Chapter 1
Integrating a Computational Perspective in Physics Courses

Daniel Marcos Caballero and Morten Hjorth-Jensen

Abstract add text here later

1.1 Introduction

Many important recent advances in our understanding of the physical world have been driven by large-scale computational modeling and data analysis, for example, the 2012 discovery of the Higgs boson, the 2013 Nobel Prize in chemistry for computational modeling of molecules, and the 2016 discovery of gravitational Waves. Given the ubiquitous use in science and its critical importance to the future of science and engineering, scientific computing plays a central role in scientific investigations and is central to innovation in most domains of our lives. It underpins the majority of today's technological, economic and societal feats and we have entered an era in which huge amounts of data offer enormous opportunities. By 2020, it is also expected that one out of every two jobs in the STEM (Science, Technology, Engineering and Mathematics) fields will be in computing (Association for Computing Machinery, 2013).

These developments, needs and future challenges, as well as the developments which are now taking place within quantum computing, quantum information theory and data driven discoveries (data analysis and machine learning) will play an essential role in shaping future technological developments. Most of these developments require true cross-disciplinary approaches and bridge a vast range of temporal and spatial scales and include a wide variety of physical processes. To develop computational tools for such complex systems that give physically meaningful insights requires a deep understanding of approximation theory, high performance computing, and domain specific knowledge of the area one is modeling.

Computing competence represents thus a central element in scientific problem solving, from basic education and research to essentially almost all advanced problems in modern societies. And these competences are not limited to STEM fields only. The statistical analysis of big data sets and how to use machine learning algorithms belong nowadays to the set of tools needed by almost all disciplines, spanning from the Social Sciences, Law, Education to

Daniel Marcos Caballero
Department of Physics and Astronomy, Michigan State University, East Lansing, 48824 Michigan, USA and Department of Physics and Center for Computing in Science Education, University of Oslo, N-0316 Oslo, Norway, e-mail: caballero@pa.msu.edu

Morten Hjorth-Jensen
Department of Physics and Astronomy and National Superconducting Cyclotron Laboratory, Michigan State University, East Lansing, 48824 Michigan, USA and Department of Physics and Center for Computing in Science Education, University of Oslo, N-0316 Oslo, Norway, e-mail: hjensen@frib.msu.edu

the tradtional STEM fields and Life Science. Unfortunately, many of our students at both the undergraduate and the graduate levels are unprepared to use computational modeling, data science, and high performance computing, skills that are much valued by a broad range of employers. Furthermore, although many universities do offer compulsory programming courses in scientific computing, and physics departments offer one or more elective courses in computational physics, there is often not a uniform and coherent approach to the development of computing competences and computational thinking. This has in turn consequences for a systematic introduction and realization of computing skills and competences and pertaining learning outcomes.

The aim of this contribution is thus to present examples on how to introduce a computational perspective in basic undergraduate physics courses, basing ourselves on experiences made at the University of Oslo in Norway and now also at Michigan State University in the USA. In particular, we will present the **Computing in Science Education** project from the University of Oslo, a project which has evolved into a center of Excellence in education, the Center for Computing in Science Education. Similar initiatives and ideas are also being pursued at Michigan State University. The overarching aim is to strengthen the computing competences of students, with keywords such as the establishment of learning outcomes, how to develop assessment programs and course transformations by including computational projects and exercises in a coherent way. The hope is that these initiatives can also lead to a better understanding of the scientific method and scientific reasoning as well as providing new and deeper insights about the physics of a system.

This contribution is organized as follows. After these introductory remarks, we present briefly what we mean by computing and present possible learning outcomes which could be applied to a bachelor degree program in Physics. Thereafter we discuss possible paths on how to include and implement computational elements in central undergraduate Physics courses. We also discuss briefly how to assess various learning outcomes and how to develop a research program around this. Finally, in the last section we present our conclusions and perspectives.

1.2 Computing competences

The focus of this article is on Computing competences and how these help in enlarging the body of tools available to students and scientists alike, going well beyond classical tools taught in standard undergraduate courses in Physics and Mathematics. We will claim through various examples that computing allows for a more generic handling of problems, where focusing on algorithmic aspects **results in deeper insights** about scientific problems.

With **Computing** we will mean solving scientific problems using all possible tools, including symbolic computing, computers and numerical algorithms, experiments and analytical paper and pencil solutions. We will thus, deliberately, avoid a discussion of computing and computational physics in particular as something separate from theoretical physics and experimental physics. It is common in the scientific literature to encounter statements like *Computational physics now represents the third leg of research alongside analytical theory and experiments*. In selected contexts where say high-performance topics or specific computational methodologies play a central role, it may be meaningful to separate analytical work from computational studies. We will however argue strongly, in particular within an educational context, for a view where computing means solving scientific problems with all possible tools. Through various examples in this article we will show that a tight connection between standard analytical work, combined with various algorithms and a computational approach, helps in enhancing the students' understanding of the scientific method, hopefully providing deeper insights about the physics (or other disciplines).

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. Algorithms involving pen and paper are traditionally aimed at what we often refer to as continuous models, of which only few can be solved analytically. The number of important differential equations in Physics that can be solved analytically are rather few, limiting thereby the set of problems that can be addressed in order to deepen a student's insights about a particular Physics case. On the other hand, the 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 we can address much larger classes of problems with discrete models, often also by simpler and more generic methodologies. In the next section we will present several examples thereof. A typical case is that where an eigenvalue problem can allow students to study the analytical solution as well as moving to an interacting quantum mechanical case where no analytical solution exists. By merely changing the diagonal matrix elements one can solve problems that span from classical mechanics and fluid dynamics to quantum mechanics and statistical physics. Using essentially the same algorithm one can thus study physics cases that are covered by several courses, allowing thereby the teachers to focus more on the physical systems of interest.

There are several advantages in introducing Computing in basic Physics courses. It allows Physics teachers to bring important elements of scientific methods at a much earlier stage in our students' education. Many advanced simulations used in Physics research can easily be introduced, via various simplifications, in introductory Physics courses, enhancing thereby the set of problems studied by the students, see also the next section. Computing gives thus university teachers a unique opportunity to enhance students' insights about physics and how to solve scientific problems, and it gives the students the skills and abilities that are asked for by society. Computing allows for solving more realistic problems earlier and provides an excellent training of creativity as well as enhancing the understanding of abstractions and generalizations. Furthermore, Computing can decrease the need for special tricks and tedious algebra, and shifts the focus to problem definition, visualization, and "what if" discussions. Finally, if the setup of undergraduate courses is properly designed, with a synchronization with Mathermatics and Computational Science courses, Computing can trigger further insights in Mathematics and other disciplines.

On the part of students, an increase in Computing competences involves being able to:

- understand how algorithms are used to solve mathematical problems,
- derive, verify, and implement algorithms,
- understand what can go wrong with algorithms,
- use these algorithms to construct reproducible scientific outcomes and to engage in science in ethical ways, and
- think algorithmically for the purposes of gaining deeper insights about scientific problems.

1.3 Learning Outcomes and Assessment Programs

An essential element in designing a synchronization of Computing in various Physics (and other disciplines as well) courses is a proper definition of learning outcomes, as well as the development of assessment programs and possibly a pertinent research program on Physics education. Having a strong Physics Education group which can define a proper research program is an essential part of such an endeavor. Michigan State University has a strong Physics Education group involved in such research programs. Similarly, the University of Oslo, with its recently established center of excellence in education, is defining a research program on the assessment of Computing in Science education.

Physics, together with basic Mathematics and Computational Science courses, is at the undergraduate level presented in a very homogeneous way worldwide. Most universities offer more or the less the same topics and courses, starting with Mechanics and Classical Mechanics, Waves, Electromagnetism, Quantum Physics and Quantum Mechanics and ending with Statistical Physics. Similalry, during the last year of the Bachelor's degree one finds elective courses on Computational Physics and Mathematical Methods in Physics, in addition to a selection of compulsory introductory laboratory courses. In Computer Science, most Physics undergraduate programs have now a compulsory introductory course in Scientific programming. Here, one encounters frequently Python as the default programming language. Similalry, one finds almost the same topics covered by the basic Mathematics courses required for a Physics degree, from basic calculus to linear algebra, differential equations and real analysis. Many Mathematics departments and/or Computational Science departments offer courses on Numerical Mathematics that are based on the first course in programming.

These developments have taken place during the last decade and several universities are attempting now at including a more coherent computational perspective to our basic education in Natural Science. In order to achieve this, it is important to develop a strategy where the introduction of computational elements are properly synchronized between Physics, Mathematics and Computational Science courses. This allows Physics teachers to focus more on the relevant Physics. To develop learning outcomes plays a central role. An additional benefit of properly developed learning outcomes is the stimulation of cross-department collaborations as well as an increased awareness about what is being taught in different courses. Here we list several possibilities, starting with some basic algorithms which can be taught in Mathematics and Computational Science courses. We end with a discussion of possible learning outcomes for central undergraduate Physics courses

General Learning Outcomes for Computing Competence.

Here we articulate high-level learning outcomes that we expect students to develop through comprehensive and coordinated instruction in numerical methods over the course of their undergraduate program. These learning outcomes are different from specific learning goals in that the former reference the end state that we aim for students to achieve. The latter references the specific knowledge, tools, and practices with which students should engage and discusses how we expect them to participate in that work.

Numerical algorithms form the basis for solving science and engineering problems with computers. An understanding of algorithms does not itself serve as an understanding on computing, but it is a necessary step along the path. Through comprehensive and coordinated instruction, we aim for students to have developed:

- A deep understanding of the most fundamental algorithms for linear algebra, ordinary and partial differential equations and optimization methods

- Numerical integration: Trapezoidal and Simpson's rule, multidimensional integrals
- Random numbers, random walks, probability distributions, Monte Carlo integration and Monte Carlo methods
- Root finding and interpolation etc.
- Machine Learning algorithms
- Statistical Data Analysis and handling of data sets
- A working knowledge of advanced algorithms and how they can be accessed in available software
- An understanding of approximation errors and how they can present themselves in different problems
- The ability to apply fundamental and advanced algorithms to classical model problems as well as real-world problems as well to assess the uncertainty of their results

Later courses should build on this foundation as much as possible and in designing learning outcomes and course contents one should make sure that there is a progression in the use of Mathematics, numerical methods and programming, as well as the contents of various Physics courses. This means also that teachers in other courses do not need to use much time on numerical tools since these are naturally included in other courses.

Learning Outcomes for Symbolic Computing.

Symbolic computing is a helpful tool for addressing certain classes of problems where a functional representation of the solution (or part of the solution) is needed. Through engaging with symbolic computing platforms, we aim for students to have developed:

- A working knowledge of at least one computer algebra system (CAS)
- The ability to apply a CAS to perform classical mathematics including calculus, linear algebra and differential equations
- The ability to verify the results produced by the CAS using some other means

Learning Outcomes for Programming.

Programming is a necessary aspect of learning computing for science and engineering. The specific languages and/or environments that students learn are less important than the nature of that learning (i.e., learning programming for the purposes of solving science problems). By numerically solving science problems, we expect students to have developed (these are possible examples):

- An understanding of programming in a high-level language (e.g., MATLAB, Python, R).
- An understanding of programming in a compiled language (e.g., Fortran, C, C++).
- The ability to to implement and apply numerical algorithms in reusable software that acknowledges the generic nature of the mathematical algorithms.
- A working knowledge of basic software engineering elements including functions, classes, modules/libraries, testing procedures and frameworks, scripting for automated and reproducible experiments, documentation tools, and version control systems (e.g., Git).
- An understanding of debugging software, e.g., as part of implementing comprehensive tests.

Learning Outcomes for Mathematical Modeling.

Preparing a problem to be solved numerically is a critical step in making progress towards an eventual solution. By providing opportunities for students engage in modeling, we aim for them to develop the ability to solve real problems from applied sciences by:

- Deriving computational models from basic principles in physics and articulating the underlying assumptions in those models,
- Constructing models with dimensionless and/or scaled forms to reduce and simplify input data, and
- Interpreting the model's dimensionless and/or scaled parameters to increase their understanding of the model and its predictions

Learning Outcomes for Verification.

Verifying a model and the resulting outcomes it produces are essential elements to generating confidence in the model itself. Moreover, such verifications provide evidence that the work is reproducible. By engaging in verification practices, we aim for students to develop:

- An understanding of how to program testing procedures
- Knowledge of testing/verification methods including the use of:
  - Exact solutions of numerical models
  - Classical analytical solutions including asymptotic solutions
  - Computed asymptotic approximation errors (i.e., convergence rates)
  - Unit tests and step-wise construction of tests to aid debugging.

Learning outcomes for presentation of results.

The results of a computation need to be communicated in some format (i.e., through figures, posters, talks, and other forms of written and oral communication). Computation affords the experience of presenting original results quite readily. Through their engagement with presentations for their findings, we aim for students to develop:

- The ability to make use of different visualization techniques for different types of computed data
- The ability to present computed results in scientific reports and oral presentations effectively
- A working knowledge of the norms and practices for scientific presentations in various formats (i.e., figures, posters, talks, and written reports)

The above learning goals and outcomes are of a more generic character. What follows here are specific algorithms that occur frequently in scientific problems. The implementation of these algorithms in various Physics courses, together with problem and project solving, is a way to implement large fractions of the above learning goals.

Central Algorithms.

The following mathematical formulations of problems from the physical sciences play a prominent role and should be reflected in how we teach physics:

- Ordinary differential equations
    1. Euler, modified Euler, Verlet and Runge-Kutta methods with applications to problems in electromagnetism, methods for theoretical physics, quantum mechanics and mechanics.
- Partial differential equations
    1. Diffusion in one and two dimensions (statistical physics), wave equation in one and two dimensions (mechanics, electromagnetism, quantum mechanics, methods for theoretical physics) and Laplace's and Poisson's equations (electromagnetism).
- Numerical integration
    1. Trapezoidal and Simpson's rule and Monte Carlo integration. Applications in statistical physics, methods of theoretical physics, electromagnetism and quantum mechanics.
- Statistical analysis, random numbers, random walks, probability distributions, Monte Carlo integration and Metropolis algorithm. Applications to statistical physics and laboratory courses.
- Linear Algebra and eigenvalue problems.
    1. Gaussian elimination, LU-decomposition, eigenvalue solvers, and iterative methods like Jacobi or Gauss-Seidel for systems of linear equations. Important for several courses, classical mechanics, methods of theoretical physics, electromagnetism and quantum mechanics.
- Signal processing
    1. Discrete (fast) Fourier transforms, Lagrange/spline/Fourier interpolation, numeric convolutions & circulant matrices, filtering. Applications in electromagnetics, quantum mechanics, and experimental physics (data acquisition)
- Root finding techniques, used in methods for theoretical physics, quantum mechanics, electromagnetism and mechanics.
- Machine Learning algorithms and Statistical Data Analysis, relevant for laboratory courses.

In order to achieve a proper pedagogical introduction of these algorithms, it is important that students and teachers alike see how these algorithms are used to solve a variety of physics problems. The same algorithm, for example the solution of a second-order differential equation, can be used to solve the equations for the classical pendulum in a mechanics course or the (with a suitable change of variables) equations for a coupled RLC circuit in the electromagnetism course. Similarly, if students develop a program for studies of celestial bodies in the mechanics course, many of the elements of such a program can be reused in a molecular dynamics calculation in a course on statistical physics and thermal physics. The two-point boundary value problem for a buckling beam (discretized as an eigenvalue problem) can be reused in quantum mechanical studies of interacting electrons in oscillator traps, or just to study a particle in a box potential with varying depth and extension.

In order to aid the introduction of computational exercises and projects, there is a strong need to develop educational resources for this. The PICUP project, Partnership for Integration of Computation into Undergraduate Physics, develops resources for teachers and students on the integration of computational material. We strongly recommend these resources. Physics is an old discipline, with a large wealth of established analytical exercises and projects. In fields like mechanics, we have centuries of pedagogical developments, with a strong emphasis on developing analytical skills. The majority of physics teachers are well familiar with this and in order to see how computing can enlarge this body of exercises and projects, and hopefully add additional insights to the physics behind various phenomena, we find it important to develop a large body of computational examples.

Central Tools and Programming Languages.

We will strongly recommend that Python is used as the high-level programming language. Other high-level environments like Mathematica and Matlab can also be presented and offered as special courses. This means that students can apply their knowledge from the basic programming course offered by most universities. Many university courses in programming make nowadays use of Python, and extend their computational knowledge in various physics classes. We recommend strongly that the following tools are used

1. Jupyter and ipython notebook.
2. Version control software like git and repositories likeGitHub
3. Other typsetting tools like LaTeX.
4. Unit tests and using existing tools for unit tests. Python has extensive tools for this

The notebooks can be used to hand in exercises and projects. They can provide the students with experience in presenting their work in the form of scientific/technical reports.

Version control software allows teachers to bring in reproducibility of science as well as enhancing collaborative efforts among students. Using version control can also be used to help students present benchmark results, allowing others to verify their results. Unit testing is a central element in the development of numerical projects, from microtests of code fragments, to intermediate merging of functions to final test of the correctness of a code.

Suggested Learning Goals and Computational Topics for Specific Physics Courses.

For a bachelor degree in Physics, it is now more and more common to require a compulsory programming course, typically taught during the first two years of undergraduate studies. The programming course, together with Mathematics courses, lays the foundation for the use of computational exercises and projects in various Physics courses. Based on this course, and the various mathematics courses included in a Physics degree, there is a unique possibility to incorporate computational exercises and projects in various Physics courses, without taking away the attention from the basic physics topics to be covered.

What follows below is a suggested listed of possible learning outcomes. The list is by no means exhaustive and is mainly meant as a guideline of what can be included.

Mechanics/Classical Mechanics.

After completing a course on Mechanics/Classical Mechanics, students should be able to:

- Represent numbers, complex numbers, vectors, matrices as variables and do simple and appropriate mathematics on these
- Access constants and physical constants defined in libraries
- Construct and slice arrays
- Use functions defined in relevant libraries
- Write functions to perform specialized tasks
- determine the root of an algebraic equation numerically using Newton’s method
- explain Newton’s method for finding roots
- solve a system of algebraic equations using Gaussian elimination numerically
- explain Gaussian elimination
- solve 1st Order, 2nd Order and Coupled ODEs numerically using Euler-Cromer, Verlet, and/or Runge-Kutta algorithms
- explain the differences between each of the above algorithms

- compare the quality of simulations (i.e., number of iterations, step size, and error control) of particle motion that use different motion prediction algorithms
- plot solutions

Thermal and Statistical Physics.

After completing a course in Thermal and Statistical Physics, students should be able to:

- use central probability distributions and their relation to various expectation values
- simulate and visualize central probability distributions like the uniform distributions, the exponential distribution and the normal (Gaussian distribution)
- use concept from statistics to understand central ensembles like the microcanonical and the canonical ensembles
- simulate Markov processes and understand the links with the process of diffusion (Fick's and Fourier's laws) and the concept of most likely states
- understand how to simulate stochastic variables using random number generators
- understand central algorithms like the Metropolis algorithm to simulate systems in statistical physics
- understand the physics of various phases and phase transitions
- study systems like ideal gas and ideal crystals analytically and numerically
- understand the link between various ensembles; both mathematical and physical links
- be able to simulate phase transitions via models like the Ising class of models
- be able to perform molecular dynamics calculations using the velocity Verlet algorithm and simulate phase transitions and visualize and analyze results using realistic interactions.

Mathematical Methods in Physics.

After completing a course on Mathematical Methods in Physics, students should be able to:

- Understand how to discretize differential equations and understand the mathematical truncation errors
- Understand errors in mathematical algorithms
- be able to rewrite differential equations using methods from linear algebra
- know important algorithms for solving eigenvalue problems
- be able to solve initial value and boundary value problems analytically and numerically
- know central algorithms for solving eigenvalue problems
- Solve differential equations numerically and compare with analytical solutions
- understand important orthogonal polynomials like Legendre, Hermite and Laguerre. Be able to set up their recursion relations and visualize the polynomials.
- Understand Fourier transforms and algorithms like Fast Fourier transform
- Know tools to analyze time series

Many of these algorithms can be discussed and used in the other courses discussed here.

Laboratory course in Physics.

After completing a Laboratory course in Physics, students should be able to:

- read data from CSV files constructed by oscilloscope or LabView program
- rescale and plot these data

- smooth, filter, and transform data as needed for specific experiments
- compute numerical derivatives or integrals of data as needed for specific experiments
- construct a linear fit, fit to exponentials, and a nonlinear fit to a sum of Gaussians or Lorentzians as needed for specific experiments
- numerically determine location of peaks in data
- perform a fast Fourier transform and construct a power spectrum of data as needed
- make histograms from a single column of data
- calculate mean, standard deviation of data
- compare histogram to Poisson distribution with the same mean
- numerically count number of peaks or dips in a spectrum
- plot data and fits on same figure
- numerically determine goodness of a fit using residuals
- propagating uncertainty when combining fit parameters using the covariance matrix

Quantum Mechanics/ and/or Quantum Physics.

After completing a course in Quantum mechanics/Physics, students should be able to:

- be able to visualize the solutions of quantum mechanical problems, both stationary and time-dependent problems
- be able to scale the equations properly and understand the meaning of natural length scales, from the Bohr radius to simple harmonic oscillator problems with frequency dependent length scale. The same scaling procedure is used to derive the analytical solutions for several single-particle problems.
- use numerical methods for solving a large variety of one-dimensional differential equations with two-point boundary value problems. For many cases one can compare directly with standard analytical solutions like the hydrogen-like problems or the harmonic oscillator.
- Verification of numerical solutions with analytical results.
- be able to rewrite Schroedinger's equation as an eigenvalue problem and use numerical eigenvalue methods for computing a single particle confined in a one-dimensional infinite potential and compare with analytical results. This problem is the same as the eigenvalue problem of a buckling beam, which can be used the in the mechanics and mathematical methods course.
- use the the same eigenvalue solvers to study a single particle confined to move in a potential well with a finite depth and extension. Study both bound and unbound states and explore the numerical solutions as functions of the potential depth and the extension of the potential.
- The same codes can be used to solve the hydrogen atom and the one-dimensional harmonic oscillator. This part allows for comparison with analytical results.
- Visualize the probability distributions for electrons (or other one-particle problems) confined to move in hydrogen*like and harmonic oscillator like problems. Study the probability distributions for ground and excited states. Discuss unbound states with a finite potential well.
- Use the same codes to study double well potentials. These are problems of great interest in solid state physics.
- Rewrite a two-electron (or two-particle problem) problem in terms of the relative and center-of-mass motion and study the role of repulsive Coulomb forces) for electrons (or other fermions) trapped
- Visualize and compute tunneling phenomena for various potentials.

- Introduce the variational principle and introduce variational Monte Carlo methods to study one-particle problems and compare these with analytical results and numerical results from differental equation solvers. The Metropolis algorithm discussed in Statistical physics can be reused here. Gives the students a further understanding of statistics related topics, including random number generators, probability distributions, mean values and standard deviations. These topics could also be discussed in PHY415. The students will then see central algorithms being used in different physics settings.

Electromagnetism.

After completing a course in Electromagnetism, students should be able to:

- use symbolic computing tools to determine the gradient of various scalar fields
- use symbolic computing to determine the divergence and curl of various vector fields
- represent the vector (e.g., electric) field visually using vector plots and stream plots
- represent a 2D scalar (i.e., potential) field visually using 2D contour plots and 3D surface plots
- apply motion prediction algorithms Euler, Verlet, and Runge-Kutta to model the motion of charged particles in electric and magnetic fields
- compare the quality of simulations (i.e., number of iterations, step size, and error control) of charged particle motion that use different motion prediction algorithms
- apply Coulomb’s law iteratively to determine the electric field produced by a given charge distribution
- apply Biot-Savart’s law iteratively to determine the magnetic field produced by a given current distribution
- explain how the application of superposition iteratively gives rise to approximate field solutions
- explain how the simple relaxation algorithm works (i.e., iteratively averaging neighboring points) and how it is derived from the properties of the solutions to Laplace’s equation
- apply this simple relaxation method to find the potential for 1D and 2D electrostatic situations where Laplace’s equation is satisfied
- explain how to use finite-difference methods to recast Poisson’s equation into a discrete formulation and how the resulting discretized form compares with the simple relaxation method (i.e., iteratively averaging neighboring points)
- apply the Jacobi and Gauss-Seidel methods to solve 2D Laplace and Poisson problems including graphing the results in three dimensions
- explain the differences between the Jacobi and Gauss-Seidel methods and how these methods are connected to the derivation using finite differencing
- Compare the quality of the simulations (i.e., number of iterations, step size, and error control) that employ the Jacobi method and the Gauss-Seidel method

Advanced Computational Physics Courses.

Towards the end of undergraduate studies it is useful to offer a course which focuses on more advanced algorithms and presents compiled languages like C++ and Fortran, languages our students will meet in actual research. Furthemore, such a course should offer more advanced projects which train the students in actual research, developing more complicated programs and working on larger projects. The course could cover

- C++ and/or Fortran programming

- Numerical derivation and integration
- Random numbers and Monte Carlo integration
- Monte Carlo methods in statistical physics
- Quantum Monte Carlo methods
- Statistical Data Analysis and Machine Learning
- Linear algebra and eigenvalue problems
- Non-linear equations and roots of polynomials
- Ordinary differential equations
- Partial differential equations
- Parallelization of codes
- High-performance computing aspects and optimization of codes

Physics Education Research and Computing in Science Education.

The introduction of computational elements in the various courses should be, if possible, strongly integrated with ongoing research on physics education. The Physics and Astronomy department at MSU is in a unique position due to its strong research group in physics education, the PERL group. Together with the Center for Computing in Science Education at the University of Oslo, we are now in the process of establishing new assessments and new assessment methods that address several issues associated with integrating computation into science courses. The issues include but are not limited to how well students learn computing, what new insights students gain about the specific science through computing, and how students' affective states (e.g., motivation to learn, computational self-efficacy) are affected by computing. Broadly speaking, these assessments should provide deeper insights into the integration of computing in science education in general as well as provide a structured framework for assessment of our efforts and a basis for systematic studies of student learning.

The central questions that our research must address are

1. How can we assess the effect of integrating computing into science curricula on a variety of learned-centered constructs including computational thinking,

motivation, self-efficacy and science identity formation,

1. how should we structure assessments to ensure valid, reliable and impactful assessment, which provides useful information to our program and central partners, and finally
2. how can the use of these structured assessments improve student outcomes in teacher-, peer-, and self-assessment.

Addressing these questions requires a combination of qualitative techniques to construct the focus of these assessments, to build assessment items and to develop appropriate assessment methods, and quantitative techniques, including advanced statistical analysis to ensure validity and reliability of the proposed methods as well as to analyze the resulting data.

The learning objectives and learning outcomes for computational methods developed as part of the first objective form parts of the basis for the assessment program, and we will also investigate the assessment of non-content learning goals such as self-efficacy and identity formation.

The effect of integration of computational methods into basic science courses have been sparsely studied, primarily because the practice is sparse. Further progress depends now on the development of assessments that can be used for investigative, comparative and/or longitudinal studies and to establish best practices in this emerging field. Some assessments will be developed for specific courses, but we will aim for broad applicability across institutions.

1.4 Examples on how to Include Computing in Physics Undergraduate Programs

Having defined possible learning outcomes, we would like now to present some examples which reflect the discussions above. These examples are taken from various courses at the University of Oslo. Since 2003, first via the Computing in Science Education project and now through the recently established center of excellence in education Center for Computing in Science Education, Computing has been introduced across the disciplines in a synchronized way.

Central elements here are a compulsory programming course with a strong mathematical flavour. This course gives a solid foundation in programming as a problem solving technique in mathematics. The line of thought when solving mathematical problems numerically enhances algorithmic thinking, and thereby the students' understanding of the scientific process. Secondly, Mathematics is at least as important as before, but should be supplemented with development, analysis, implementation, verification and validation of numerical methods. Finally, these methods are used in modeling and problem solving with numerical methods and visualisation, as well as traditional methods in various Science courses, from the Physical Sciences to Life science.

Crucial ingredients for the success of the Computing in Science Education project has been the support from governing bodies as well as extensive cooperations across departmental boundaries. And finally the willingness of several university teachers and researchers to give priority to teaching reform.

In addition to the above, over the years we have

- 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. This may include the organization of courses for university teachers.
- Teachers get good summer students to aid in developing and introducing computational exercises
- Developed courses and exercise modules with a computational perspective, both for students and teachers. Several new textbooks have been developed, from the basic Mechanics course to a course in Statistical physics.

Basic idea has been a mixture of mathematics, computational science and topics from the physical sciences. One interesting outcomes is a higher focus on teaching and pedagogical topics.

The Physics Undergraduate Program at the University of Oslo.

The layout of the Physics bachelor's degree program at the University of Oslo is given by the following table

6th Semester	Elective	Elective	Elective
5th Semester	FYS2160 Statistical Physics	FYS3110 Quantum Mechanics	Elective
4th Semeters	FYS2130 Waves and Motion	FYS2140 Quantum Physics	FYS2150 Physics Laborato
3rd Semester	FYS1120 Electromagnetism	MAT1120 Linear Algebra	AST2000 Introduction to Astro
2nd Semester	FYS-MEK1100 Mechanics	MEK1100 Vector Calculus	MAT1110 Calculus and Linear
1st Semester	MAT 1100 Calculus	MAT-INF1100 Modeling and Computations	IN1900 Introduction to Programming with S
Credits	10 ECTS	10 ECTS	10 ECTS

In the first semester the students encounter the first level of syncronization between the programming and the two mathematics courses. As an example consider integration by Trapezoidal Rule. Integral calculus is typically discussed first in the Calculus course MAT1100. Thereafter, the algorithm for computing the integral vha the Trapezoidal rule for an interval $x \in [a,b]$

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

is discussed and developed in MAT-INF1100, the Modeling and Computations course that serves as an intermediate step between the standard Calculus course and the programming course. Finally, the algorithm is implemented in IN1900 Introduction to Programming with Scientific Applications. We show here a typical Python code which exemplifies this.

```
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)
```

Here we have defined an integral given by $I = \int_1^3 dx \exp(-x*x)*\log(1+x*\sin(x))$.

Coming back to your learning outcomes, we would like to emphasize that 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** again using **SymPy** to evaluate an integral and compute the absolute error with respect to the numerically evaluated one of the integral $\int_0^1 dx x^2 = 1/3$:

```
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 x*x

a = 0.0; b = 1.0; n = 100
result = Trapez(a,b,function,n)
print("Trapezoidal rule=", result)
# define x as a symbol to be used by sympy
x = Symbol('x')
exact = integrate(function(x), (x, 0.0, 1.0))
```

```
print("Sympy integration=", exact)
# Find relative error
print("Relative error", abs((exact-result)/exact))
```

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 x*x
# define integration limits
a = 0.0; b = 1.0;
# find result from sympy
# define x as a symbol to be used by sympy
x = Symbol('x')
exact = integrate(function(x), (x, a, b))
# set up the arrays for plotting the relative error
n = np.zeros(9); y = np.zeros(9);
# find the relative error as function of integration points
for i in range(1, 8, 1):
    npts = 10**i
    result = Trapez(a,b,function,npts)
    RelativeError = abs((exact-result)/exact)
    n[i] = log10(npts); y[i] = log10(RelativeError);
plt.plot(n,y, 'ro')
plt.xlabel('n')
plt.ylabel('Relative error')
plt.show()
```

The last example shows the potential of combining numerical algorithms with symbolic calculations, allowing thereby students and teachers to validate their algorithms. With concepts like unit testing, one has the possibility to test and verify 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. The figure here shows clearly the region where the relative error starts increasing. The mathematical error which follows the Trapezoidal rule goes as $O(h^2)$ where h is the chosen numerical step size. Before numerical round-off errors and loss of numerical precision kicks in (near $h \sim 10^{-7}$) we see that the relative error in the log-log plot has a slope which follows the mathematical error. There are several additional benefits here. 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)

The conventions and techniques outlined here will save students a lot of time when one extends incrementally software over time, from simpler to more complicated problems. In particular, the student can 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.

From Mathematics to Physics.

1. Ordinary differential equations (ODE): RLC circuit
2. ODE: Classical pendulum
3. ODE: Solar system
4. and many more cases

Can use essentially the **same algorithms to solve these problems**, either some simple modified Euler algorithms or some Runge-Kutta class of algorithms or perhaps the so-called Verlet class of algorithms. **Algorithms students use in one course can be reused in other courses**. Mechanics and electromagnetism, initial value problems.

When properly scaled, these equations are essentially the same. Scaling is important.

Classical pendulum with damping and external force as it could appear in a mechanics course (PHY 321)

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

Easy to solve numerically and then visualize the solution. Almost the same equation for an RLC circuit in the electromagnetism course (PHY 482)

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

Classical pendulum equations with damping and external force

$$\frac{d\theta}{d\hat{t}}=\hat{v},$$

and

$$\frac{d\hat{v}}{d\hat{t}}=Acos(\hat{\omega}\hat{t})-\hat{v}\xi-\sin(\theta),$$

with $\omega_0=\sqrt{g/l}$, $\hat{t}=\omega_0 t$ and $\xi=mg/\omega_0\nu$.

The RLC circuit

$$\frac{dQ}{d\hat{t}}=\hat{I},$$

and

$$\frac{d\hat{I}}{d\hat{t}} = Acos(\hat{\omega}\hat{t}) - \hat{I}\xi - Q,$$

with $\omega_0 = 1/\sqrt{LC}$, $\hat{t} = \omega_0 t$ and $\xi = CR\omega_0$.

The equations are essentially the same. **Great potential for abstraction**.

Two-point boundary value problems and scaling.

These physics examples can all be studied using almost the same types of algorithms, simple eigenvalue solvers and Gaussian elimination with the same starting matrix!

1. A buckling beam and Toeplitz matrices (mechanics and mathematical methods), eigenvalue problems
2. A particle in an infinite potential well, quantum eigenvalue problems
3. A particle (or two) in a general quantum well, quantum eigenvalue problems
4. Poisson's equation in one dim, linear algebra (electromagnetism)
5. The diffusion equation in one dimension (Statistical Physics), linear algebra
6. and many other cases

A buckling beam, or a quantum mechanical particle in an infinite well.

This is a two-point boundary value problem

$$R\frac{d^2u(x)}{dx^2} = -Fu(x),$$

where $u(x)$ is the vertical displacement, R is a material specific constant, F the force and $x \in [0,L]$ with $u(0) = u(L) = 0$.

Scale equations with $x = \rho L$ and $\rho \in [0,1]$ and get (note that we change from $u(x)$ to $v(\rho)$)

$$\frac{d^2v(\rho)}{dx^2} + Kv(\rho) = 0,$$

a standard eigenvalue problem with $K = FL^2/R$.

If you replace $R = -\hbar^2/2m$ and $-F = \lambda$, we have the quantum mechanical variant for a particle moving in a well with infinite walls at the endpoints.

Discretize the second derivative and the rhs

$$-\frac{v_{i+1} - 2v_i + v_{i-i}}{h^2} = \lambda v_i,$$

with $i = 1,2,\dots,n$. We need to add to this system the two boundary conditions $v(0) = v_0$ and $v(1) = v_{n+1}$. The so-called Toeplitz matrix (special case from the discretized second derivative)

$$\mathbf{A} = \frac{1}{h^2}\begin{bmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & -1 & 2 & -1 & & \\ & \dots & \dots & \dots & \dots & \dots \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 \end{bmatrix}$$

with the corresponding vectors $\mathbf{v} = (v_1, v_2, \dots, v_n)^T$ allows us to rewrite the differential equation including the boundary conditions as a standard eigenvalue problem

$$\mathbf{Au} = \lambda\mathbf{v}.$$

The Toeplitz matrix has analytical eigenpairs!! Adding a potential along the diagonals allows us to reuse this problem for many types of physics cases.

Adding complexity, hydrogen-like atoms or other one-particle potentials

We are first interested in the solution of the radial part of Schroedinger's equation for one electron. This equation reads

$$-\frac{\hbar^2}{2m}\left(\frac{1}{r^2}\frac{d}{dr}r^2\frac{d}{dr}-\frac{l(l+1)}{r^2}\right)R(r)+V(r)R(r)=ER(r).$$

Suppose in our case $V(r)$ is the harmonic oscillator potential $(1/2)kr^2$ with $k=m\omega^2$ and E is the energy of the harmonic oscillator in three dimensions. The oscillator frequency is ω and the energies are

$$E_{nl}=\hbar\omega\left(2n+l+\frac{3}{2}\right),$$

with $n=0,1,2,\dots$ and $l=0,1,2,\dots$.

Radial Schroedinger equation

Since we have made a transformation to spherical coordinates it means that $r\in[0,\infty)$. The quantum number l is the orbital momentum of the electron. Then we substitute $R(r)=(1/r)u(r)$ and obtain

$$-\frac{\hbar^2}{2m}\frac{d^2}{dr^2}u(r)+\left(V(r)+\frac{l(l+1)}{r^2}\frac{\hbar^2}{2m}\right)u(r)=Eu(r).$$

The boundary conditions are $u(0)=0$ and $u(\infty)=0$.

Scaling the equations

We introduce a dimensionless variable $\rho=(1/\alpha)r$ where α is a constant with dimension length and get

$$-\frac{\hbar^2}{2m\alpha^2}\frac{d^2}{d\rho^2}v(\rho)+\left(V(\rho)+\frac{l(l+1)}{\rho^2}\frac{\hbar^2}{2m\alpha^2}\right)v(\rho)=Ev(\rho).$$

Let us choose $l=0$. Inserting $V(\rho)=(1/2)k\alpha^2\rho^2$ we end up with

$$-\frac{\hbar^2}{2m\alpha^2}\frac{d^2}{d\rho^2}v(\rho)+\frac{k}{2}\alpha^2\rho^2v(\rho)=Ev(\rho).$$

We multiply thereafter with $2m\alpha^2/\hbar^2$ on both sides and obtain

$$-\frac{d^2}{d\rho^2}v(\rho)+\frac{mk}{\hbar^2}\alpha^4\rho^2v(\rho)=\frac{2m\alpha^2}{\hbar^2}Ev(\rho).$$

A natural length scale comes out automagically when scaling We have thus

$$-\frac{d^2}{d\rho^2}v(\rho)+\frac{mk}{\hbar^2}\alpha^4\rho^2v(\rho)=\frac{2m\alpha^2}{\hbar^2}Ev(\rho).$$

The constant α can now be fixed so that

$$\frac{mk}{\hbar^2}\alpha^4=1,$$

and it defines a natural length scale (like the Bohr radius does)

$$\alpha=\left(\frac{\hbar^2}{mk}\right)^{1/4}.$$

Defining

$$\lambda = \frac{2m\alpha^2}{\hbar^2}E,$$

we can rewrite Schroedinger's equation as

$$-\frac{d^2}{d\rho^2}v(\rho) + \rho^2 v(\rho) = \lambda v(\rho).$$

This is similar to the equation for a buckling beam except for the potential term. In three dimensions the eigenvalues for $l = 0$ are $\lambda_0 = 1.5, \lambda_1 = 3.5, \lambda_2 = 5.5, \ldots$.

Define first the diagonal matrix element

$$d_i = \frac{2}{h^2} + V_i,$$

and the non-diagonal matrix element

$$e_i = -\frac{1}{h^2}.$$

In this case the non-diagonal matrix elements are given by a mere constant. *All non-diagonal matrix elements are equal.*

With these definitions the Schroedinger equation takes the following form

$$d_i u_i + e_{i-1}v_{i-1} + e_{i+1}v_{i+1} = \lambda v_i,$$

where v_i is unknown. We can write the latter equation as a matrix eigenvalue problem

$$\begin{bmatrix} d_1 & e_1 & 0 & 0 & \ldots & 0 & 0 \\ e_1 & d_2 & e_2 & 0 & \ldots & 0 & 0 \\ 0 & e_2 & d_3 & e_3 & 0 & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & \ldots & \ldots & \ldots & \ldots & d_{n_{\text{step}}-2} & e_{n_{\text{step}}-1} \\ 0 & \ldots & \ldots & \ldots & \ldots & e_{n_{\text{step}}-1} & d_{n_{\text{step}}-1} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \ldots \\ \ldots \\ \ldots \\ v_{n_{\text{step}}-1} \end{bmatrix} = \lambda \begin{bmatrix} cv_1 \\ v_2 \\ \ldots \\ \ldots \\ \ldots \\ v_{n_{\text{step}}-1} \end{bmatrix} \tag{1.1}$$

or if we wish to be more detailed, we can write the tridiagonal matrix as

$$\begin{pmatrix} \frac{2}{h^2}+V_1 & -\frac{1}{h^2} & 0 & 0 & \ldots & 0 & 0 \\ -\frac{1}{h^2} & \frac{2}{h^2}+V_2 & -\frac{1}{h^2} & 0 & \ldots & 0 & 0 \\ 0 & -\frac{1}{h^2} & \frac{2}{h^2}+V_3 & -\frac{1}{h^2} & 0 & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & \ldots & \ldots & \ldots & \ldots & \frac{2}{h^2}+V_{n_{\text{step}}-2} & -\frac{1}{h^2} \\ 0 & \ldots & \ldots & \ldots & \ldots & -\frac{1}{h^2} & \frac{2}{h^2}+V_{n_{\text{step}}-1} \end{pmatrix} \tag{1.2}$$

Recall that the solutions are known via the boundary conditions at $i = n_{\text{step}}$ and at the other end point, that is for ρ_0. The solution is zero in both cases.

The code sets up the Hamiltonian matrix by defining the minimun and maximum values of r with a maximum value of integration points. It plots the eigenfunctions of the three lowest eigenstates.

```
#Program which solves the one-particle Schrodinger equation
#for a potential specified in function
#potential().

from matplotlib import pyplot as plt
import numpy as np
#Function for initialization of parameters
def initialize():
   RMin = 0.0
   RMax = 10.0
   lOrbital = 0
   Dim = 400
   return RMin, RMax, lOrbital, Dim
# Different types of potentials
def potential(r):
   return 0.5*r*r
   # return 0.0
   # return -1.0/r
   #if r >= 0.0 and r <= 10.0:
   #   V = -0.05
   #else:
   #   V =0.0
   #return V

#Get the boundary, orbital momentum and number of integration points
RMin, RMax, lOrbital, Dim = initialize()

#Initialize constants
Step  = RMax/(Dim+1)
DiagConst = 1.0/ (Step*Step)
NondiagConst = -0.5 / (Step*Step)
OrbitalFactor = 0.5*lOrbital * (lOrbital + 1.0)

#Calculate array of potential values
v = np.zeros(Dim)
r = np.linspace(RMin,RMax,Dim)
for i in xrange(Dim):
   r[i] = RMin + (i+1) * Step;
   v[i] = potential(r[i]) + OrbitalFactor/(r[i]*r[i]);

#Setting up a tridiagonal matrix and finding eigenvectors and eigenvalues
Matrix = np.zeros((Dim,Dim))
Matrix[0,0] = DiagConst + v[0];
Matrix[0,1] = NondiagConst;
for i in xrange(1,Dim-1):
   Matrix[i,i-1] = NondiagConst;
   Matrix[i,i] = DiagConst + v[i];
   Matrix[i,i+1] = NondiagConst;
Matrix[Dim-1,Dim-2] = NondiagConst;
Matrix[Dim-1,Dim-1] = DiagConst + v[Dim-1];
# diagonalize and obtain eigenvalues, not necessarily sorted
EigValues, EigVectors = np.linalg.eig(Matrix)
# sort eigenvectors and eigenvalues
permute = EigValues.argsort()
EigValues = EigValues[permute]
EigVectors = EigVectors[:,permute]
# now plot the results for the three lowest lying eigenstates
for i in xrange(3):
   print EigValues[i]
FirstEigvector = EigVectors[:,0]
SecondEigvector = EigVectors[:,1]
ThirdEigvector = EigVectors[:,2]
plt.plot(r, FirstEigvector**2 ,'b-',r, SecondEigvector**2 ,'g-',r, ThirdEigvector**2 ,'r-')
plt.axis([0,4.6,0.0, 0.025])
plt.xlabel(r'$r$')
plt.ylabel(r'Radial probability $r^2|R(r)|^2$')
plt.title(r'Radial probability distributions for three lowest-lying states')
plt.savefig('eigenvector.pdf')
plt.show()
```

The last example shows the potential of combining numerical algorithms with analytical results (or eventually symbolic calculations), allowing thereby students and teachers to

- make abstraction and explore other physics cases easily where no analytical solutions are known
- 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 eigenvalue solver by changing the number of integration points. The students get trained from day one to think error analysis.
- The algorithm can be tailored to any kind of one-particle problem used in quantum mechanics or eigenvalue problems
- A simple rewrite allows for reuse in linear algebra problems for solution of say Poisson's equation in electromagnetism, or the diffusion equation in one dimension.
- With an ipython notebook the students can keep exploring similar examples and turn them in as their own notebooks.

1.5 Conclusions and Perspectives

- 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, ask for details
- Resources and expenses.
- Tailor to specific disciplines.
- Organizational matters.
- With a local physics education group one can do much more!! At MSU we have a very strong Physics Education Research group headed by Danny Caballero and Washti Sawtelle
- 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. Broad focus on university pedagogical topics.
- 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?

Acknowledgements