Examples on how to integrate a computational perspective in physics courses

Morten Hjorth-Jensen[1]

Michigan State University, USA, and University of Oslo, Norway[1]

Jun 21, 2018

Why Computing in Physics Courses

- One of two jobs in the stem fields will in 2020 be within computing.
- Demands on computing in research continue to grow.
- How can we integrate computing in core Physics courses in a coordinated and coherent way?
- Computing competence represents a central element in scientific problem solving, from basic education and research to essentially almost all advanced problems in modern societies.

A large fraction of our students get jobs were computing is central

Computational modeling and data analysis is core to a variety of non-academic career paths that are often pursued by physics students following the receipt of their Bachelor's degree.

A 2012 AIP (American Institute of Physics) survey of 5,000 recent physics Bachelor's degree recipients indicated that 50% chose to enter the workforce immediately after receiving their degree. Of these, roughly 75% go into STEM-related fields including engineering, software development, or information technology.

Depending on sub-field, 60-80% of physics degree-holders in industry regularly need to engage in computer programming, simulation and modeling tasks. Many respondents to the survey stressed the importance of computer skills, including programming, in increasing their marketability to potential employers.

Computing as a way to develop intuition about the scientific process

The use of computational modeling in the classroom setting provides students with insights that are complementary to those resulting from pencil-and-paper manipulation of equations.

Computing in core physics courses allows us to bring important elements of scientific methods at a much earlier stage in our students' education.

The ability to closely examine the behavior of systems that are too complex to be easily analytically tractable, or that have no analytic solutions (i.e., many systems of practical interest), helps to develop intuition that is unavailable to many students from analytic calculation.

Computing competence

Computing means solving scientific problems using all possible tools, including symbolic computing, computers and numerical algorithms, and analytical paper and pencil solutions . **Computing is about developing an understanding of the scientific method** by enhancing algorithmic thinking when solving problems.
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.

Better understanding of the scientific method

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.

Started initiative at Michigan State University

There are several courses in Physics where numerical exercises and/or projects have been included. Examples are: PHY321 (Mechanics), PHY410 (Thermal Physics), PHY415 (Mathematical Methods in Physics), PHY481 (Electromagnetism I) and PHY482 (Electromagnetism II).

The programming course CMSE 201 is required in order to take PHY321. Our students have thus a good basis in scientific programming

We have proposed a focus on integrating computing in upper division physics courses including:

- outline proposed learning outcomes relating to computational modeling and data analysis within the upper-division physics curriculum,
- offer a concrete and realistic plan of action that will allow us to attain these learning outcomes for selected courses, and
- provide a plan to assess the progress of our efforts.

Started initiative at Michigan State University

The proposed computational curriculum goals are to:

- Integrate computational learning practices in PHY321 by Fall 2018 or Spring 2019
- Gradually integrate the remaining core (also elective) courses (PHY410, PHY415, PHY451, PHY471, PHY481), keeping pace with the cohort of students on this initial schedule.
- Develop resources to support faculty and ensure continuity over year-to-year transitions.
- Maintain program vitality by systematically refreshing computational course material as our programs evolve.
- The working group will continue to work on the above action items and will invite current (and future) instructors to serve as members of the working group for the specific courses.

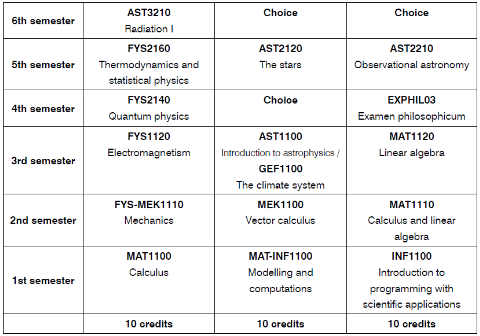
Implementation in Oslo: The C(omputing in)S(cience)E(Education) project (since 2003)

What we do

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

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

Example of bachelor program, Physics and Astronomy (astro path), University of Oslo



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

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

Example: Computations from day one

Differentiation

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

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

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

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

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

- Implementation in Python in INF1100

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

Example: Computations from day one

Differentiation and comparison with symbolic expressions

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

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

Other Examples

Integration by Trapezoidal Rule

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

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

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

Typical implementation first semester of study

Integration by Trapezoidal Rule

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

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

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

Symbolic calculations and numerical calculations in one code

Python offers an extremely versatile programming environment, allowing for the inclusion of analytical studies in a numerical program. Here we show an example code with the **trapezoidal rule** using **SymPy** to evaluate an integral and compute the absolute error with respect to the numerically evaluated one of the integral $\int_0^1 dx x^2 = 1/3$:

```
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)
```

Error analysis

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

```
from math import log10
import numpy as np
from sympy import Symbol, integrate
import matplotlib.pyplot as plt
# function for the trapezoidal rule
def Trapez(a,b,f,n):
   h = (b-a)/float(n)
   s = 0
   x = a
   for i in range(1,n,1):
       x = x+h
       s = s+ f(x)
   s = 0.5*(f(a)+f(b)) +s
   return h*s
#  function to compute pi
def function(x):
    return 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
```

Integrating numerical mathematics with calculus

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

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

Additional benefits: A structured approach to solving problems

In this process we easily bake in

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

Additional benefits: A structure approach to solving problems

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

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

Learning outcomes three first semesters

Knowledge of basic algorithms

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

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

Later courses

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

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

Coordination

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

Challenges...

.. and objections

Standard objection: computations take away the attention from other central topics in 'my course'.
CSE incorporates computations from day one, and courses higher up do not need to spend time on computational topics (technicalities), but can focus on the interesting science applications. Coordination and synchronization across departments and courses. Increases collaboration on teaching and awareness of pedagical research.

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

Examples of simple algorithms, initial value problems and proper scaling of equations

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.

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

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

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

Mechanics and electromagnetism, initial value problems and now proper scaling

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**

Other examples of simple algorithms that can be reused in many courses, two-point boundary value problems and scaling

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

1. A buckling beam and tridiagonal 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 and get the same type of problem

Discretize the second derivative and the rhs

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

with $i = 1, 2, \ldots, n$. 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 $\mathbf{v} = (v_1, v_2, \ldots, v_n)^T$ allows us to rewrite the differential equation including the boundary conditions as a standard eigenvalue problem

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

Assume we want to solve the radial part of Schroedinger's equation for one particle in three dimensions. 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.
Scale now the equations with $\rho=r/\alpha$ where α is a constant of dimension length.

A natural length scale comes out automagically when scaling

Manipulating the equations by requiring

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

which define 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

The Python code

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.

```python
#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
# Harmonic oscillator here, easy to change
def potential(r):
    return 0.5*r*r

#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)
```

The power of numerical methods

The last example shows the potential of combining numerical algorithms with analytical results (or eventually symbolic calculations). A simple change of potential gives a new physics case, example of a box potential

```python
# Different types of potentials
def potential(r):
    if r >= 0.0 and r <= 10.0:
        V = -0.05
    else:
        V =0.0
    return V
```

This allows 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

Which aspects are important for a successful introduction of CSE?

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

Summary

- Make our research visible in early undergraduate courses, enhance research based teaching
- Possibility to focus more on understanding and increased insight.
- Impetus for broad cooperation in teaching. 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?

We invite you to visit (online and/or in real life) our new center on Computing in Science Education

More links

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