Computing in Physics courses 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

Mar 30, 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 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 subatomic 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.

On the part of students, this competence 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.

All these elements are central for maturing and gaining a better understanding of the modern 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 towards 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 classes of problems 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 towards 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 can increase the relevance in education by solving more realistic problems earlier.
- Computing through programming can be excellent training of creativity.

- Computing can enhance the understanding of abstractions and generalization.
- Computing can decrease 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 research group can play a central role in promoting this). Not only is computing via programming a very powerful tool, it can also be 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 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.

Moreover, today's projects 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

Below, 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 bachelor's program at Michigan State. 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. We reserve the discussion of specific learning goals to individual course experiences (e.g., Electrostatics).

Learning outcomes for numerical algorithms. 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, optimization, and statistical uncertainty quantification
- A working knowledge of advanced algorithms and how they can be accessed in available software
- A working knowledge of high-performance computing elements including memory usage, vectorized and parallel algorithms
- A deep 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

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, 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:

- 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., GitHub).
- 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 (i.e., modeling) 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 forms to reduce and simplify input data, and
- Interpreting the model's dimensionless parameters to increase their understanding of the model and its predictions

Learning outcomes for verification. Verifying the 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 for students to develop:

- An understanding of how to program testing procedures
- A deep knowledge of testing/verification methods including the use of:
- Exact solutions of numerical models
- Method of manufactured solutions (i.e., choose solution and fit a problem)
- Classical analytical solutions including asymptotic solutions
- Computed asymptotic approximation errors (i.e., convergence rates)
- 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)

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 reserve the coupling of the the broad learning goals above to the algorithms articulated below to our discussion of specific course (or topical) 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. In addition, use of the algorithms can facilitate instruction in an ethical approach to science.

The algorithms and tools listed here can be intergated in different ways depending on the specific learning goals for the course or topic. Furthermore, several of these algorithms can be used and, then, revisited in the various courses.

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.

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. 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.

As part of this proposal, we are in the process of developing 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

We hope these examples can aid physics teachers in their usage of computing in physics courses. Furthermore, via a consensus and collectively driven approach, the hope is that this body of examples can be continuously enlarged.

Central tools and programming languages. We will strongly recommend that Python is used as the high-level programming language for all the courses proposed below. Other high-level environments like Mathematica and Matlab can also be presented and offered as special courses like PHY102 - Physics Computations I. This means that students can apply their knowledge from CMSE 201, which makes 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 like GitHub
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 for specific physics courses

For a major in physics degree at Michigan State University, the course CMSE 201 Introduction to Computational Modeling is compulsory and it 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. These suggestions are to be viewed as inputs to the ongoing discussions. The list of possible outcomes can be reduced or enlarged.

Danny: I have only constructed learning goals for PHY 481. I intend these to provide a example of what our goals might look like. They are debatable and not set in stone, but what they do is provide us with a starting point for discussion and an example of what measurable (i.e., assessable) learning goals might look like.

We propose that the following courses aim to integrate the above learning outcomes and goals:

Classical Mechanics 1 PHY321. After completing Classical Mechanics 1 PHY321, 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 PHY410. After completing Thermal and Statistical Physics PHY410, 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 micro-canonical 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.

Methods in Theoretical Physics PHY415. After completing Methods in Theoretical Physics PHY415, 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 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.

Advanced Laboratory course in Physics PHY451. After completing Advanced Laboratory course in Physics PHY451, 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 1 PHY 471. After completing Quantum mechanics 1 PHY 471, 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 issues would also be discussed in PHY415. The students will then see central algorithms being used in different physics settings.

Many of these topics can be included and extended upon in PHY472.

Electromagnetism 1 PHY481. After completing Electromagnetism 1 PHY481, 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 the finite*differencing 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
- See the detailed learning outcomes for PHY 481 for more information.

Many of these topics can be included and extended upon in PHY482.

Danny: I haven't yet reviewed the material below in detail because I wanted us to spend time discussing the learning outcomes. These learning outcomes are connected to the work that the Physics Education Research Lab would do as we help to develop assessments and evaluations to share with the faculty

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?

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. 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
- 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

During the last years, we have witnessed an enhanced background in computing and programming skills among our students. Our students will however always form a heteregeneous group, with often differing competences, experiences and

interests. In order to be able to offer a wider spectrum of computational physics courses, the present working group would like to propose courses that can be used in both a major in computational physics, or a Master of Science in computational physics, or a dual degree from the Physics and Astronomy department and the new Computational Mathematics, Science and Engineering (CMSE) department, or just satisfy the scientific interests of our students

In addition to PHY480, we may consider more advanced (at the graduate level) courses which cover specialized topics. These could be (presently some of these topics are 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. This course can be jointly taught with the new department of Computational Science.

5. Additional CMSE courses, see list below.

The CMSE department offers in addition a series of courses at all levels.

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

The above advanced courses can form the basis for a MSc in computational physics, as well as, together with CSME courses, a major in computational physics.

The following is a suggestion for a major in Computational physics and a Master of Science degree in Computational Physics.

Master of Science in Computational Physics. The Master of Science in Computational Physics has a duration of two years, with three semesters of course work and one semester of thesis work. (discuss whether we should have the flexibility of two semesters of course work and two semesters of thesis work).

1. The thesis work is to be conducted with one or more research groups at the Department of Physics and Astronomy. It can also be of a more interdisciplinary character and can be a joint thesis with another department at the College of Natural Science and College of Engineering.

2. Core physics courses (these courses should be compulsory)

- PHY 810, Methods of Theoretical Physics, 3 credits, fall semester
- PHY 820, Classical Mechanics, 3 credits, fall semester
- PHY 851, Quantum Mechanics I, 3 credits, fall semester
- PHY 841, Classical Electrodynamics I, 3 credits, spring semester
- PHY 831, Statistical Mechanics, 3 credits, fall semester

3. Computational Science and Physics courses, select in order to fill up necessary credit requirements.

    - Computational quantum mechanics, spanning from Monte Carlo methods to other methods used in atomic, molecular, nuclear, condensed matter and high*energy physics
    - Computational Statistical mechanics, covering topics spanning from Molecular dynamics to studies of phase transitions, highly relevant for several subdisciplines
    - Computational Astrophysics and Astronomy
    - Data analysis and machine learning tailored to physics problems. This course can be jointly taught with the new department of Computational Science.
    - CMSE 802, Methods in Computational Modeling, 3 credits, spring and fall semesters
    - CMSE 820, Mathematical Foundations of Data Science, 3 credits, spring semester
    - CMSE 821, Numerical Methods for Differential Equations, 3 credits, spring semester
    - CMSE/CSE 822, Parallel Computing, 3 credits, fall semester
    - CMSE 823, Numerical Linear Algebra, I, 3 credits, fall semester

Major of Science in Computational Physics. The major of science in Computational Physics replaces all physics laboratory courses with either Computational physics courses and/or CMSE courses. Need more input here.

Links with CMSE and Mathematics courses

We would like to propose to establish a working group which studies an eventual integration and possibly coordination of many of the abovementioned algorithms in the mathematics courses our physics students take. For example, algorithms from linear algebra play a central role, and with CS201 as background, there is a unique opportunity to enlarge the body of mathematical examples with numerical tools and algorithms. The most likely scenario in the next five to ten years is that our students will be better educated in programming and

computations. This should ease the introduction of numerical algorithms and exercises in many of the basic mathematics courses.

With such a background, it is easier for physics teachers, with proper repetitions and adequate material, to introduce numerical exercises and projects in many basic physics courses.