OLIVER KNILL

Lecture 13: Computing

13.1. **Computing** deals with algorithms and the practice of programming. While the subject intersects with computer science, information technology, the theory is by nature rather mathematical. New aspects have emerged. Computers have opened the field of **experimental mathematics** and serve now as the **laboratory** for new mathematics. Computers are not only able to **simulate** more and more of our physical world, they allow us to **explore** new worlds. They have become important parts of our **senses** and interfaces between measuring devices and sensors and our brains.

13.2. A mathematician pioneering new grounds with computer experiments does similar work than an experimental physicist. Computers have blurred the boundaries between physics and mathematics. There are some mathematicians who do much more experiments than some theoretical physicists working in domains where access to measurements are absent. According to Borwein and Bailey, experimental mathematics consists of:

Gain insight and intuition.	Explore possible new results
Find patterns and relations	Suggest approaches for proofs
Display mathematical principles	Automate lengthy hand derivations
Test and falsify conjectures	Confirm already existing proofs

13.3. Computers can also assist us in proofs. If all the source code for the computation can be inspected and all computations are done symbolically or with integers, one can verify that the proof is correct and there is no fundamental difference between a computer assisted proof or a proof done on paper. There are aspects which need to be considered however. There is a trust required in that the computer does the right thing, that the programs are correct. But this is no different with our brains. We have to trust that we do not forget things when doing a proof or that previous work is correct. The layers of trust we have in humanly produced proofs are already wide. A theorem which has been proven in many different ways is more reliable than a theorem for which the proof is thousands of pages long and been done by dozens of mathematicians.

13.4. We have seen examples in this course, where the computer has proven geometric theorems. Algebraic computations or integral computations, or solving differential equations, this all gets done swiftly with computers. When using computers to prove things, reading and verifying the computer program is part of the proof. If Goldbach's conjecture would be known to be true for all $n > 10^{18}$, the conjecture should be accepted because numerical verifications have been done until $2 \cdot 10^{18}$ until today. The first famous theorem proven with the help of a computer was the "4 color theorem" in 1976. The theorem tells that any finite simple planar graph admits a function on the nodes taking 4 values so that neighboring nodes do not have the same color.

13.5. It would be unreasonable to ask a human to manually verify the finitely many cases not covered by the theory. A mathematician would like to insist that software which is used open source. But even that would not be enough for a purist. It is possible for example that the CPU produces errors. Commercial computer algebra software without insight into the source code does not qualify because there could be bugs. Mathematics is eternal. A once established fact is true

for ever. With more openness in CPU architecture as well as insistence of open source compilers and tools one might accept computer assisted proofs more. Still, one also wants to understand "why" a result is true and hope that a proof produces "insight".

13.6. Besides assisting in experiments and proofs, also the visualization and illustration aspect is important in mathematical activities. And that is where computers shine. It is possible these days to record a lecture with various cameras, cut the material together, enhance it with multimedia like pictures, movies, graphics and so generate a **virtual reality environment** which would not be possible without the computer. This is not only valuable for pedagogical reason. A good visualization can lead to new result and a good illustration can help to increase the global understanding of the subject and to pass it on to future generations.

13.7. Here are some pointers in the history of computing. It is best done by looking at computing devices. The conceptual development of computing is much more subtle. There is dispute for example whether the Sumerian Abacus is real and should count as a computing device. Writing on clay and so moving physically matter around is the same already than counting with pebbles without having them be formally attached to a device.

2700BC	Sumerian Abacus	1935	Zuse 1 programmable	1973	Windowed OS
200BC	Chinese Abacus	1941	Zuse 3	1975	Altair 8800
150BC	Astrolabe	1943	Harvard Mark I	1976	Cray I
125BC	Antikythera	1944	Colossus	1977	Apple II
1500	Khipus	1946	ENIAC	1981	Windows I
1300	Modern Abacus	1947	Transistor	1983	IBM PC
1400	Yupana (Inkas)	1948	Curta Gear Calculator	1984	Macintosh
1600	Slide rule	1952	IBM 701	1985	Atari
1623	Schickard computer	1958	Integrated circuit	1988	Next
1642	Pascal Calculator	1969	Arpanet	1989	HTTP
1672	Leibniz multiplier	1971	Microchip	1993	Webbrowser, PDA
1801	Punch cards	1972	Email	1998	Google
1822	Difference Engine	1972	HP-35 calculator	2007	iPhone, Android
1876	Mechanical integrator	1972	first digital watch	2015	Apple watch
				2021	desktop quantum computer

13.8. We live in a time of exponentially exploding technology measures. **Moore's law** from 1965 predicted that semiconductor technology doubles in capacity and overall performance every 2 years. This has happened since. Some futurologists like Ray Kurzweil conclude from this technological singularity in which artificial intelligence might take over.

13.9. In 1937, **Alan Turing** introduced the idea of a **Turing machine**, a theoretical model of a computer which allows to quantify complexity. It has finitely many states $S = \{s_1, ..., s_n, h\}$ and works on an tape of $0-1$ sequences. The state h is the "halt" state. If it is reached, the machine stops. The machine has rules which tells what it does if it is in state s and reads a letter a. Depending on s and a, it writes 1 or 0 or moves the tape to the left or right and moves into a new state. Turing showed that anything we know to compute today can be computed with Turing machines. For any known machine, there is a polynomial p so that a computation done in k steps with that computer can be done in $p(k)$ steps on a Turing machine.

13.10. What can be computed? The Church's thesis of 1934 states that everything which can be computed can be computed with Turing machines. This almost certainly will remain a thesis as we never know whether we have understood all fundamental mechanisms of nature. It could be possible in principle that a time machine is possible allowing a computer to look ahead what happens and do so "computations" which are not possible by a Turing machine. The Church thesis is a sound assumption as it removes possibilities like unnatural phenomenons or computations done by some sort of religion, like asking a deity for help with some computation.

13.11. Similarly as in mathematics itself, there are limitations of computing. Turing's setup allowed him to enumerate all possible Turing machine and use them as input of an other machine. Denote by TM the set of all pairs (T, x), where T is a Turing machine and x is a finite input. Let $H \subset TM$ denote the set of Turing machines (T, x) which halt with the tape x as input. Turing looked at the decision problem: is there a machine which decides whether a given machine (T, x) is in H or not. An ingenious Diagonal argument of Turing shows that the answer is "no". [Proof: assume there is a machine $HALT$ which returns from the input (T, x) the output HALT(T, x) = true, if T halts with the input x and otherwise returns HALT(T, x) = false.

13.12. Turing constructs a Turing machine DIAGONAL, which does the following:
1) Read x. 2) Define Stop=HALT(x,x) 3) While Stop=True repeat Stop:=True; 4) Stop
Now, DIAGONAL is either in H or not. If DIAGONAL is in H, then the variable Stop is true which means that the machine DIAGONAL runs for ever and DIAGONAL is not in H. But if DIAGONAL is not in H, then the variable Stop is false which means that the loop 3) is never entered and the machine stops. The machine is in H.]
Lets go back to the problem of distinguishing "easy" and "hard" problems: One calls **P** the class of decision problems that are solvable in polynomial time and **NP** the class of decision problems which can efficiently be tested if the solution is given. These categories do not depend on the computing model used.

13.13. An important aspect of computing is the question how to decide whether a computation is "easy" or "hard". The question "N=NP?" is the most important open problem in theoretical computer science. It is one of the seven **millenium problems** and it is widely believed that $P \neq NP$.

13.14. If a mathematical task is such that every other NP problem can be reduced to it, it is called **NP-complete**. Popular games like **Minesweeper** or **Tetris** are known to be NP-complete. If it were true that $P \neq NP$ (which is what most computer scientists believe to be the case), then there are no efficient algorithm to beat the game. An example of a rather accessible NP-complete problem is the task to find the largest complete subgraph in a finite simple graph. While for many classes of graphs like networks triangulating a surface this is not a big deal, in general, this is computationally hard.

13.15. An other example of an NP-complete problem is the **balanced number partitioning problem**: given n positive integers, divide them into two subsets A, B, so that the sum in A and the sum in B are as close as possible. A first shot: chose the largest remaining number and distribute it to alternatively to the two sets.

13.16. We all feel that it is harder to **find a solution to a problem** rather than to **verify a solution**. If $N \neq NP$ there are one way functions, functions which are easy to compute but hard to verify.
For some important problems, we do not even know whether they are in NP. Here are two examples: 1) **the integer factoring problem**: given n find the factors 2) **the merit factor problem**: minimize $\sum_{k=-n}^{n} c_k^2$, where $c_k = \sum_{j=0}^{n-k} a_j a_{j+k}$ An efficient algorithm for the first one would have enormous consequences for our modern lives. Watch the intellectual thriller movie "Traveling Salesman (2012)" to appreciate this.

13.17. Finally, lets look at some mathematical problems in artificial intelligence AI:

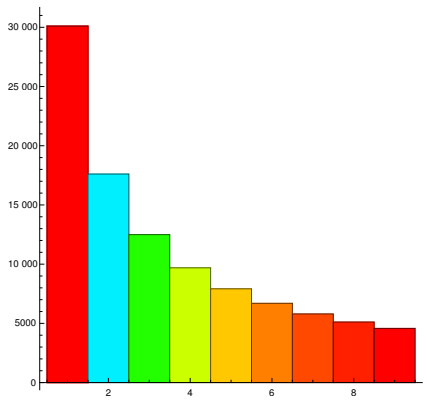
problem solving	playing games like chess, performing algorithms, solving puzzles
pattern matching	speech, music, image, face, handwriting, plagiarism detection, spam
reconstruction	tomography, city reconstruction, body scanning
research	computer assisted proofs, discovering theorems, verifying proofs
data mining	knowledge acquisition, knowledge organization, learning
translation	language translation, porting applications to programming languages
creativity	writing poems, jokes, novels, music pieces, painting, sculpture
simulation	physics engines, evolution of bots, game development, aircraft design
inverse problems	earth quake location, oil depository, tomography
prediction	weather prediction, climate change, warming, epidemics, supplies

We had started with basic human activities defining mathematical fields, we end the course with mathematical activities defining some aspects of computing. Our journey through math is over.

Work problems

13.18. 1) Experimental mathematics uses a method used a lot in other natural sciences sciences: we experiment! This can happen on paper, but also with the help of a computer. We will look at examples illustrating this. with **Benford's law** which deals with the statistics of the first significant digit in data. Simon Newcomb found the law in 1881 and Frank Benford made significant progress on it in 1938. Here is an example where one can prove things. Look at the first digits of the sequence 2^n. One can prove that the digit k appears with probability $p_k = \log_{10}(1 + 1/k)$. The digit 1 for example occurs with about $\log_{10}(2) = 0.30$ which is 30 percent. Lets experiment and look at 2^n for $n = 1$ to $n = 100'000$ and determine the first digit:

```
data = Table[First[IntegerDigits[2^n]], {n, 1, 100000}];
S = Histogram[data, 10, ColorFunction -> Hue]
```



13.19. How does one compute the probability? If we look at the logarithms, then $\log(2^n) = n\log(2)$. The first digit is 1 if the rest of $[n\log(2)]$ modulo 1 is between 0 and $\log(2)$. The first digit is 2 if it is between $\log(2)$ and $\log(3)$ etc. The probability that the letter is k is $\log_{10}(k+1) - \log_{10}(k)$.

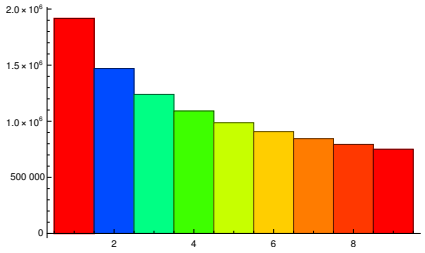
13.20. One can look at the first significant digit problem on other sequences like squares $1, 4, 9, 1, 2, 3, 4, 6, 8,$ Here is an experiment:

```
data = Table[First[IntegerDigits[n^2]], {n, 1, 10000000}];
S = Histogram[data, 10, ColorFunction -> Hue]
```

It is interesting because we want to see what the distribution of $2\log(n)$ is modulo 1. It looks as if we have a similar Benford law here. Indeed it is a generalized Benford law with $p_k = \frac{\int_k^{k+1} x^{-\alpha}\, dx}{\int_1^{10} x^{-\alpha}\, dx} =$

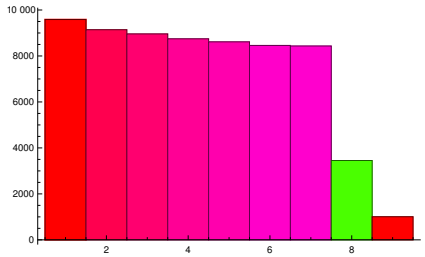
$[(k+1)^{1-\alpha}-k^{1-\alpha}]/(10^{1-\alpha}-1)$. It interpolates the Benford law $\alpha = 1$ with the uniform distribution $\alpha = 0$.

We have the digit 1, if $\log(n) \in k + [0, \log(2)]$. How many cases are in 1000 and 2000. It is $\sqrt{2000}-\sqrt{1000} = \sqrt{1000}(\sqrt{2}-1)$. How many cases are in 2000 and 3000. It is $\sqrt{3000}-\sqrt{2000} = \sqrt{1000}(\sqrt{3}-\sqrt{2})$.



a) Experiment to find the What is the first significant digit of the prime numbers?

```
data = Table[First[IntegerDigits[Prime[n]]], {n, 1, 664000}];
S = Histogram[data, 10, ColorFunction -> Hue]
```



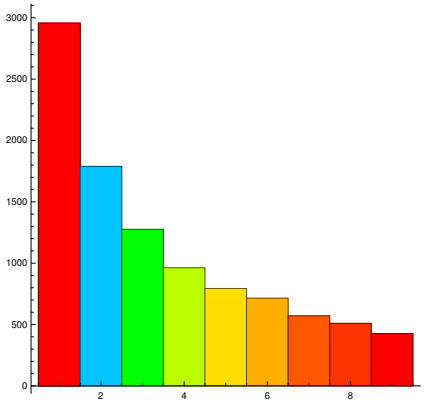
How many primes are there in 1000 and 2000. We expect $1000/Log[1000]$ primes in there and $1000/Log[2000]$ with first significant digit 2.

S1=ListPlot[Table[PrimePi[k],k,10000]]; S2=ListPlot[Table[k/Log[k],k,10000]]; Show[S1,S2]

We expect the distribution to be $a/\log(k)$, where $a = \sum 1/\log(k)$.

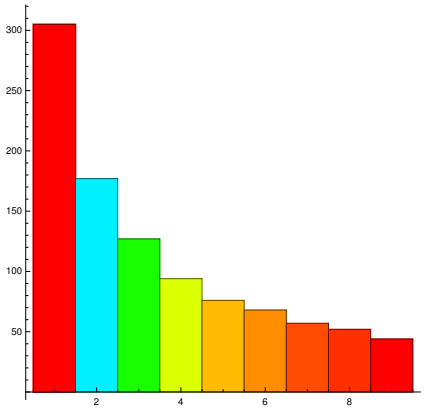
For factorials, the limiting distribution is known to be the Benford distribution. There is no reason why $\log_{10}(n!) mod 1$ should not be uniformly distributed.

```
data = Table[First[IntegerDigits[n!]], {n, 1, 10000}];
S = Histogram[data, 10, ColorFunction -> Hue]
```



b) Also for the partition numbers, $p(n)$, which give the number of possibilities in which the number n can be written as a sum of integers, we measure that the Benford distribution takes place. As far as we know this is not known.

```
data = Table[First[IntegerDigits[PartitionsP[n]]], {n, 1, 10000}];
S = Histogram[data, 10, ColorFunction -> Hue]
```



2) We look at an other problems where computers could play a role for the solution. Usually, in the case a counter example exists.
Goldbach's conjecture tells that every even number larger than 2 is the sum of two primes. While it is unlikely that a computer search will find a counter example, it might be that computer search finds the solution to the problem. How? Here is a complex analytic approach which is a caricature of much more sophisticated methods developed since 90 years. Search for function $f(x) = \sum_p a_n x^p$ with a_p positive so that f can be written in terms of functions for which integration theory works well (trig functions, exponentials, polynomials, hypergeometric functions etc). Then check (theoretically) that the Taylor coefficients of $f(x)^2$ are positive. One can do that by integration in the complex plane. This approach is a long shot since it is unlikely that a closed form for $f(x)$ can be found which works. More likely is that one can approximate things well enough to push the threshold higher above which counter examples must appear. The task is now to find coefficents a_n such that $f(x)$ is an explicit function we can compute with.
3) **Euler cuboid** It is not known whether there exists a cuboid with integer side length such that all face diagonals are integers and additionally, also the large diagonal is an integer. Computer searches have found nothing. One can search on parametrized families of Euler cubes but the **perfect Euler cuboid** - if it exists - could be so large that no computer could find it by brute force search. There are ways to explore large Euler cuboids. One is to look for families of Euler cuboids and in this space of Euler cuboids use linear analysis to predict large parameters for which we are close to **perfect cuboids**. Maybe such an approach could lead to an example. By a lucky punch. It is however still also possible that there are no perfect Euler cuboid.
4) **The Riemann hypothesis** is a prototype problem, where experiments led to more and more support that the Riemann hypothesis is true. One approach looks numerically for roots of the Riemann zeta function on the critical line. One approach is called the **Merten's approach**. Define the **Möbius function** $\mu(n) = 1$ if n is the product of an even number of different primes and -1 if n is the product of an odd number of different primes. In all other cases, that is if n has a square factor larger than 1, we have $\mu(n) = 0$. Is μ sufficiently random so that $S_n = \sum_{k=1}^n \mu(k)$ grows like the iterated law of logarithm? While it looks as if the $\mu(n)$ behave like a random sequence, there are some correlations.
5) **Billiards** One does not know whether there are triangular billiards without periodic points. One also does not know whether there are smooth convex billiards besides the ellipse for which one has **integrability** in the sense that all points are either periodic, asymptotic to a periodic point or almost periodic in the sense that the dynamics is equivalent to a translation on a finite or infinite dimensional torus. Integrability implies that one can compute the future orbit arbitrarily well without running into the sensitive dependence on initial condition problems.
5) **Standard map**. Verify that for $c > 2$ and all n $\frac{1}{n}\int_0^1 \int_0^1 \log|\partial_x f_n(x,y)|\ dxdy \geq \log(\frac{c}{2})$. I myself have tried over a decade to prove this using methods from quantum mechanics, calculus of variations and complex analytic methods. The problem is open. The left hand side converges to the average of the Lyapunov exponents which is in this case also the **entropy** of the map.

6) **prime twins**. This is a problem, which can first of all be investigated statistically. Similarly as Gauss looked numerically for a law describing the frequency of the prime numbers, one can first see, hoe many prime twins one has to expect in a certain interval and then see whether this expectation is confirmed. Furthermore one can look whether there are any patterns on arithmetic subsequences. Maybe there are some unexpected sequences along which there are more prime twins. Related to the twine prims problem is the problem to estimate the minimal distance between two Gaussian primes in the complex plane.

7) **Normality of π. This is an example where one can have fun with statistics. Are there any statistical tests which indicate that the digits of π are not normal? The decimal digits of π appear random enough so that every digit appears with the same frequency.**

8) Odd perfect numbers. While a brute force search is unlikely, there are other approaches which are more likely to find an odd perfect number. Any perfect number satisfies $\sigma(n)/n = 2$, where $\sigma(n)$ is the sum of all the divisors of n including n. Take a large set $B = \{p_1, ..., p_s\}$ of primes. For every $k = (k_1, ..., k_s)$, form the number $n = p_1^{k_1} p_2^{k_2} ... p_s^{k_l}$. We have $\sigma(p^k) = (1+p+p^2+...p^k)/p^k = (p-p^{-k})/(p-1)$. Let $a(p,k) = \log(\sigma(p^k))$. The goal is to find $(k_1, ..., k_s)$ such that

$$\log(a(p_1, k_1)) + ... + \log(a(p_l, k_l)) - k_1 \log(p_1) - ...k_s \log(p_s) = \log(2) \ .$$

The idea is to keep first the primes and change only the "dials" k_j in a controlled way. Certain dial changes will produce a very small net change of the left hand side. For large primes and large n the first order change will dominate and methods from Diophantine geometry could be used and linear algebra to get close to the right hand side. If lucky (provided of course there is an odd perfect number), one could hit it like this. If we would take 1000 primes each 1000 digits long and deal with exponents of the order of 1000, we would investigate numbers with billions of digits.

9) Turing machines. We see a concrete Turing machine and evolve it a few steps to see what it does. The machine is initially in state $\boxed{1}$ and starts with an empty tape.

	-3	-2	-1	0	1	2	3	···
···	0	0	0	0	0	0	0	···

Here is the definition of a machine with three states:

Input 0

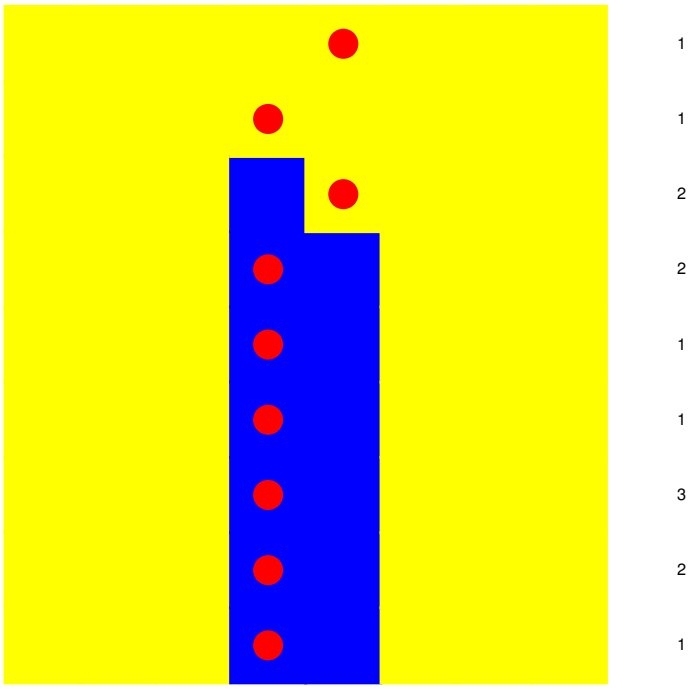
State	New state	Write	move to
1	2	0	left
2	3	1	right
3	1	1	left

Input 1

State	New state	Write	move to
1	2	1	left
2	1	0	right
3	3	0	right

a) We run this machine a few steps and mark the number, where the machine reads the tape. This place will move with time.

	-3	-2	-1	0	1	2	3	···	State=1
···	0	0	0	0	0	0	0	···	State=1
···	0	0	0	0	0	0	0	···	State=2
···	0	0	0	0	0	0	0	···	State=3
···	0	0	0	0	0	0	0	···	State=1
···	0	0	0	0	0	0	0	···	State=1



b) Here is the definition of a new machine with three states:

Input 0

State	New state	Write	move to
1	2	1	right
2	3	1	right
3	3	0	left

Input 1

State	New state	Write	move to
1	2	1	left
2	1	0	right
3	3	0	left

c) Run it!

···	0	0	0	0	0	0	0	···	State=1
···								···	State=
···								···	State=
···								···	State=

10) The following problem is known to be NP-complete. This means that if it could be solved in polynomial time, then all NP problems would be polynomial and P=NP. In other words, if you can design a method, which solves the problem in a manner which is polynomial in n, you win a Million dollars and you would have solved the most important problem in computer science.

Given n positive integers $a_1, \ldots, a_n$, divide them up into two subsets, so that the sum of these numbers in one set and the sum of numbers in the other set are as close together as possible.

For example, let

$$A = \{2, 3, 5, 7, 11, 13, 17\} .$$

We can apply the "cruel school selection" algorithm: the largest two elements are the team leaders of a team, who alternatively select the best remaining candidate. What do you get?

a. Can you do better than that? How well can you divide up the numbers better?

11) One does not know how to factor integers of 200 digits efficiently. Can we find some factors fast?

b) There is a simple method to decide whether a number is divisible by 9. What is it? For example: is

$$121212121212121212$$

divisible by 9?

c) There is a simple method to decide whether a number is divisible by 11. What is it? 4) For example is

$$1212121212121212121212$$

divisible by 11?

d) Which factors can be detected by looking at the last decimal digit of a number only?

e) The 1,-3,2 method for divisibility by 7 is to split the number into groups of 6, then multiply in each group the three pairs by 1,-3,2, then add up all results. For example $11111111111114 = (000011)(111111)(111114)$ leads to $2 * 11 + 1 * 11 - 3 * 11 + 2 * 11 + 1 * 11 - 3 * 11 + 2 * 14 = 28$ which is divisible by 7.

Is

$$111111111111111111111111111111111111111111111111$$

divisible by 7 (48 digits of 1).

Oliver Knill, knill@math.harvard.edu Math E-320, Harvard Extension School, 2010-2022