Computational Physics at the Physics and Astronomy Department, Michigan State University

Danny Caballero, Sean Couch, Wade Fisher, Connor Glosser, Morten Hjorth-Jensen, Claire Kopenhafer, Brian O'Shea, and Carlo Piermarocchi

February 24, 2017

Introduction: Scientific and educational motivation

Numerical simulations of various systems in science are central to our basic understanding of nature and technology. The increase in computational power, improved algorithms for solving problems in science as well as access to high-performance facilities, allow researchers nowadays to study complicated systems across many length and energy scales. Applications span from studying quantum physical systems in nanotechnology and the characteristics of new materials or subamotic physics at its smallest length scale, to simulating galaxies and the evolution of the universe. In between, simulations are key to understanding cancer treatment and how the brain works, predicting climate changes and this week's weather, simulating natural disasters, semi-conductor devices, quantum computers, as well as assessing risk in the insurance and financial industry.

Computing competence. Computing means solving scientific problems using computers. It covers numerical 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 has always been a central part of education in the sciences and engineering disciplines.

This competence involves:

- derivation, verification, and implementation of algorithms
- understanding what can go wrong with algorithms
- overview of important, known algorithms
- understanding how algorithms are used to solve mathematical problems

- reproducible science and ethics
- algorithmic thinking for gaining deeper insights about scientific problems

All these elements (and many more) are central for maturing and gaining a better understanding of the scientific process *per se*.

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 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.

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

- 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.

So, why should basic university education undergo a shift from classical mathematics to modern computing?

The impact of the computer on mathematics and science is tremendous: science and industry now rely on solving mathematical problems through computing.

- Computing increases the relevance in education by solving more realistic problems earlier.
- Computing through programming is excellent training of creativity.
- Computing enhances the understanding of abstractions and generalization.

- 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 and the scientific method (we hope, and here our physics education group can play a central role in promiting this). Not only is computing via programming a very powerful tool, it also a great pedagogical aid.

For the mathematical training, there is one major new component among the arguments above: understanding abstractions and generalization. While many of the classical methods developed for continuous models are specialized for a particular problem or a narrow class of problems, computing-based algorithms are often developed for problems in a generic form and hence applicable to a large problem class.

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. 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).

General learning outcomes for computing competence

Learning outcomes for numerical algorithms:

1. Deep knowledge of the most fundamental algorithms for linear algebra, ordinary and partial differential equations, optimization, and statistical uncertainty quantification.
2. Overview of advanced algorithms and how they can be accessed in available software.
3. Knowledge of high-performance computing elements: memory usage, vectorized and parallel algorithms.
4. Understanding of approximation errors.
5. Application of fundamental and advanced algorithms to classical model problems as well as real-world problems with assessment of the uncertainty in the answer.

Learning outcomes for symbolic computing:

1. Knowledge of at least one computer algebra system and how it is applied to perform classical mathematics (calculus, linear algebra, differential equations - with verification).

Learning outcomes for programming:

1. Experience with programming in a high-level language (MATLAB, Python, R). Experience with programming in a compiled language (Fortran, C, C++).
2. Experience with implementing and applying numerical algorithms in reusable software that acknowledges the generic nature of the mathematical algorithms.
3. Knowledge of basic software engineering elements: functions, classes, modules/libraries, testing procedures and frameworks, scripting for automated and reproducible experiments, documentation tools, and version control systems.
4. Experience with debugging software, e.g., as part of implementing comprehensive tests.

Learning outcomes for verification:

1. Experience with programming of testing procedures.
2. Deep knowledge of testing/verification methods:
   (a) Exact solution of numerical models
   (b) Method of manufactured solutions (choose solution and fit a problem)
   (c) Classical analytical solutions (incl. asymptotic solutions)
   (d) Computing of asymptotic approximation errors (convergence rates)
   (e) Step-wise construction of tests to aid debugging.

Learning outcomes for mathematical modeling:

1. Experience with deriving computational models from basic principles in physics.
2. Experience with bringing models on dimensionless form to reduce and simplify input data and increase the understanding of the model by interpreting its dimensionless parameters.
3. Experience with solving real problems from applied sciences.

Learning outcomes for presentation of results:

1. Experience with different visualization techniques for different types of computed data.
2. Extensive experience with presenting computed results in scientific reports and oral presentations.

What is deep knowledge? By deep knowledge we here mean the understanding of the underlying fundamental ideas and concepts from which a plethora of seemingly different methods and technologies can be derived. In other words, the deep knowledge brings structure to all the technical details. Obtaining this type of knowledge requires time in class and a lot of exercises. In addition, the students need to reflect about theory and practice. The reflection process is often difficult to implement. Below are some suggestions. A useful concept is simplify, understand, and then generalize. Giving a superficial overview of a bunch of unrelated methods and their applications to unrelated scientific problems equips the students with a wide toolbox, but fails to enhance a fundamental understanding of how multidisciplinary topics play together. Instead, we believe in the following list.

1. Pick a few selected classes of problems,
2. start out with simplified models,
3. apply general, fundamental ideas to construct algorithms,
4. understand all details to correctly implement the algorithms,
5. understand how to judge the numerical quality of the algorithms,
6. understand how to verify that the computations are mathematically correct.

The verification process forces the student to reflect on all the points: What type of problem is actually solved? How can I test that the solution is right? After obtaining an understanding of the simplified problem, one can generalize the models to real applications, but illustrate how the insight from the simplified models and methods gives very valuable knowledge when attacking the generalizations. The focus on simplified models help to detach the mathematics from a lot of discipline-dependent application details and cultivate the common mathematical and implementational ideas. This philosophy is closely related to the key principle stated earlier:

1. solving a complicated problem first starts with the purpose of breaking up the problem into subtasks that belong to general classes of well-studied problems in mathematics,
2. each subproblem is understood with great help simplified models in that class,
3. and finally a synthesis of the subproblems can solve the original problem.

Possible learning outcomes

What follows here is a list of suggested learning outcomes. Not all may be relevant, but hopefully they can serve as a way to spark off discussions.

The learning outcomes are subdivided in three general categories, knowledge, skills and general competence.

- **Knowledge**: A student with a degree in Physics
  - has deep knowledge of the scientific method meaning that the candidate
    1. has the ability to understand advanced scientific results in new fields
    2. has fundamental understanding of methods and tools
    3. can develop and apply advanced computational methods to scientific problems
    4. is capable of judging and analyzing all parts of the obtained scientific results
    5. can present results orally and in written form as scientific reports/articles
    6. can propose new hypotheses and suggest solution paths
    7. can generalize mathematical algorithms and apply them to new situations
    8. can link computational models to specific applications and/or experimental data
    9. can develop models and algorithms to describe experimental data
    10. masters methods for reproducibility and how to link this to a sound ethical scienfitic conduct
    11. has a thorough understanding of how computing is used to solve scientific problems
    12. knows fundamental algorithms in computational science
  - has a fundamental understanding and knowledge of scientific work, meaning that
    1. the candidate can develop hypotheses and suggest ways to test these
    2. can use relevant analytical, experimental and numerical tools and results to test the scientific hypotheses
    3. can generalize from numerical and experimental data to mathematical models and underlying principles
    4. can analyze the results and evaluate their relevance with respect to the actual problems and/or hypotheses
    5. can present the results according to good scientific practices

- **Skills (mainly computational ones)**: A candidate
  - has a deep understanding of what computing means, entailing several or all of the topics listed below
    1. knows the most fundamental algorithms involved (which algorithms should we focus on?)

2. has overview of advanced algorithms and how they can be accessed in available software and how they are used to solve scientific problems
3. has knowledge and understands high-performance computing elements: memory usage, vectorization and parallel algorithms
4. can use effeciently high-performance computing resources, from compilers to hardware architectures
5. understands approximation errors and what can go wrong with algorithms
6. has knowledge of at least one computer algebra system and how it is applied to perform classical mathematics
7. has extensive experience with programming in a high-level language (MATLAB, Python, R)
8. has experience with programming in a compiled language (Fortran, C, C++)
9. has experience with implementing and applying numerical algorithms in reusable software that acknowledges the generic nature of the mathematical algorithms
10. has experience with debugging software
11. has experience with test frameworks and procedures
12. has experience with different visualization techniques for different types of data
13. can critically evaluate results and errors
14. can develop algorithms and software for complicated scientific problems independently and in collaboration with other students
15. masters software carpentry: can design a maintainable program in a systematic way, use version control systems, and write scripts to automate manual work
16. understands how to increase the efficiency of numerical algorithms and pertinent software
17. has knowledge of stringent requirements to efficiency and precision of software
18. understands tools to make science reproducible and has a sound ethical approach to scientific problems

- **General competence**:
  - is able to develop professional competence, entailing:
    1. mature professionally and be able to work independently
    2. can communicate in a professional way scientific results, orally and in written form
    3. can plan and complete a research project

4. can develop a scientific intuition and understanding that makes it possible to present and discuss scientific problems, results and uncertainties

– is able to develop virtues, values and attitudes that lead to a better understanding of ethical aspects of the scientific method, as well as promoting central aspects of the scientific method to society. This means for example that the candidate

1. can reflect on and develop strategies for making science reproducible and to promote the need for a proper ethical conduct
2. has a deep understanding of the role basic and applied research and computing play for progress in society
3. is able to promote, use and develop version control tools in order to make science reproducible
4. is able to critically evaluate the consequences of own research and how this impacts society
5. matures an understanding of the links between basic and applied research and how these shape, in a fundamental way, progress in science and technology
6. can develop an understading of the role research and science can play together with industry and society in general
7. can reflect over and develop learning strategies for life-long learning.

Specific algorithms and computational skills

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.

We list also tools that are important in developing numerical projects. These tools allow students to develop a better understanding of the scientific process as well, in addition to bring in a sound ethical approach to science.

The algorithms and tools listed here reflect to various degrees the previously discussed learning outcomes. They can be integrated to various degrees in the physics courses that we propose below here. Furthermore, several of these algorithms can be used and reused in the various courses. The ideal would also be that some of these algorithms have been presented in different mathematics courses.

Central algorithms. The following mathematical formulations of problems from the physical sciences play a prominent role and should be reflected in the way 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.

- Root finding and interpolation techniques, used in methods for theoretical physics, quantum mechanics, electromagnetism and mechanics.

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, we will 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.

As part of this proposal, we have developed several examples of problems and projects that can be included in our undergraduate courses in physics. You can click on the following links (ipython notebooks or PDF formats)

- Mechanics
  1. Ipython notebook
  2. Standard PDF file
- Quantum mechanics
  1. Ipython notebook
  2. Standard PDF file
- Electromagnetism
  1. Ipython notebook
  2. Standard PDF file
- Statistical and thermal physics
  1. Ipython notebook
  2. Standard PDF file
- Methods in Theoretical Physics (not yet ready)
  1. Ipython notebook
  2. Standard PDF file
- Physics laboratory (not yet ready)
  1. Ipython notebook
  2. Standard PDF file

Central tools and programming languages. We will strongly recommend that Python is used as programming language for all the courses proposed below. This means that students can apply their knowledge from CSE 201 and extend and add to their knowledge of programming in various physics classes. We recommend strongly that the following tools are used

1. Jupiter and ipython notebook.
2. Version control software like git and repositories like github

3. Other typsetting tools like Latex.

4. Developing unit tests and using existing tools for unit tests. Python has extensive tools for this

5. Other high-level environments like Mathematica and Matlab can also be presented and offered as special courses like PHY102 - Physics Computations I.

The notebooks can be used to hand in exercises and projects and give the students a training in presenting their studies in the form of scientific/technical reports.

Version control software allows teachers to bring in naturally aspects like reproducibility of science as well as enhancing collaborative efforts among students. Making science reproducible and ethical aspects linked with reproducibility can be easily implemented with proper usage of version control software. Using version control can also be used to train students in presenting benchmark results, allowing thereby for verification by others of 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.

Proposed physics courses that could integrate a computational perspective

For a major in physics degree at Michigan State University, the course CSE 201 Fundamentals of Information Technology is compulsory and it lays the foundation for the insertion of computational exercises and projects in various physics courses.

We would like to propose that the following courses that aim at integrating the above learning outcomes and goals:

- Classical Mechanics 1: PHY321.

    1. Here one could focus on ordinary differential equations like modified Euler, Verlet and perhaps RK methods. There are several interesting problems that can deepen the understanding of newtonian problems, from modeling planetary motion to models for friction and earthquake simulations. With not too advanced programs one can easily bring in actual research problems (friction is a good example). The classical pendulum is also a good case to illustrate. The code can easily be reused for studies of for example an RLC circuit in electromagnetism. The equations for a buckling beam lead to an eigenvalue problem which can reused in the quantum mechanics course.

- Thermal and Statistical Physics, PHY410

1. Random numbers, random walks, probability distributions, Monte Carlo integration and Metropolis algorithm. Applications to statistical physics. Simulation of simple phase transitions with spin models like the Ising model. Studies of Brownian motion and diffusion can easily be done.

- Methods in Theoretical Physics, PHY415

  1. Here one could discuss algorithms which are relevant for the other physics courses discussed here, such as particular eigenvalue solvers. Furthermore, when solving famous differential equations in physics, one can write functions to generate polynomials like Legendre, Laguerre, Hermite functions etc. Here it would be natural to discuss Gaussian quadrature and how to use this to integrate numerically.

- Advanced Laboratory course in Physics PHY451

  1. Data analysis plays a central role, natural to have algorithms on data fitting, computations of mean values, variance and standard deviations, covariance, famous distributions. Many of these topics find also their natural place in a course on statistical physics like PHY410.

- Quantum mechanics 1 PHY 471 (and possible extentions to Quantum mechanics 2 PHY 472)

  1. Ordinary and partial differential equations, eigenvalue solvers and root finding methods. Can explore interacting two-electron problem with simple rewriting of for example harmonic oscillator problem. The Metropolis algorithm can be used to simulate one and two-body problems like the Helium atom using Variational Monte Carlo. Gives students a feeling for variational calculus which can be included and taught in PHY415

- Electromagnetism 1 PHY481 (and possible extensions to Electromagnetism 2 PHY482)

  1. Both ordinary and partial differential equations with two-point boundary value problems. Central algorithms based on finite difference for one and two-dimensional wave equation and Poisson's and Laplace's equations. See the detailed learning outcomes for more information.

Question to us: Should we propose detailed learning outcomes as done by Danny?

Physics Education research and computing in science education

The introduction of computational elements in the various courses should be stronly 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. This means that it is possible to develop research motivated curricular changes. There are many interesting challenges here, like which are the main obstacles when transferring from a classical pen and paper approach to actually have a working program which solves the same (and more general problems). What is a good progression in presenting numerical topics in physics courses? Are there specific mathematical skills we would like our students to have? How do we integrate student-active teaching, how do we develop and test various assessment methods?

Links with CSME and Mathematics courses

We need to figure out, beyond CSE 201, if computational elements, or some of the abovementioned algorithms are included in the mathematics courses our physics students take.

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, like the present PHY480 Computational Physics offers as well more advanced projects which train the students in actual research, developing more complicated programs and working on larger projects. Such a 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
- 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

Then we should consider more advanced (at the graduate level) courses which could cover specialized topics. These could be (presently some of this is covered by PHY905, sections 002 and 003)

1. Computational quantum mechanics, spanning from Monte Carlo methods to other methods used in atomic, molecular, nuclear, condensed matter and high-energy physics
2. Computational Statistical mechanics, covering topics spanning from Molecular dynamics to studies of phase transitions, highly relevant for several subdisciplines
3. Computational Astrophysics and Astronomy
4. Data analysis and machine learning tailored to physics problems (PA+CSME?)
5. High-performance computing (CSME)
6. More...

A possible Physics major in computational physics and a possible MSc in Computational Physics

To be discussed.