependent manner. One implementation of such a network was proposed by (1). This simple recurrent network (SRN) consists of four different types of units:

- input units: they provide the time dependent signal. This signal is fed forward to
- hidden units: these constitute the recurrent part of the network by passing on an exact copy of their state to
- context units, which project the signal back to the hidden units.
- output units: they receive a feed-forward signal from the hidden units.

How exactly such a network processes time dependent input should depend on the task that the learning mechanism is supposed to achieve. If the learning task can be formulated as an error-minimization problem, one can apply a modified backpropagation algorithm called *backpropagation through time* (BPTT) to the network structure described above. The idea of the BPTT algorithm is to unfold recurrent causal relations in the network into a feed-forward network structure, which can then be learned with a normal backpropagation algorithm. A detailed description of this method can be found in (2).

Using the Elman SRN to Predict Time Dependent Signals. One way to make the Elman SRN "learn" something about the time dependent input that it receives is to train it on the prediction of the time course of the signal. If the network is updated in discrete steps, a straightforward task would be to make the output generate the value that the input takes in the next step.

What to Do in the Project. A mathematical formulation of the network's functionality is given by

$$x_i^H(t) = \tanh\left(\sum_{j=1}^{N_I} W_{ij}^{HI} x_j^I(t) + \sum_{k=1}^{N_H} W_{ik}^{HC} x_k^H(t-1) - \theta_i^H\right) \tag{18}$$

$$x_i^O(t) = \sum_{j=1}^{N_H} W_{ij}^{OH} x_j^H(t) - \theta_i^O \tag{19}$$

where W_{ij}^{HI} is a matrix holding connections from input nodes to those in the hidden layer, W_{ij}^{HC} connects context to hidden units and W_{ij}^{OH} are the connections from hidden to output units. N_I and N_H denote the number of input and hidden units. θ_i^H and θ_i^O are input biases of hidden and output nodes. Note that the copying from hidden to context units is implicitly expressed by projecting the state of the hidden units in the previous time step $\mathbf{x}^H(t-1)$ back onto their current state. The nonlinear hyperbolic tangent in the hidden units is used to keep the activity of nodes bounded, in this case within $[-1, 1]$.

1. Implement the system described by (18) and (19). As a starting point, you may initialize the weights randomly.

2. The predictive task of the learning algorithm should be to match $\mathbf{x}^O(t)$ to $\mathbf{x}^I(t+1)$. The mean squared error thus can be expressed as $\mathrm{E}(t) = \frac{1}{N_I}\sum_{i=1}^{N_I}\left(x_i^O(t-1) - x_i^I(t)\right)^2$. Note that for this task the size of the output layer N_O should—obviously—match the size of the input layer N_I. To begin with, your network should have a single input and output node. The first input sequence that the network should be able to learn is a completely deterministic, alternating signal: $x^I(t) = 2\mathrm{mod}(t, 2) - 1$.

   Implement the BPTT algorithm in your network, acting on all weight matrices in (18) and (19). Additionally, biases in the hidden and output nodes should be treated as nodes generating a constant input of -1 and having weights given by the strength of the biases. This will allow you to include them into the backpropagation algorithm.

3. Once this simple signal can be predicted by the network, you can try training more complex sequences. For example, you could use a sine wave discretized in time, or even some chaotic (but deterministic) map. Investigate how well the network can learn to predict these signals depending on the size of the hidden layer and/or the depth of your backpropagation algorithm.

4. The XOR sequence is an example of a partially deterministic sequence, consisting of zeros and ones. Every third entry of the sequence is given by the result of an XOR operation on the two elements preceding it. These two elements are drawn randomly from $\{0, 1\}$. While the average squared error of the prediction for the randomly generated numbers is expected to be $0.5^2 = 0.25$, it should drop significantly for the elements given by the XOR operation.

5. Let's assume that your network has learned to precisely predict the next element of some deterministic sequence. In theory, you could then use this prediction as the next input, and the network will then generate the sequence *autonomously* (learning should be turned off at this point). Try this with a periodic signal, as well as a chaotic sequence. For how long does the network's output match the actual signal?

(1) Finding Structure in Time (Elman, J. L. (1990), *Cognitive Science*)

(2) Backpropagation Through time: What It Does and How to Do It (Werbos, P. J. (1990), *Proceedings of the IEEE*, 78(10))

(3) Das Elman Netz

(4) A Beginner's Guide to LSTMs and Recurrent Neural Networks

(5) Simple Recurrent Neural Network

2.18 Quantum Mechanics – Wave Function in the Plane

(Fabian Schubert)

The evolution of a particle's wave function in two dimensions under a static potential is given by

$$\frac{\partial}{\partial t}\psi\left(\mathbf{x},t\right) = \imath\frac{\hbar}{2m}\nabla_{xy}^2\psi\left(\mathbf{x},t\right) - \frac{\imath}{\hbar}V(\mathrm{x})\psi\left(\mathbf{x},t\right) \tag{20}$$

A simple approach to solving this equation explicitly would be to use an explicit forward step in time combined with a finite difference approximation of the spatial derivative:

$$\psi(\mathbf{x},t+\Delta t) \approx \psi(\mathbf{x},t) + \Delta t\frac{\partial}{\partial t}\psi\left(\mathbf{x},t\right) \tag{21}$$

$$\nabla_{xy}^2\psi\left(x,y,t\right) \approx \frac{\psi\left(x+\Delta x,y,t\right)+\psi\left(x-\Delta x,y,t\right)-2\psi\left(x,y,t\right)}{\Delta x^2} \tag{22}$$

$$+\frac{\psi\left(x,y+\Delta y,t\right)+\psi\left(x,y-\Delta y,t\right)-2\psi\left(x,y,t\right)}{\Delta y^2} \tag{23}$$

Note that this discretization in time could also be interpreted as a first-order approximation of the time evolution operator $\exp\left(-\imath\hat{H}\Delta t/\hbar\right) \approx 1-\imath\hat{H}\Delta t/\hbar$. This scheme comes with some problems, though. First, it is not unconditionally stable, meaning that too large step sizes can cause the numerical solution to diverge. Second, it does not preserve the total probability $\int \mathrm{d}^2\mathrm{x}||\psi\left(\mathbf{x},t\right)||^2 = 1$. Both of these problems can be avoided by using the implicit Crank-Nicolson algorithm (2). For simplicity, one can define $\hbar = 1$.

A number of interesting scattering experiments can be performed by choosing appropriate potentials and by initializing a traveling wave package. For example, a wave function of the form

$$\psi\left(\mathbf{x}\right) \propto \exp\left[\imath\mathbf{k}_0\mathbf{x} - \frac{||\mathbf{x}-\mathbf{x}_0||^2}{4\sigma_x^2}\right] \tag{24}$$

will evolve as a traveling wave packet with mean wave vector $\mathbf{k}_0$. It can be used to investigate the scattering behavior for different types of potentials.

- A potential described by a step function should lead to a splitting of the wave function because of its partial reflection, where the reflection depends on the ratio between the step height and the particle's energy (3).
- An impenetrable potential wall with two small slits in it should result in the well-known interference patterns seen in double-slit experiments.
- Scattering of a plane wave with wave number k on a hard sphere with radius R results in a differential cross section of $\frac{\mathrm{d}\sigma}{\mathrm{d}\Omega} = \frac{\sin^2 kR}{k^2}$.

1. Implement a solver of the two-dimensional Schrödinger equation using the method described in the introduction. You should provide an option for both periodic boundaries as well as hard walls, meaning $\psi = 0$ for all boundary sites.

2. Check whether the solver gives correct solutions for known cases, e.g. a simple free particle.

3. Initialize the wave function as a Gaussian wave packet and the potential as a step function. Investigate the cases where the potential energy of the step is larger/smaller than the mean energy of the packet. For a more focused spectrum of momenta, the spatial width of the packet should not be too small.

4. Simulate the scattering of a wave packet on a double slit. Instead of a potential, the wall should be implemented by fixing the amplitude of the wave function to zero at sites belonging to the obstacle. Compare the resulting interference pattern with the theoretical solution.

5. For the hard sphere scattering, use the same method as for the wall by fixing amplitudes to zero inside the sphere. A problem when estimating the resulting cross section of the scattering process is the fact that the resulting wave function will be a superposition of a scattered and an unperturbed component. So, you should think of a method how to separate the perturbed component from the solution. Since the differential cross section given above applies to the far field, you should make your system sufficiently large enough for a comparison with the simulation result.

(1) Solution of the Time-dependent Schrödinger Equation using the Crank-Nicolson algorithm

(2) Overview of the Crank-Nicolson algorithm

(3) Solution of Schrödinger equation for a step potential

2.19 Tetris

(Emanuele Varriale)

Tetris a simple tile-matching puzzle video game from the '80s (1). The name of the game comes from Tetrominoes, which are geometrical shapes formed by four square blocks. A random sequence of Tetriminoes falls down the playing field and the player can move them sideways or rotate them by 90 degrees, with the purpose of creating a horizontal line of ten units without gaps. When such a line is created, it gets cleared and any block above the deleted line will fall. When a certain number of lines are cleared, the game enters a new level. As the game progresses, each level causes the Tetriminos to fall faster, and the game ends when the stack of Tetriminos reaches the top of the playing field and no new Tetriminos are able to enter.

Your task is to implement (at least) a playable Tetris level. For the graphical part you can download the Simple and Fast Multimedia Library (SFML) library (2), in the tutorial (3) you can find instructions for installing the library and for compiling a SFML program. SFML provides useful classes such as:

RenderWindow: The window in which the game takes place.

Texture: An image to be used as texture.

Sprite: A textured rectangle, the basic building block of computer graphics.

Event: Necessary to track user input

Keybord: Handles key pressing.

Clock: Handles time.

The tutorial explains in more detail the properties of these and other classes, but using the one listed here is enough to implement the game. You can also have a look at code implementing the Pong game (4).

During the game a score counter has to be displayed and when the game ends it has to be displayed if the player won or lost. You can use this textures for the background and for the tiles (5), but it is not mandatory.

(1) Tetris

(2) SFML download

(3) SFML tutorial

(4) Pong example: https://itp.uni-frankfurt.de/~varriale/Pong.zip

(5) Tetris textures: https://itp.uni-frankfurt.de/~varriale/images.zip

2.20 Pathfinding

(Emanuele Varriale)

Pathfinding algorithms have to find the optimal path between two points on a map (1). A map can be represented as a graph, the easiest way being "discretizing" the map into a grid: cells are nodes that are connected, for example, with nearest neighbours.

In an unweighted graph one can use a Breadth first search algorithm to explore the graph, enumerate all possible paths and then find the shortest paths to the targets (2).

The graph can also be weighted, representing, for example, the cost of walking over different terrains. The Uniform Cost Search algorithm (4) (similar to Dijkstra's algorithm) takes the weights into account by enumerating all possible paths, but prioritizing the ones with the least cost.

Another possibility is the greedy best first search (3), that has a heuristic measure of distance to the target and first explore the paths with the least distance. One such heuristic on a square grid can be, for example, the Manhattan distance $d = |x_1 - x_2| + |y_1 - y_2|$. It is faster than the other approaches but it is not guaranteed to find the optimal solution.

The A* algorithm (5) also prioritizes the search but it combines the calculated distance and the heuristic distance. With the right heuristic it finds the optimal solution, but exploring less paths than Uniform cost algorithm.

Your task is to implement this algorithms, compare the time performance, optimality of solution on different kind of maps. You should consider maps with normal cells with weight 1, walls, i.e. cells that cannot be visited, and, "forest" cells with weight 5. Use these elements to create interesting geometries showing different behaviours of the algorithms. As an implementation detail, `std::queue` and `std::priority_queue` can be used to store the nodes to visit next.

(1) Pathfinding

(2) Breadth first search

(3) Greedy best first

(4) Uniform cost search

(5) A* algorithm

2.21 Stochastic simulation of chemical reactions

(Emanuele Varriale)

Modelling the time evolution of a system of chemically reacting molecules with a molecular dynamics simulation describing every molecule and collision is computationally costly.

On the other hand, in a well-stirred system of molecules of N different species, individual molecules can be disregarded and it is possible to consider only the vector formed by the total number of molecules of each species $\vec{X}(t) = [X_1(t), \ldots, X_N(t)]$. Often such a system is studied with a Reaction Rate Equation (RRE) such as $d\vec{X}/dt = \vec{f}(\vec{X})$, a deterministic equation that treats the X_i's as continuous variables. Solving the RRE gives reasonable solutions only if the stochastic fluctuations in the system are negligible.

When the fluctuations are not negligible, a possibility is to solve the Chemical Master Equation (CME) describing the time evolution of the conditional probability $P(\vec{x}, t|\vec{x}_0, t_0)$ of having the state vector $\vec{X}(t) = \vec{x}$ with initial condition $\vec{X}(t_0) = \vec{x}_0$. This is in general hard to do, and it is much easier to simulate many runs of the system with a dynamics that follows without approximation from the CME.

There are different algorithms to implement this, such as the First-reaction Method, the Direct Method and the Sorting Direct Method, they differ in how the time of the next reaction and which reaction occurs are computed. An introduction to this topic and to the methods can be found in the review (1).

The central dogma of molecular biology states that genetic information flows from genes to mRNA molecules through transcription and from mRNA to proteins through translation (2). Gene expression is an inherently stochastic process: Genes are activated and inactivated by random association and dissociation events, transcription of mRNA molecules is typically rare, and many proteins are present in

low numbers per cell. A simple "standard model" of gene expression can be used to study this system, as described in (3).

This a model is analytically tractable since for non-homogeneous differential equations the following holds:

$$\dot{x}(t) = -rx(t) + f(t) \qquad \Longrightarrow \qquad x(t) = x_0 e^{-rt} + e^{-rt} \int_0^t e^{-rt'} f(t') dt'. \tag{25}$$

Your task is to simulate the "standard model" of gene expression using the three different methods. Compare the number of operations that are needed for each method, for a fixed simulation time. An interesting quantity for the direct methods is the *search depth*, see (1). Discuss the results, what quantities fluctuate the most? Why? Generate 100 simulations and check that the analytical solutions for the average quantities are compatible with the numerical results. Initial conditions should be $n_1(0) = n_2(0) = n_3(0) = 0$, and use the following table for the parameters, see (3) for their meaning.

λ_1^+	λ_1^-	λ_2	λ_3	τ_1	τ_2	τ_3
0.1	0.1	0.05	0.5	5 s	100 s	333 s

Repeat a similar analysis for the gene regulation network given in Fig. 7 of (4) (excluding the analytical solution). You should better appreciate the performance boost of the sorting direct method since the probabilities of reactions wildly change over the simulation.

(1) Stochastic Simulation of Chemical Kinetics (Gillespie, Annu. Rev. Phys. Chem., 2007)

(2) Central dogma of molecular biology

(3) Models of stochastic gene expression, Paulson (Physics of Life Reviews, 2005)

(4) The sorting direct method (McCollum et al., Computational Biology and Chemistry, 2006)

2.22 Classification via Decision Trees: Reproduction of Human Biases

(Emanuele Varriale)

When assigning a decision-making task to a machine, it was originally expected that the machine would be capable of forming more rational choices than humans, basing the decision on facts rather than prejudice.

There are however several examples for machine learning algorithms that reproduce or even manifest social bias, as it is presented to the algorithm in the training data that is learned as ground truth. Among these for example the PredPol algorithm that was designed to predict time and location of crimes to reduce human bias and distribute human resources in policing. It was found to target certain neighbourhoods based on racial background instead of true crime rates (1).

In this project, it should be analysed whether social bias reproduction can be observed for credit approvals. The dataset for this analysis can be found at UCI machine learning repository (2) which includes information on credit score, income, education level, debts and years of employment aswell as gender, ethnicity and marital status with a label of whether credits were approved or not.

The applied machine learning technique should deliver an interpretable set of rules for prediciting (a so-called white box classifier as opposed to black box algorithms where inner structures remain hidden) and because of the mix of features in the data, the algorithm should be capable of handling categorical (gender, employment) and numerical (debt, age) features at the same time. Both prerequesites are met by CART (Classification And Regression Trees (3)). With this technique, a binary tree is built, the dataset is split into two parts at every node, creating branches until a decision is reached. To find the best splits, a greedy algorithm (4) is used to find the feature and threshold that minimizes the impurity of the data at the nodes. As measure, the Gini Impurity (5) is used. For a set of items with J classes at a node it is computed as

$$I_G(p) = \sum_{i=1}^{J} p_i \sum_{k \neq i} p_k = 1 - \sum_{i=1}^{J} p_i^2$$

where p_i is the fraction of items labeled with class i in the set.

To prevent overfitting, a minimal number of instances is set for any leaf (end node of a branch where the prediction is made).

Outline of the project

1. Define a data structure, that can handle arbitrary number of categorical and numerical values similar to arrays or vectors
2. Implement a class Decision Tree, that has the following functions:
   - `save` and `load` of the trained model
   - calculate the `gini_index` for a given data set
   - `split` a given dataset into two parts using a given feature and threshold
   - `evaluate` all possible splits for a given dataset
   - `build_tree` by top-down greedy search until pure node or minimal number of instances at node is reached
   - `predict` a label for a given instance
   - `test` the model by comparing predictions to given labels
3. Train the decision tree on 70% of the credit screening data
4. Evaluate the trained model using the remaining 30% of data as testing set
5. Find the position of nodes, at which ethnicity or gender are used as split feature
6. Systematically perturb ethnicity or gender values in the remaining 30% of the data and analyse whether these perturbations influence the prediction for credit approval.

(1) PredPol social bias

(2) Dataset of Japan Credit Screening

(3) CART

(4) Greedy Algorithms

(5) Gini Impurity

2.23 Digit recognition with multilayer perceptron and Qt-based GUI

(Anton Motornenko)

Artificial neural networks are universal computing tools. Such systems "learn" to perform tasks by considering examples, generally without being programmed with any task-specific rules. For example, in image recognition, they might learn to identify images that contain cats by analyzing example images that have been manually labeled as "cat" or "no cat" and using the results to identify cats in other images.

This project deals with implementation of the Feedforward neural network (1) that is realized through multilayer perceptron (MLP) (2). MLP utilizes a supervised learning technique called backpropagation for training (3), (4) that is needed in the calculation of the weights to be used in the network.

The implemented network then may be trained to perform recognition of handwritten digits. Since we work with supervised learning technique the database of handwritten and classified images of digits is needed to serve as an input into the network. For this purpose the MNIST dataset (5) may be used.

On top of it, the graphical user interface (GUI) should be realized which will enable a user to draw numbers into a given space in the GUI. The drawn number will then be classified by the algorithm of the network. Qt (6) – the C++ based cross-platform framework for creating GUI would be used.

The project is divided in two parts. The first part is the Back-End, where all the calculations are made, and the second one is the Front-End that implements GUI.

The Back-End contains:

- the neural network itself, which should recognize the handwritten digits;
- a small library for vector calculations;
- implementation of gradient descent method (3), that will be used to determine weights of the network;

The Front-End contains:

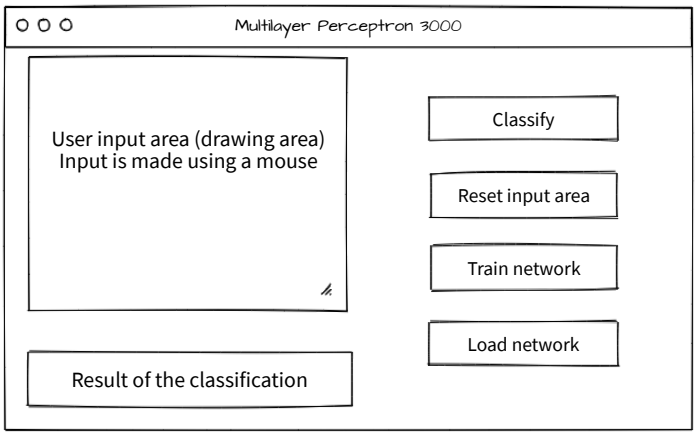

Figure 1: Sketch of the GUI.

- GUI that enables a user to draw numbers by hand into a given space in the GUI;
- passing of the drawn number into the Back-End, so the classification is performed;
- output of the classification result.

Note: Benchmark the predictive power of your network. For this split the MNIST dataset in two samples. Use 70% of images in the dataset to train network (training sample), and 30% (validation sample) to test the predictive power. Estimate accuracy of the network by calculating the number of correct guesses in the validation sample.

(1) Feedforward Neural Network (Wikipedia)

(2) Multilayer perceptron (Wikipedia)

(3) Gradient descent (Wikipedia)

(4) Backpropagation (Wikipedia)

(5) The MNIST database of handwritten digits

(6) Qt Documentation (Qt)

2.24 Reinforcement learning: AI Tetris

(Anton Motornenko)

Nowadays machine learning serve as a universal tool that can be applied to almost any problem. The machine learning provide methods to create a mathematical model for some specific task without being explicitly programmed to perform the task. These models (called as networks) are constructed through learning when a network is trained to reproduce some desired output while processing corresponding input. These techniques allow computers to learn to determine objects on images, recognize speech, or to play games.

The goal of this project is to implement a neural network and train it to play Tetris (1). This may be done using methods of Reinforcement machine learning (2). Reinforcement learning is particularly well-suited to problems that include a long-term versus short-term reward trade-off. It has been applied successfully to various problems, including robot control, elevator scheduling, telecommunications, backgammon, checkers and go (AlphaGo) (3). This will allow computer to learn how to play Tetris through experience (i.e. no input data is required), while playing the computer should optimize its model of behavior to receive the maximal score in the game.

The task itself is:

- Implement a Tetris game with some simple visualization;
- Create a mapping between state of the game and neural network, i.e. translate the current state of the game into some vector that will be used as input into the network;

- *Either* use Tensorflow (4) C++ API;
- *Either* provide your own implementation of the neural network;
- Connect game and neural network, so network will be able to play the game and train at the same time. The network will define it's actions in the game according to its current state and update it's weights according to the obtained score *maybe provide some details concerning the algorithm for updating the weights*;

(1) Tetris game

(2) Reinforcement learning

(3) AlphaGo

(4) Tensorflow

(5) Playing Tetris with Deep Reinforcement Learning