Integrating a computational perspective in the basic science education

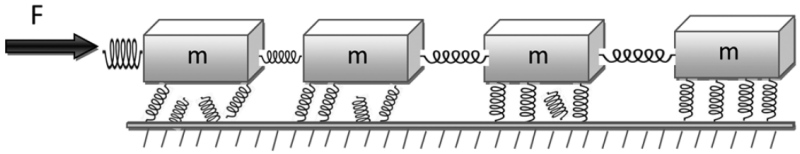
Morten Hjorth-Jensen Email morten.hjorth-jensen@fys.uio.no[1,2]

National Superconducting Cyclotron Laboratory and Department of Physics and Astronomy, Michigan State University, East Lansing, MI 48824, USA[1]

Department of Physics, University of Oslo, Oslo, Norway[2]

April 4, 2016

Wouldn't it be cool if your mechanics students could reproduce results in a PRL?



Grand challenge project in FYS-MEK1100 (Mechanics, University of Oslo), Second Semester: a friction model to be solved as coupled ODEs. And find problems with the article?

PRL **103,** 194301 (2009) PHYSICAL REVIEW LETTERS week ending 6 NOVEMBER 2009

Dynamics of Transition from Static to Kinetic Friction

O. M. Braun,[1] I. Barel,[2] and M. Urbakh[2]
[1]*Institute of Physics, National Academy of Sciences of Ukraine, 03028 Kiev, Ukraine*
[2]*School of Chemistry, Tel Aviv University, 69978 Tel Aviv, Israel*
(Received 29 June 2009; published 6 November 2009)

We propose a model for a description of dynamics of cracklike processes that occur at the interface between two blocks prior to the onset of frictional motion. We find that the onset of sliding is preceded by well-defined detachment fronts initiated at the slider trailing edge and extended across the slider over limited lengths smaller than the overall length of the slider. Three different types of detachment fronts may play a role in the onset of sliding: (i) Rayleigh (surface sound) fronts, (ii) slow detachment fronts, and (iii) fast fronts. The important consequence of the precursor dynamics is that before the transition to overall sliding occurs, the initially uniform, unstressed slider is already transformed into a highly nonuniform, stressed state. Our model allows us to explain experimental observations and predicts the effect of material properties on the dynamics of the transition to sliding.

DOI: 10.1103/PhysRevLett.103.194301 PACS numbers: 46.55.+d, 46.50.+a, 81.40.Pq

Why is computing comptence important?

Definition of computing

With computing I will mean solving scientific problems using computers. It covers numerical, analytical as well as symbolic computing. Computing is also about developing an understanding of the scientific process by enhancing algorithmic thinking when solving problems.

Computing competence, what is it?

Computing competence has always been a central part of the science and engineering education. Traditionally, such competence meant mastering mathematical methods to solve science problems - by pen and paper.
Today our candidates are expected to use all available tools to solve scientific problems; computers primarily, but also pen and paper.
I will use the term/word algorithms in the broad meaning: methods (for example mathematical) to solve science problems, with and without computers.

What is computing competence about?

Computing competence is about

1. derivation, verification, and implementation of algorithms
2. understanding what can go wrong with algorithms
3. overview of important, known algorithms
4. understanding how algorithms are used to solve mathematical problems
5. reproducible science and ethics
6. algorithmic thinking for gaining deeper insights about scientific problems

Continuous verse discrete

Algorithms involving pen and paper are traditionally aimed at what we often refer to as continuous models.
Application of computers calls for approximate discrete models.
Much of the development of methods for continuous models are now being replaced by methods for discrete models in science and industry, simply because much larger problem classes can be addressed with discrete models, often also by simpler and more generic methodologies. However, verification of algorithms and understanding their limitations requires much of the classical knowledge about continuous models.

Why should basic university education undergo a shift from classical mathematics to modern computing?

1. The impact of the computer on mathematics is tremendous: science and industry now rely on solving mathematical problems through computing.
2. Computing increases the relevance in education by solving more realistic problems earlier.
3. Computing through programming is excellent training of creativity.
4. Computing enhances the understanding of abstractions and generalization.
5. Computing decreases the need for special tricks and tedious algebra, and shifts the focus to problem definition, visualization, and "what if" discussions.

The result is a deeper understanding of mathematical modeling. Not only is computing via programming a very powerful tool, it also a great pedagogical aid. For the mathematical training, there is one

Key principle in scientific modeling

The power of the scientific method lies in identifying a given problem as a special case of an abstract class of problems, identifying general solution methods for this class of problems, and applying a general method to the specific problem (applying means, in the case of computing, calculations by pen and paper, symbolic computing, or numerical computing by ready-made and/or self-written software). This generic view on problems and methods is particularly important for understanding how to apply available, generic software to solve a particular problem.

Computing competence is central to solving scientific problems

Definition of computing

Computing competence represents a central element in scientific problem solving, from basic education and research to essentially almost all advanced problems in modern societies. Computing competence is simply central to further progress. It enlarges the body of tools available to students and scientists beyond classical tools and allows for a more generic handling of problems. Focusing on algorithmic aspects results in deeper insights about scientific problems.

Today's project in science and industry tend to involve larger teams. Tools for reliable collaboration must therefore be mastered (e.g., version control systems, automated computer experiments for reproducibility, software and method documentation).

Modeling and computations as a way to enhance algorithminc thinking

Algorithmic thinking as a way to

- Enhance instruction based teaching
- Introduce Research based teaching from day one
- Trigger further insights in math and other disciplines
- Validation and verification of scientific results (the PRL example), with the possibility to emphasize ethical aspects as well. Version control is central.
- Good working practices from day one.

Research based teaching

How do we define it?

One possible definition: It is coupled to a direct participation in actual research and builds upon established knowledge and insights about scientific methods.

- It is the standard situation at all universities and takes normally place at the senior undergraduate/graduate level (isn't it too late?)
- It is seldom done in undergraduate courses.
- Taught by a researcher
- The student starts seeing the countour of the scientific approach leading her/him to make new interpretations, develop new insights and understandings that lead to further research.

Research based education

What should the education contain?

The standard situation we meet at an almost daily basis:

- Theory+experiment+simulation is almost the norm in research and industry
- To be able to model complex systems with no simple answers on closed form. Solve real problems.
- Emphasis on insight and understanding of fundamental principles and laws in the Sciences.
- Be able to visualize, present, discuss, interpret and come with a critical analysis of the results, and develop a sound ethical attitude to own and other's work.

Our education should reflect this.

Research based education

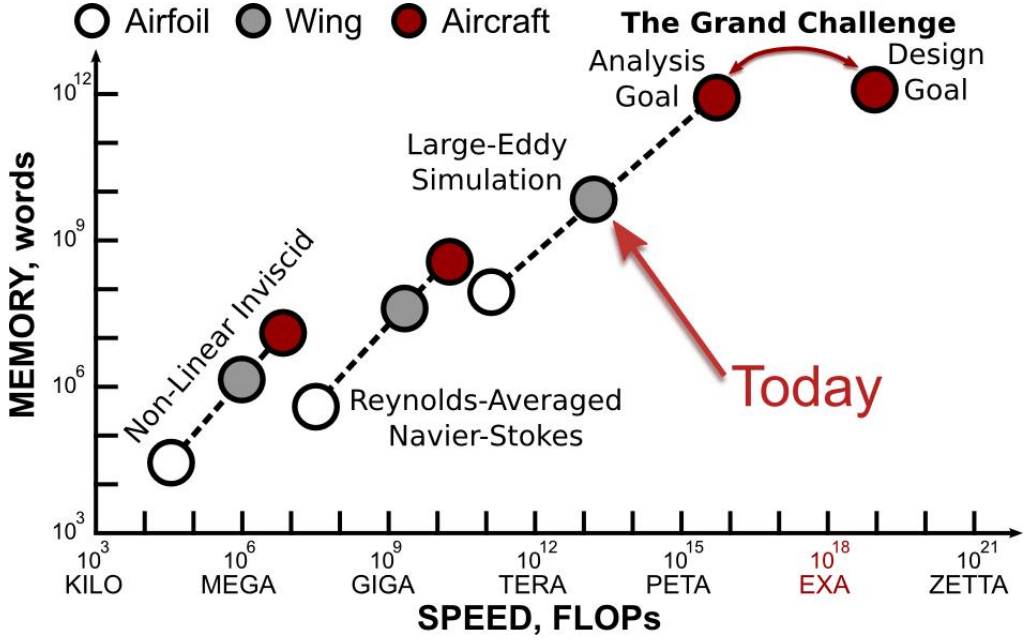
Normal workflow in Science and Engineering

- A problem is properly described using a precise (normal) language.
- It is translated to a mathematical problem using known laws and principles.
- It is solved, normally via numerical similations.
- The solution is visualized and analyzed.
- The solution to the problem is formulated.

People who master these skills bring an important compentence to society.

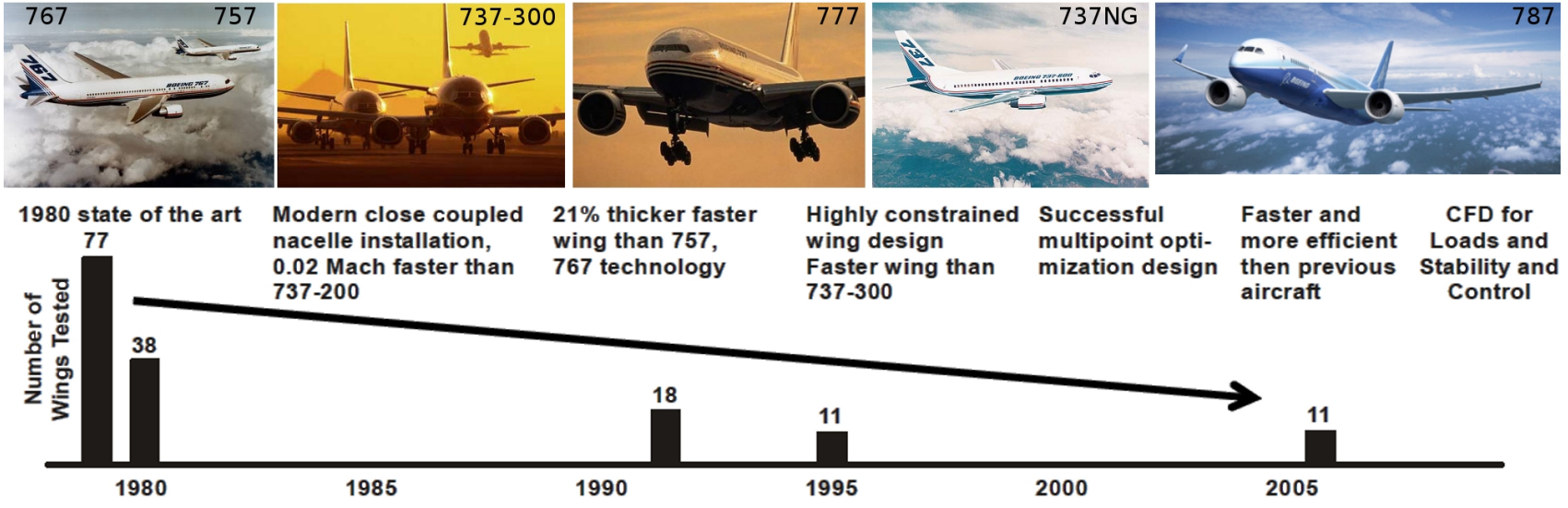
Large scale simulations

Fluid dynamical simulations central in air industry. Typical university courses which are taught address the physics of the lower left corner.



Large scale simulations

Fluid dynamical simulations central in air industry, wings tested.



Preliminary summary

Computations should enter basic science education

- Computation is a fundamental tool to gain new insights and should be included in our elementary teaching.
- Requires development of algorithmic thinking.
- Basic numerical methods should be part of the compulsory curriculum.
- The students should also learn to develop new numerical methods and adapt to new software tools.
- Requires more training than simple programming in a mathematics course.

Can we catch many birds with one stone?

- How can we include and integrate an algorithmic (computational) perspective in our basic education?
- Can this enhance the students' understanding of mathematics and science?
- Can it strengthen research based teaching?

What is needed?

Programming

A compulsory programming course with a strong mathematical flavour. *Should give a solid foundation in programming as a problem solving technique in mathematics*. Programming is understanding! The line of thought when solving mathematical problems numerically enhances algorithmic thinking, and thereby the students' understanding of the scientific process.

Mathematics and numerics

Mathematics is at least as important as before, but should be supplemented with development, analysis, implementation, verification and validation of numerical methods. Science ethics and better understanding of the pedagogical process, almost for free!

Sciences

Training in modelling and problem solving with numerical methods and visualisation, as well as traditional methods in Science courses, Physics, Chemistry, Biology, Geology, Engineering...

Implementation

Crucial ingredients

- Support from governing bodies (now priority 1 of the College of Natural Science at UOslo)
- Cooperation across departmental boundaries
- Willingness by individuals to give priority to teaching reform

Consensus driven approach.

Implementation in Oslo: The CSE project

What we do

- Coordinated use of computational exercises and numerical tools in most undergraduate courses.
- Help update the scientific staff's competence on computational aspects and give support (scientific, pedagogical and financial) to those who wish to revise their courses in a computational direction.
- Teachers get good summer students to aid in introducing computational exercises
- Develop courses and exercise modules with a computational perspective, both for students and teachers.
- Basic idea: mixture of mathematics, computation, informatics and topics from the physical sciences.

Interesting outcome: higher focus on teaching and pedagogical issues!!

Example of bachelor program, astrophysics

6th semester	**AST3210** Radiation I	**Choice**	**Choice**
5th semester	**FYS2160** Thermodynamics and statistical physics	**AST2120** The stars	**AST2210** Observational astronomy
4th semester	**FYS2140** Quantum physics	**Choice**	**EXPHIL03** Examen philosophicum
3rd semester	**FYS1120** Electromagnetism	**AST1100** Introduction to astrophysics / **GEF1100** The climate system	**MAT1120** Linear algebra
2nd semester	**FYS-MEK1110** Mechanics	**MEK1100** Vector calculus	**MAT1110** Calculus and linear algebra
1st semester	**MAT1100** Calculus	**MAT-INF1100** Modelling and computations	**INF1100** Introduction to programming with scientific applications
	10 credits	**10 credits**	**10 credits**

Table 2. Programme option for Astronomy in the bachelor programme Physics, Astronomy and Meteorology at UiO.

Example: Computations from day one

Differentiation

Three courses the first semester: MAT1100, MAT-INF1100 og INF1100.

- Definition of the derivative in MAT1100 (Calculus and analysis)

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}.$$

- Algorithms to compute the derivative in MAT-INF1100 (Mathematical modelling with computing)

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} + O(h^2).$$

- Implementation in Python in INF1100

```
def differentiate(f, x, h=1E-5):
     return (f(x+h) - f(x-h))/(2*h)
```

Example: Computations from day one

Differentiation and comparison with symbolic expressions

Combined with the possibility of symbolic calculations with *Sympy*, Python offers an environment where students and teachers alike can test many different aspects of mathematics and numerical mathematics, in addition to being able to verify and validate their codes. The following simple example shows how to extend the simple function for computing the numerical derivative with the possibility of obtaining the closed form or analytical expression

```
def differentiate(f, x, h=1E-5, symbolic=False):
    if symbolic:
        import sympy
        return sympy.lambdify([x], sympy.diff(f, x))
    else:
        return (f(x+h) - f(x-h))/(2*h)
```

Other Examples

Integration by Trapezoidal Rule

- Definition of integration in MAT1100 (Calculus and analysis).
- The algorithm for computing the integral vha the Trapezoidal rule for an interval $x \in [a, b]$

$$\int_a^b (f(x)dx \approx \frac{1}{2}\left[f(a) + 2f(a+h) + \cdots + 2f(b-h) + f(b)\right]$$

- Taught in MAT-INF1100 (Mathematical modelling)
- The algorithm is then implemented in INF1100 (programming course).

Typical implementation first semester of study

Integration by Trapezoidal Rule

```
from math import exp, log, sin
def Trapez(a,b,f,n):
    h = (b-a)/float(n)
    s = 0
    x = a
    for i in range(1,n,1):
        x = x+h
        s = s+ f(x)
    s = 0.5*(f(a)+f(b)) +s
    return h*s

def f1(x):
     return exp(-x*x)*log(1+x*sin(x))

a = 1;  b = 3; n = 1000
result = Trapez(a,b,f1,n)
print result
```

Symbolic calculations and numerical calculations in one code

Python offers an extremely versatile programming environment, allowing for the inclusion of analytical studies in a numerical program. Here we show an example code with the **trapezoidal rule** using **SymPy** to evaluate an integral and compute the absolute error with respect to the numerically evaluated one of the integral $4\int_0^1 dx/(1+x^2) = \pi$:

```
from math import *
from sympy import *
def Trapez(a,b,f,n):
    h = (b-a)/float(n)
    s = 0
    x = a
    for i in range(1,n,1):
        x = x+h
        s = s+ f(x)
    s = 0.5*(f(a)+f(b)) +s
    return h*s

#  function to compute pi
def function(x):
     return 4.0/(1+x*x)

a = 0.0;  b = 1.0; n = 100
result = Trapez(a,b,function,n)
```

Error analysis

The following extended version of the trapezoidal rule allows you to plot the relative error by comparing with the exact result. By increasing to 10^8 points one arrives at a region where numerical errors start to accumulate.

```
from math import log10
import numpy as np
from sympy import Symbol, integrate
import matplotlib.pyplot as plt
# function for the trapezoidal rule
def Trapez(a,b,f,n):
    h = (b-a)/float(n)
    s = 0
    x = a
    for i in range(1,n,1):
        x = x+h
        s = s+ f(x)
    s = 0.5*(f(a)+f(b)) +s
    return h*s
#  function to compute pi
def function(x):
     return 4.0/(1+x*x)
# define integration limits
a = 0.0;  b = 1.0;
# find result from sympy
# define x as a sumbol to be used by sumpy
```

Integrating numerical mathematics with calculus

The last example shows the potential of combining numerical algorithms with symbolic calculations, allowing thereby students and teachers to

- Validate and verify their algorithms.
- Including concepts like unit testing, one has the possibility to test and validate several or all parts of the code.
- Validation and verification are then included *naturally* and one can develop a better attitude to what is meant with an ethically sound scientific approach.
- The above example allows the student to also test the mathematical error of the algorithm for the trapezoidal rule by changing the number of integration points. The students get trained from day one to think error analysis.
- With an ipython notebook the students can keep exploring similar examples and turn them in as their own notebooks.

Additional benefits: A structured approach to solving problems

In this process we easily bake in

1. How to structure a code in terms of functions
2. How to make a module
3. How to read input data flexibly from the command line
4. How to create graphical/web user interfaces
5. How to write unit tests (test functions or doctests)
6. How to refactor code in terms of classes (instead of functions only)
7. How to conduct and automate large-scale numerical experiments
8. How to write scientific reports in various formats (LaTeX, HTML)

Additional benefits: A structure approach to solving problems

The conventions and techniques outlined here will save you a lot of time when you incrementally extend software over time from simpler to more complicated problems. In particular, you will benefit from many good habits:

1. New code is added in a modular fashion to a library (modules)
2. Programs are run through convenient user interfaces
3. It takes one quick command to let all your code undergo heavy testing
4. Tedious manual work with running programs is automated,
5. Your scientific investigations are reproducible, scientific reports with top quality typesetting are produced both for paper and electronic devices.

Learning outcomes three first semesters

Knowledge of basic algorithms

- Differential equations: Euler, modified Euler and Runge-Kutta methods (first semester)
- Numerical integration: Trapezoidal and Simpson's rule, multidimensional integrals (first semester)
- Random numbers, random walks, probability distributions and Monte Carlo integration (first semester)
- Linear Algebra and eigenvalue problems: Gaussian elimination, LU-decomposition, SVD, QR, Givens rotations and eigenvalues, Gauss-Seidel. (second and third semester)
- Root finding and interpolation etc. (all three first semesters)
- Processing of sound and images (first semester).

The students have to code several of these algorithms during the first three semesters.

Later courses

Later courses should build on this foundation as much as possible.

1. In particular, the course should not be too basic! There should be progression in the use of mathematics, numerical methods and programming, as well as science.
2. Computational platform: Python, fully object-oriented and allows for seamless integration of c++ and Fortran codes, as well as Matlab-like programming environment. Makes it easy to parallelize codes as well.

Coordination

- Teachers in other courses need therefore not use much time on numerical tools. Naturally included in other courses.

Realistic Pendulum

Classical pendulum with damping and external force

$$ml\frac{d^2\theta}{dt^2} + \nu\frac{d\theta}{dt} + mgsin(\theta) = Asin(\omega t).$$

Easy to solve numerically without classical simplification, and then visualize the solution. Done in first semester! Same equation for an RLC circuit

$$L\frac{d^2Q}{dt^2} + \frac{Q}{C} + R\frac{dQ}{dt} = V(t).$$

FYS1120 Electromagnetism, Third Semester

RLC circuit

Same equation as the pendulum for an RLC circuit

$$L\frac{d^2Q}{dt^2} + \frac{Q}{C} + R\frac{dQ}{dt} = V(t).$$

From the numerics, the students found the optimal parameters for studying experimentally chaos in an RLC circuit. Then they did the experiment.

More Examples from Physics Courses, 2-5 semester

Second-fourth semester

- Air resistance in two and three dimensions with quadratic velocity dependence.
- Launching a probe into a tornado
- Rocket launching with realistic parameters, gravity assist
- How to kick a football and model its trajectory.
- Planet motion and position of planets
- Magnetic fields with various geometries based on Biot-Savart's law
- Harmonic oscillations and various forms of electromagnetic waves.
- Combined effect of different potentials such as the electrostatic potential and the gravitational potential.
- Simple studies of atoms and molecules, and much more

First computational physics course

Late: Fifth semester, FYS3150 Computational Physics

The first computational physics course can then be used to summarize many of the gained insights about algorithms, mathematical models, physics etc. And direct the students to more advanced algorithms and applications like

- Monte carlo methods
- Parallelization
- Solving quantum mechanical problems by Variational Monte Carlo or other quantum mechanical methods
- Study phase transitions with for example the Ising and Potts model.
- Molecular dynamics simulations etc etc

Challenges...

.. and objections

Standard objection: computations take away the attention from other central topics in 'my course'.
CSE incorporates computations from day one, and courses higher up do not need to spend time on computational topics (technicalities), but can focus on the interesting science applications.

- To help teachers: Developed pedagogical modules which can aid university teachers. Own course for teachers.

Challenges and future plans

- The project depends crucially on few individuals.
- Need to get more teachers involved, not only good TAs.
- How to implement a CSE perspective in other programs like Chemistry, Molecular Biology, Biology, Engineering. New courses are being developed.
- Now a national pilot for other universities and regional colleges.

Key issue: modularization of topics and development of a 'technological platform' which glues together different modules

Which aspects are important for a successful introduction of CSE?

- Early introduction, programming course at beginning of studies linked with math courses and science and engineering courses.
- Crucial to learn proper programming at the beginning.
- Good TAs
- Choice of software.
- Textbooks and modularization of topics.
- Resources and expenses.
- Tailor to specific disciplines.
- Organizational matters.

What about life science/biology? Overarching questions

Which skills are needed by candidates in biology?

There is new demand for more

- quantitative methods & reasoning
- understanding data and phenomena via models
- creating *in silico* virtual labs

Challenge:

How to integrate such computing-based activities in the undergraduate programs when the students are *not* interested in mathematics, physics, and programming?

How to teach computing in biology?

Do we need to still follow the tradition and teach mathematics, physics, computations, chemistry, etc. in separate discipline-specific courses?

- Uninteresting to first study tools when you want to study biology
- Little understanding of what the tools are good for
- Minor utilization of tools later in biology

It's time for new thinking

- Just-in-time teaching: teach tools *when needed*
- Teach tools in the *context of biology*
- Emphasize development of *intuition and understanding*
- Base learning of the students' own *explorations in biology projects*
- Integrate lab work with computing tools

The pedagogical framework

Aim: Develop intuition about the scientific method

- Method: case-based learning
- Coherent problem solving *in biology* by integrating mathematics, programming, physics/chemistry, ...
- Starting point: data from lab or field experiments
- Visualize data
- Derive computational models directly from mathematical/intuitive *reasoning*
- Program model(s), fit parameters, compare with data
- Develop intuition and understanding based on
  - the principles behind the model
  - exploration of the model ("what if")
  - prediction of new experiments

Example 1: ecoli lab experiment

Observations of no of bacteria vs time in seconds, stored in Excel and written to a CVS file:

```
0,100
600,140
1200,250
1800,360
2400,480
3000,820
3600,1300
4200,1700
4800,2900
5400,3900
6000,7000
```

Visualize data

- Meet a text editor and a terminal window
- Very basic Unix

First program:

```
t = [0, 600, 1200, 1800, 2400, 3000, 3600,
     4200, 4800, 5400, 6000]
N = [100, 140, 250, 360, 480, 820, 1300, 1700, 2900, 3900, 7000]
import matplotlib.pyplot as plt
plt.plot(t, N, 'ro')
plt.xlabel('t [s]')
plt.ylabel('N')
plt.show()
```

Concepts must be introduced implicitly in a structured way

- Always identify new concepts
- Train new concepts in simplified ("trivial") problems

Concepts in the previous example:

- Lists or arrays of numbers
- Plotting commands
- Curve = function of time

Notice

The concept of a continuous function $N(t)$ is not necessary, just straight lines between discrete points on a curve.

Read data from file

```
import numpy as np
data = np.loadtxt('ecoli.csv', delimiter=',')
print data  # look at the format
t = data[:,0]
N = data[:,1]
import matplotlib.pyplot as plt
plt.plot(t, N, 'ro')
plt.xlabel('t [s]')
plt.ylabel('N')
plt.show()
```

Typical pattern:

The population grows faster and faster. Why? Is there an underlying (general) mechanism?

Lab journal

Use IPython notebook as lab journal.

How can we reason about the process?

1. Cells divide after T seconds on average (one generation)
2. $2N$ celles divide into twice as many new cells ΔN in a time interval Δt as N cells would: $\Delta N \propto N$
3. N cells result in twice as many new individuals ΔN in time $2\Delta t$ as in time Δt: $\Delta N \propto \Delta t$
4. Same proportionality wrt death (repeat reasoning)
5. Proposed model: $\Delta N = b\Delta t N - d\Delta t N$ for some unknown constants b (births) and d (deaths)
6. Describe evolution in discrete time: $t_n = n\Delta t$
7. Program-friendly notation: N at t_n is N^n
8. Math model: $N^{n+1} = N^n + r\Delta t\, N$ (with $r = b - d$)
9. Program model: `N[n+1] = N[n] + r*dt*N[n]`

How can we reason about the process?

1. Cells divide after T seconds on average (one generation)
2. $2N$ celles divide into twice as many new cells ΔN in a time interval Δt as N cells would: $\Delta N \propto N$
3. N cells result in twice as many new individuals ΔN in time $2\Delta t$ as in time Δt: $\Delta N \propto \Delta t$
4. Same proportionality wrt death (repeat reasoning)
5. Proposed model: $\Delta N = b\Delta t N - d\Delta t N$ for some unknown constants b (births) and d (deaths)
6. Describe evolution in discrete time: $t_n = n\Delta t$
7. Program-friendly notation: N at t_n is N^n
8. Math model: $N^{n+1} = N^n + r\Delta t\, N$ (with $r = b - d$)
9. Program model: `N[n+1] = N[n] + r*dt*N[n]`

How can we reason about the process?

1. Cells divide after T seconds on average (one generation)
2. $2N$ celles divide into twice as many new cells ΔN in a time interval Δt as N cells would: $\Delta N \propto N$
3. N cells result in twice as many new individuals ΔN in time $2\Delta t$ as in time Δt: $\Delta N \propto \Delta t$
4. Same proportionality wrt death (repeat reasoning)
5. Proposed model: $\Delta N = b\Delta t N - d\Delta t N$ for some unknown constants b (births) and d (deaths)
6. Describe evolution in discrete time: $t_n = n\Delta t$
7. Program-friendly notation: N at t_n is N^n
8. Math model: $N^{n+1} = N^n + r\Delta t\, N$ (with $r = b - d$)
9. Program model: `N[n+1] = N[n] + r*dt*N[n]`

How can we reason about the process?

1. Cells divide after T seconds on average (one generation)
2. $2N$ celles divide into twice as many new cells ΔN in a time interval Δt as N cells would: $\Delta N \propto N$
3. N cells result in twice as many new individuals ΔN in time $2\Delta t$ as in time Δt: $\Delta N \propto \Delta t$
4. Same proportionality wrt death (repeat reasoning)
5. Proposed model: $\Delta N = b\Delta t N - d\Delta t N$ for some unknown constants b (births) and d (deaths)
6. Describe evolution in discrete time: $t_n = n\Delta t$
7. Program-friendly notation: N at t_n is N^n
8. Math model: $N^{n+1} = N^n + r\Delta t\, N$ (with $r = b - d$)
9. Program model: `N[n+1] = N[n] + r*dt*N[n]`

How can we reason about the process?

1. Cells divide after T seconds on average (one generation)
2. $2N$ celles divide into twice as many new cells ΔN in a time interval Δt as N cells would: $\Delta N \propto N$
3. N cells result in twice as many new individuals ΔN in time $2\Delta t$ as in time Δt: $\Delta N \propto \Delta t$
4. Same proportionality wrt death (repeat reasoning)
5. Proposed model: $\Delta N = b\Delta t N - d\Delta t N$ for some unknown constants b (births) and d (deaths)
6. Describe evolution in discrete time: $t_n = n\Delta t$
7. Program-friendly notation: N at t_n is N^n
8. Math model: $N^{n+1} = N^n + r\Delta t\, N$ (with $r = b - d$)
9. Program model: `N[n+1] = N[n] + r*dt*N[n]`

How can we reason about the process?

1. Cells divide after T seconds on average (one generation)
2. $2N$ celles divide into twice as many new cells ΔN in a time interval Δt as N cells would: $\Delta N \propto N$
3. N cells result in twice as many new individuals ΔN in time $2\Delta t$ as in time Δt: $\Delta N \propto \Delta t$
4. Same proportionality wrt death (repeat reasoning)
5. Proposed model: $\Delta N = b\Delta t N - d\Delta t N$ for some unknown constants b (births) and d (deaths)
6. Describe evolution in discrete time: $t_n = n\Delta t$
7. Program-friendly notation: N at t_n is N^n
8. Math model: $N^{n+1} = N^n + r\Delta t\, N$ (with $r = b - d$)
9. Program model: `N[n+1] = N[n] + r*dt*N[n]`

How can we reason about the process?

1. Cells divide after T seconds on average (one generation)
2. $2N$ celles divide into twice as many new cells ΔN in a time interval Δt as N cells would: $\Delta N \propto N$
3. N cells result in twice as many new individuals ΔN in time $2\Delta t$ as in time Δt: $\Delta N \propto \Delta t$
4. Same proportionality wrt death (repeat reasoning)
5. Proposed model: $\Delta N = b\Delta t N - d\Delta t N$ for some unknown constants b (births) and d (deaths)
6. Describe evolution in discrete time: $t_n = n\Delta t$
7. Program-friendly notation: N at t_n is N^n
8. Math model: $N^{n+1} = N^n + r\Delta t\, N$ (with $r = b - d$)
9. Program model: `N[n+1] = N[n] + r*dt*N[n]`

How can we reason about the process?

1. Cells divide after T seconds on average (one generation)
2. $2N$ celles divide into twice as many new cells ΔN in a time interval Δt as N cells would: $\Delta N \propto N$
3. N cells result in twice as many new individuals ΔN in time $2\Delta t$ as in time Δt: $\Delta N \propto \Delta t$
4. Same proportionality wrt death (repeat reasoning)
5. Proposed model: $\Delta N = b\Delta t N - d\Delta t N$ for some unknown constants b (births) and d (deaths)
6. Describe evolution in discrete time: $t_n = n\Delta t$
7. Program-friendly notation: N at t_n is N^n
8. Math model: $N^{n+1} = N^n + r\Delta t\, N$ (with $r = b - d$)
9. Program model: `N[n+1] = N[n] + r*dt*N[n]`

How can we reason about the process?

1. Cells divide after T seconds on average (one generation)
2. $2N$ celles divide into twice as many new cells ΔN in a time interval Δt as N cells would: $\Delta N \propto N$
3. N cells result in twice as many new individuals ΔN in time $2\Delta t$ as in time Δt: $\Delta N \propto \Delta t$
4. Same proportionality wrt death (repeat reasoning)
5. Proposed model: $\Delta N = b\Delta t N - d\Delta t N$ for some unknown constants b (births) and d (deaths)
6. Describe evolution in discrete time: $t_n = n\Delta t$
7. Program-friendly notation: N at t_n is N^n
8. Math model: $N^{n+1} = N^n + r\Delta t\, N$ (with $r = b - d$)
9. Program model: `N[n+1] = N[n] + r*dt*N[n]`

How can we reason about the process?

1. Cells divide after T seconds on average (one generation)
2. $2N$ celles divide into twice as many new cells ΔN in a time interval Δt as N cells would: $\Delta N \propto N$
3. N cells result in twice as many new individuals ΔN in time $2\Delta t$ as in time Δt: $\Delta N \propto \Delta t$
4. Same proportionality wrt death (repeat reasoning)
5. Proposed model: $\Delta N = b\Delta t N - d\Delta t N$ for some unknown constants b (births) and d (deaths)
6. Describe evolution in discrete time: $t_n = n\Delta t$
7. Program-friendly notation: N at t_n is N^n
8. Math model: $N^{n+1} = N^n + r\Delta t\, N$ (with $r = b - d$)
9. Program model: `N[n+1] = N[n] + r*dt*N[n]`

The first simple program

Let us solve the difference equation in as simple way as possible, just to train some programming: $r = 1.5$, $N^0 = 1$, $\Delta t = 0.5$

```
import numpy as np

t = np.linspace(0, 10, 21)  # 20 intervals in [0, 10]
dt = t[1] - t[0]
N = np.zeros(t.size)

N[0] = 1
r = 0.5

for n in range(0, N.size-1, 1):
    N[n+1] = N[n] + r*dt*N[n]
    print 'N[%d]=%.1f' % (n+1, N[n+1])
```

The output

```
N[1]=1.2
N[2]=1.6
N[3]=2.0
N[4]=2.4
N[5]=3.1
N[6]=3.8
N[7]=4.8
N[8]=6.0
N[9]=7.5
N[10]=9.3
N[11]=11.6
N[12]=14.6
N[13]=18.2
N[14]=22.7
N[15]=28.4
N[16]=35.5
N[17]=44.4
N[18]=55.5
N[19]=69.4
N[20]=86.7
```

Parameter estimation

- We do not know r
- How can we estimate r from data?

We can use the difference equation with the experimental data

$$N^{n+1} = N^n + r\Delta t N^n$$

Say N^{n+1} and N^n are known from data, solve wrt r:

$$r = \frac{N^{n+1} - N^n}{N^n \Delta t}$$

Use experimental data in the fraction, say $t_1 = 600$, $t_2 = 1200$, $N^1 = 140$, $N^2 = 250$: $r = 0.0013$.

More sophisticated methods

Can do a nonlinear least squares parameter fit, but that is too advanced at this stage.

A program relevant for the biological problem

```
import numpy as np

# Estimate r
data = np.loadtxt('ecoli.csv', delimiter=',')
t_e = data[:,0]
N_e = data[:,1]
i = 2  # Data point (i,i+1) used to estimate r
r = (N_e[i+1] - N_e[i])/(N_e[i]*(t_e[i+1] - t_e[i]))
print 'Estimated r=%.5f' % r
# Can experiment with r values and see if the model can
# match the data better

T = 1200      # cell can divide after T sec
t_max = 5*T   # 5 generations in experiment
t = np.linspace(0, t_max, 1000)
dt = t[1] - t[0]
N = np.zeros(t.size)

N[0] = 100
for n in range(0, len(t)-1, 1):
    N[n+1] = N[n] + r*dt*N[n]

import matplotlib.pyplot as plt
plt.plot(t, N, 'r-', t_e, N_e, 'bo')
plt.xlabel('time [s]');  plt.ylabel('N')
plt.legend(['model', 'experiment'], loc='upper left')
plt.show()
```

Discuss the nature of such a model

- Write up all the biological factors that influence the population size of bacteria
- Understand that all such effects are merged into one parameter r
- Understand that the reasoning must be the same whether we have bacteria, animals or humans - this is a generic model! (even the interest rate in a bank follows the same model)

Discuss the limitations of such a model

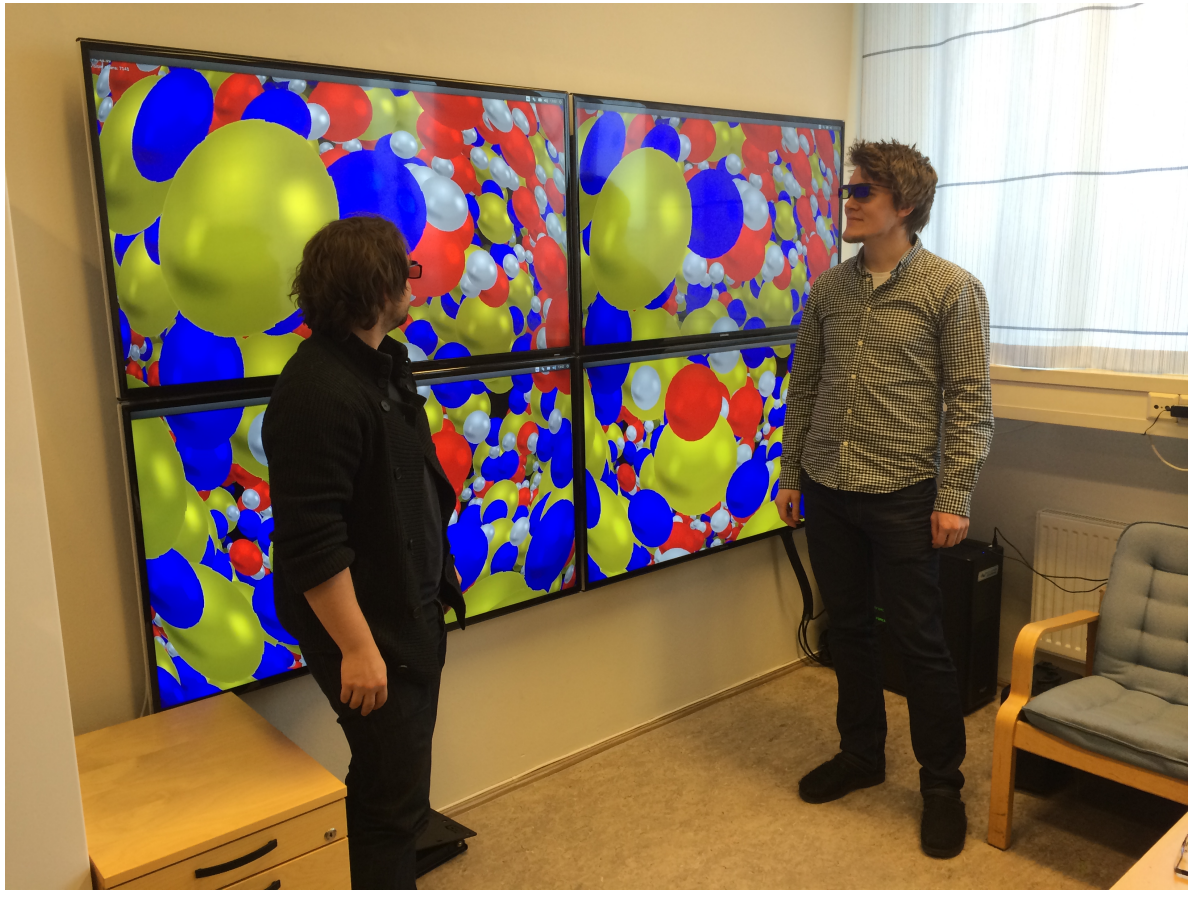
- How many bacteria in the lab after one month?
- Growth is restricted by environmental resources!
- Fix the model (logistic growth)
- Is the logistic model appropriate for a lab experiment?
- Find data to support the logistic model
  (it's a *very* simple model)

The pedagogical template (to be iterated!)

- Start with a real biological problem
- Be careful with too many new concepts
- Workflow:
  - data
  - visualization
  - patterns
  - modeling (*discrete*)
  - programming
  - testing
  - parameter estimation (difficult)
  - validation
  - prediction
- Make many small exercises that train the new concepts
- Repeat the case in a way that makes a complete understanding

Do we get better students?

Molecular dynamics visualization by two MSc students



Summary

- Make our research visible in early undergraduate courses, enhance research based teaching
- Possibility to focus more on understanding and increased insight.
- Impetus for broad cooperation in teaching.
- Strengthening of instruction based teaching (expensive and time-consuming).
- Give our candidates a broader and more up-to-date education with a problem-based orientation, often requested by potential employers.
- And perhaps the most important issue: does this enhance the student's insight in the Sciences?

People and links

- Hans Petter Langtangen, Computer Science
- Knut Morken, Mathematics
- Anders Malthe Sorensen and Arnt Inge Vistnes, Physics
- Oyvind Ryan, Mathematics
- Solveig Kristensen, Dean of Education
- Hanne Solna, Director of studies
- `http://www.mn.uio.no/english/about/collaboration/cse/`
- `http://www.mn.uio.no/english/about/collaboration/cse/national-group/computing-in-science-education.pdf`

More links

- Python and our first programming course, first semester course. Excellent new textbook by Hans Petter Langtangen, click here for the textbook or the online version
- Mathematical modelling course, first semester course. Textbook by Knut Morken to be published by Springer.
- Mechanics, second semester course. New textbook by Anders Malthe-Sorenssen, published by Springer, Undergraduate Lecture Notes in Physics
- Computational Physics I, fifth semester course. Textbook to be published by IOP in 2016, with online version