Integrating a computational perspective in physics courses

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

Apr 5, 2017

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

And find problems with the article?

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.

Computing competence

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.

Integration by Trapezoidal Rule

- The algorithm for computing the integral vha the Trapezoidal rule for an interval $x \in [a, b]$

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

Typical implementation

Integration by Trapezoidal Rule.

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

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

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

Symbolic calculations and numerical calculations in one code

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

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

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

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

Error analysis

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

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

Integrating numerical mathematics with calculus

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

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

Examples of simple algorithms, two-point boundary value problems and scaling

1. The buckling beam and Toeplitz matrices (mechanics and math 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, lin algebra (elmag)
5. The diffusion equation in one dim, the same matrix again, lin algebra

Examples of simple algorithms, initial value problems and scaling

1. ODE RLC circuit
2. ODE Classical pendulum
3. ODE Solar system

Mechanics, Realistic Pendulum

Classical pendulum with damping and external force

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

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

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

Electromagnetism, RLC circuit

Same equation as the pendulum for an RLC circuit

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

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

More Examples from Physics Courses

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