PH-204 Physics Simulation

James Bateman

December 9, 2019

Contents

Lectures

About

These are notes-to-self which I use during lectures.
They may be useful, but they are necessarily less structured that other material on the course website.
Please refer to the other material as well. This PDF will not make sense on its own.

Lecture 0 2019-09-30 Mon

- Welcome; Introduction
- Course about using comp to solve phys problems
  - phrasing mathematics you already know
- If preconception "not good with computers":
  - please reconsider; this is not a useful way to think about yourself
  - this is about logic & instructions; a branch of mathematics
  - programming makes this real; making a machine to think
- Rationalle
  - simulation
    - often easy to write down physics equations, but difficult to solve
    - e.g. gravity: 2 body works (conic sections); 3 body not analytic in general
    - therefore solve on a computer
  - also, data analysis
    - want to know the temperature of 1B stars measured by Gaia telescope?
      - not going to analyse by hand!
    - or 1000 hours of data looking for a signal meeting specific criteria
- Approach
  - learn how to write instructions for a task
  - Assignment 1a is about writing instructions for playing a game – no computer code
  - (Assignment 1b is about writing the associated code)
  - this is the core idea: writing instructions
  - then we can learn specifics
- Structure
  - Lecture (Mon 2pm); PC workshop (Tue 2pm - 4pm)
  - Something of a hybrid between lecture-based and lab-based module
  - 100% continually assessed
  - Problem sheets 1 (a & b), 2, 3
  - preceded by Competence tests 0,1,2,3,4
    - Comp tests should take no more than 30 mins
    - if you struggle, there's help; able to do questions by end of session
- Course information
  - Blackboard, with link to jamesbateman.eu
    - (lack of imagination in the domain name)
  - Course notes as PDFs
  - Submit work on Blackboard
- Python
  - real language – science & industry; e.g. LIGO & Youtube
  - not just a teaching aid
- Libraries
  - need the fundamentals: loops, variables, conditionals
  - then use libraries

  - for example:
    * newton's method for root finding $f(x) = 0$
    * trapezium rule for numerical integration
    * Runge–Kutta for solving ordinary differential equations
  - these are **approximate** numerical methods
    * spend time understanding
      · how to tackle a problem
      · what method to use and why
      · limitations of that method
    * rather than writing code to implement that particular method

- Beyond
  - Use the biggest block you can find to get the work done
  - Don't reinvent the wheel
  - e.g.
    * want to do image recognition? Use OpenCV
    * machine learning? use TensorFlow.
    * controlling instruments? `usbtmc` etc
  - Scientific libraries are just one example; there's so much available
    * beware libraries that make something complex when it should be easy
    * this can arise from trying to treat a very general case

- A word on **Academic Integrity**
  - Work you submit must be your own
    * Collaboration/Collusion
    * Fine (encouraged!) to work on a problem together
    * However, if you cannot do the work **once you've left the group** then you're kidding yourself
  - Exams
    * awful for many reasons
    * do pretty much solve academic malpractice
    * this is why majority of modules use exams
  - Authentic assessment
    * Exams not appropriate; this isn't how anyone writes code
    * Copying from notes is **encouraged**; the course notes are intended to be useful well beyond this course
    * Copying from colleagues is academic malpractice
      · I will notice
      · There's no team of markers; there's no automation; there's just me
      · I don't notice variable names, or other superficial changes
      · I see the story in the code, and I will notice if you've copied
      · Please please don't
    * Assessment is
      · Part problem-sheet like, with small straightforward questions
      · For these, there might just be one correct answer
      · Part open-ended question, more in the style of a mini-project of a lab-report
      · For these, discuss the problem
      · Collaborate
      · Write your own answers
    * Leave ENOUGH TIME
      · This is the first thing you've submitted that counts towards your degree

- PH113
  - 2 hour session in Skills Session
  - Please review if hazy
- 10 credits means 100 hours
  - 11 weeks x (1 hour lecture + 2 hour PC)/week = 33 hours scheduled
  - Therefore, spend twice as long **independent** study as contact time
  - Note that this cannot be compressed into revision as in exam-based module
  - WORK DURING THE TERM on this course

Show PH113 notes
Open Spyder
Review the basics of Python

```
# Calculator
2+2

# Variables
a = 2 + 2 # calculate RHS and store in memory location called "a"

b = 2 * a # refer to value stored in location called "a"

# Lists
c = [1,2,3,"cat",9,9] # lots of things stored in one object

c[0] # refer to elements by index (starting at zero)

c[0] = "mouse" # can change the contents of a list -- "mutable"

c.append(100) # add something to the end of the list

c.[TAB] # tab completion; lists all the possible completions

c.index? # details on what c.index is

c.index("cat") # look up where "cat" appears in list "c"

# In-built functions
abs?

abs(-3)

abs(3+4j) # complex numbers
```

Defining our own functions is where it gets really useful
Function has well-defined inputs and well-defined outputs.
Any Python code (including other functions) can be in a function.

```
def square(x):
    y = x*x
    return y
```

Indent says that this belongs to function definition
`square` is the function name
`x` is the input
`return y` means "the output from this function is `y`".
This `name` is just like any other variable name.

```
def f(x):
    return x**2

g = f

g(3)
```

```
9
```

Make inputs explicit; for others, Python will look up the value when the function is called (not when it's defined)

```
a = 2

def f(x):
    return a*x

a = 3

f(4)
```

```
12
```

Loops

```
for n in [iterable]: # any iterable
    # do something
```

iterables are something for which it makes sense to ask for each item in turn e.g. list, set, range
range makes integers as they're needed

```
a = [] # a is now an empty list
for n in range(10):
    a.append(n*n)

a
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Also, **list comprehension**

```
a = [n*n for n in range(10)]
```

Conditionals

Often need to make a decision

```
if [condition]:
    # do something
else:
    # do something else
```

- [condition] is something that is either True or False
  - a < b; a > b;
  - a <= b; a >= b
  - a != b; a == b

Note the double equals

this is to make it distinct from a=b which means **assign** the value currently stored in b to the name a.

The double equals means "are these things equal" whereas the single equals means "set LHS equal to the RHS".

Lecture 1 2019-10-07 Mon

- Well done: many for whom these ideas were new made real progress
- Need to get the basics solid; will be using libraries to do real work
- i.e. get quickly to stage where this is useful

Competence 0

- I sent out comments on everyone's code, and general notes on how to approach
  - if you haven't read it, you really should
- `isprime` did something useful; we used this well-defined unit to do something else
- *show isprime code; show function "primes"*
- **don't reproduce** functionality of `isprimes`; just use it!
- not only more reliable and easier to debug, but also more flexible
  - e.g. could use exact same code with `isprimes` replaced
  - by making a function which takes a function

Material 1

- functions & mutiple inputs, optional inputs, multiple returns
- flow control extras
- imports

Numpy

- `import numpy as np`
- show what numpy can do
- remind of `thing.[TAB]` and `thing?` for documentation
- illustrate with `np.linspace`
- arrays vs lists

Newton's method

- Sketch to illustrate method
- Homemade code
  - start with differential as a required input
  - then introduce numerical differential
- `from scipy.optimize import newton`
- `newton?`

Plotting

- emphasise difference between function and expression

Assignment 1a

- Absolutely essential to get the idea of using logic
- For A1a, asked to play a game
  - no code yet
  - think about how to write the clear instructions
  - therefore not limited by how to write this in Python
  - for A1b, write the code
  - deadline A1a is 1 week before A1b
- *Read from "The game" part of A1a*

Flowchart or pseudocode

- any way to express the ideas unambiguously
- don't fret about any formal definitions of pseudocode
- *sketch flowchart for "cat" strategy*
  - could also be as pseudocode
  - one way might be clearer if using "decisions"
- electronic submission
  - so that you have it when finishing A1b
  - happy with handdrawn; just take a photo & upload
  - ask me if you have trouble digitizing

Lecture 2.1 2019-10-14 Mon

Status update

We have now covered

- Part 0: **The Fundamentals** *revision of PH-113*
- Part 1:: **Fundamentals, functions, and finding numerical solutions**

i.e. you have everything you need for Assignment 1. Doing it is hopefully a useful exercise. You have to combine components in novel ways. I am happy to answer questions, either in Office Hours, in or after Lectures, or by Email.

Next, we meet

- Part 2: **Numerical integrals, interpolation, and fitting to data**

Next two workshops are **unassessed** practice of this material, and (if time) assistance with Assignment 1.

If haven't yet, please attempt Assignment 1b. May be pleasantly surprised. Introduces a different root finder; use `thing?` to see how to use it.

Plotting

Intended to get to plotting, but didn't quite.

First, meet **arrays**. Array contains a regular grid of the same thing.

Simplest is a 1D array of numbers.

```
import numpy as np
a = np.array([1,2,3]) # make from a list
```

This is useful because you can do mathematical operations on the whole thing at once:

```
b = a**2
b
```

```
[1 4 9]
```

Numpy includes convenience functions for making linearly spaced numbers between end-points

```
x = np.linspace(0,10,1001)
```

You can calculate explicit numbers from these:

```
y = x**2 + 2*x + 1
```

We can now plot this using a plotting library. The library is `matplotlib` and we're using a specific part called `pyplot`:

```
import matplotlib.pyplot as plt # import a specific part of the library and call it plt

import numpy as np
x = np.linspace(-1,1,101)
y = x**3 - 2*x**2

plt.plot(x,y)
plt.title("This is the title")
plt.xlabel("Give x-axis a label")
plt.ylim([-1,1])
# plt.show()
```

Interactive/non-interactive plotting Plots in Spyder are, by default, shown in the Terminal. This is often annoying. Switch to it making a new window, in which you can zoom etc, by changing, in Spyder Preferences, IPython Console/Graphics/Backend to Automatic.

`plt.plot` needs explicit values for `x` and `y`.

Save a figure *from the code* using something like `plt.savefig("plot.pdf")`. Follow with `plt.close('all')` or subsequent plots may contain previous plots.

Data structures

`list` and `array` are two examples of data structures.

Others include `tuple`

Also `dict` and `set`.

Numerical integration

Emphasised writing functions. Now trivial to integrate.

Use quad and pass it a function

```
f = lambda x: np.exp(-x**2)/np.sqrt(np.pi)

from scipy.integrate import quad
quad(f, -10,10)
```

```
(0.9999999999999999, 2.085073301853646e-13)
```

Beware that this is a bit black-box; if want more control do trapezium by feeding points

```
x = np.linspace(-10,10,10001)
y = f(x)

from scipy.integrate import trapz
trapz(y,x) # show trapz? to show how I know it's y,x not x,y!
```

```
1.0
```

no indication of uncertainty!! could we use fewer points? how many? how does accuracy depend on function and number of points?

Flow control extras

`break`; `continue`

list comprehension extras: `if` clause for filtering

Lecture 2.2 2019-10-21 Mon

Assignment 1 *illustration*

Inform that this exists
Did anyone get Blackboard announcement? Email copy?
Talk through

- equivalent assignment, with reasonable answers, for a different game
- many have found that writing down every possibility is not practical
- this was a point; find abstract rules.
- Get **something** down for deadline tomorrow; clarify and TEST CODE for Assignment 1b

Today

Some more Python in-builts; advise strongly to glance at docs (linked in Material 2)
Then interpolation and curve-fitting.
Have been asked to make available any code I type, so will do that.

In-builts

- types of brackets
- in, any, all
- is vs ==; python visualizer
- slicing (see Material 2)

Interpolation

Starting to talk about working with data
First illustration: interpolate
Make and plot some points:

```
def f(x):
    from numpy import exp, sqrt, pi
    return exp(-x**2)/sqrt(pi)

x_data = np.linspace(-2,2,11)
y_data = f(x)
plt.plot(x,y,'.')
plt.close('all')# plt.show()
```

Build interpolating function

```
from scipy.interpolate import interp1d
g = interp1d(x,y)
plt.plot(x_data,y_data,'.')
x_axis = np.linspace(-2,2, 1001)
plt.plot(x_axis, g(x_axis))
plt.close('all')# plt.show()
```

Now with cubic spline option (see `interp1d?` for more)

```
from scipy.interpolate import interp1d
g = interp1d(x,y,kind='cubic',bounds_error=False,fill_value=0)
plt.plot(x_data,y_data,'.')
x_axis = np.linspace(-5,5, 1001)
plt.plot(x_axis, g(x_axis))
plt.close('all')# plt.show()
```

2D interpolation

Possible. Generally more difficult.

Need to decide which points are nearest.

In 2D, make a triangle of the 3 nearest points, and linearly interpolate by using the plane which passes through these points.

Generalises to higher dimensions.

Fitting

Interpolating must be used with caution. It makes up points in the gaps without any reference to the physics. Can be badly misleading.

Instead, have a physical model, and use data to inform free parameters of that model.

Way to think about it: **maximize** probability of model parameters have certain value, given the data we have observed. This is what "least-squares" fitting does, provided uncertainties are symmetric. See "Data Analysis: A Bayesian Tutorial" by Sivia & Skilling.

See Material 2.

Lecture 2.3 2019-10-28 Mon

Assignment 1

Well done assignment 1. Not seen all yet but some good code.

Intention was for you to learn how to combine core programming concepts to solve a problem. This was tough.

- First 5 marks (of 15 assigned to this in 1b) were straightforward (provided know function, return, a few if statements).
- Second 5 marks: some easy with a simple strategy; getting more difficult
- Final 5 marks were really quite challenging

A reductionist view focusing on the specifics of the game, and resulting in a huge flowchart, will not work for 4x4. Tried to communicate this including in feedback to 1a. Anyone who engaged with support got advice to do more abstract. Some came to me with abstract ideas but not quite sure how to implement.

Favourite (so far):

- construct list of all winning combinations
- remove those which we cannot get (because opposition has at least one)
- Do we have q-1 elements of any?
  - If so, pick the winner
- Do we have q-2 elements of any?
  - If so, pick the number which appears most frequently in the combinations for which we have q-2 tiles.

Generally, making lists and filtering them is often a good structure.

Assignment 2

Will be less abstract.

Needed to make you go through the work of thinking of logical solutions.

Subsequent assignments have aspects of this but with less emphasis.

Likely to involve:

- Loading real data
- Numerical integration of a curve based on real data
- Fitting theoretical curves to real data and extracting fit parameters
- Plotting 2D data
- Working with vectors and matrices

Material

- curve fitting and unpacking
- can incorprate error bars
  - different points have different uncertainties
  - fit algorithm needs to know this
  - maximizing probability of free parameters given the data
  - least squares **only** when uncertainties are symmetric and points independent
- Randomness
  - briefly only
- Loading and saving
  - get data from a file
  - this is how we process experimental data, or save our analysis
- 2D arrays
  - slicing
  - slicing in 2D
- Plotting in 2D
- glob
  - get a list of files

Lecture 3.1 2019-11-04 Mon

Assignment 1b

Marking now complete.

When code didn't work, tried to interpret what you intended.

Average for the problem sheet was 60 +/- 20 percent.

Mostly very good. Intended as a logic problem, rather than exercise in writing long code.

Some who wrote only player1 did so in a way that worked without change for player2 and player3; awarded marks if appropriate even if no actual player2, player3 function presented. (Similarly, marker player2 when no player3 given.)

Writing code is making a machine. If you put it together wrong, it doesn't work. Please test your code. Please use names as specified in question.

Look over marks and feedback and let me know asap if missed something.

Two simple agents:

```
def agent_simple(state):
    from random import choice
    remaining = [n for n in range(1,N+1) if not n in state]
    for n in remaining:
        if win(state + [n]):
            return n
    return choice(remaining)

def agent_better(state):
    from random import choice
    remaining = [n for n in range(1,N+1) if not n in state]
    for n in remaining:
        if win(state + [n]):
            return n

    for n in remaining:
        if win(state + [0,n]):
            return n

    return choice(remaining)
```

Function exits as soon as it hits a return. Pay attention to indentation.

Solving ODEs

Part 3 of this course is about solving ordinary differential equations.

Will use functions, libraries, arrays, etc.

Quickly review matrices and vectors, then move on to one more component: generators.

Review vectors and matrices; use R(theta) as function which returns matrix.

Generators

Euler's method

- simple teaching aid; don't use for a real problem
- can re-write any nth order ODE as n coupled 1st order ODEs

Assignment 2

More physically tangible problem.
Later part is characterising 50 stars. Clearly, loop over; don't just work out manually for each of the 50.
Will provide some updated hints. May find `format` useful. This is a method of a string.
Formatting a string

```
"x = {x0:.2f} +/- {sx:.2f}".format(x0=1.3531,sx=0.54414)
```

```
x = 1.35 +/- 0.54
```

Code displayed in the lecture

Two ways to make filenames as relevant to assignment 2

```
from glob import glob
filenames = glob("star???.csv")
for f in filenames:
    starname = f[:-4]

for n in range(1,51):
    starname = 'star{:03d}'.format(n)
    f = starname + '.csv'
```

Introducing generators

```
# Functions look like this
def func(a,b,c):
    # some code
    return x,y,z

# Generators look like this
def ostrich(N):
    for n in range(N):
        yield n**2

thing = ostrich(10)
for n in thing:
    print(n)

# Infinite generators are possible
def integers():
    n = 0
    while True:
        yield n
        n = n + 1

# A more interesting infinite generator
def primes():
    n = 0
    while True:
        if isprime(n):
            yield n
        n = n + 1
```

Lecture 3.2 2019-11-11 Mon

Euler's method & notation

Met generators as a tool for solving ODEs.

Rushed Euler's method a little, so explain better today.

Then show how to convert nth order ODEs to n coupled 1st order ODEs (and hence solve any ODE).

Explain Euler's method: "derive", and explain (t_n, x_n) notation.

(Explicit time-dependence in Euler; should have kept it in.)

Companion Form

x_n can be a state-vector i.e. hold all values to describe state of system at time t_n. e.g. Harmonic oscillator position and velocity.

Illustrate companion form with 1D Harmonic oscillator by introducing $v = \dot{x}$

$\partial_t X = f(t, X); X = (x, v)$

Write a function $f(t, X)$ which computes rate of change for every element of X.

Methods

Euler uses "Rectangle rule" to approximate numerical integral; use more accurate approximation?

Many methods. Focus on Runge–Kutta.

Give equation; show code.

Use `rk4` as drop-in replacement for `euler`. Provide `solvers.py` which has these. Use it. e.g. `from solvers import rk4`

Illustrate procedure with flow diagram.

Procedure re-cap

1. Introduce variables to make into coupled first order ODEs
2. Group variables into one state-vector X.
3. Write function which computes **rate of change** for every element of X
4. Glue together with appropriate solver: e.g. `euler`, `rk4`, etc

Reading

Material 3 has worked example of Damped, Driven Harmonic Oscillator. Please go through. Ask if unsure. Workshop tomorrow is practice solving ODEs; also work on Assignment 2 in second hour. Office hours on Friday. Deadline for Assignment 2 is next Monday, 18th November 2019.

Code shown in the lecture

```
def rk4(f, x0, dt):
    """Solve differential equation x'(t) = f(t,x(t))
    using 4th order Runge--Kutta with timestep dt.
    Returns a generator which yields each tn,xn.
    """
    tn = 0
    xn = x0
    while True:
        # gives x0 first, then calculates xn for n>0
        yield tn, xn
        k1 = dt*f(tn,xn)
        k2 = dt*f(tn+dt/2,xn+k1/2)
        k3 = dt*f(tn+dt/2,xn+k2/2)
        k4 = dt*f(tn+dt,xn+k3)
        # use xn = xn + ... (not x += ...) so a numpy array is copied, not changed in-place
        xn = xn + (k1+2*k2+2*k3+k4)/6
        tn = tn + dt


import numpy as np
import matplotlib.pyplot as plt

def f(t,X): # function to embody the damped oscillator
    x,v = X
    xdot = v
    vdot = -a*v-b*x
    Xdot = np.array([xdot,vdot])
    return Xdot

a, b = 0.1, 1
X0 = np.array([1,0])
solver = rk4(f, X0, 0.01)

ts = []; Xs = []
for tn,Xn in solver:
    ts.append(tn)
    Xs.append(Xn)
    if tn > 100:
        break

Xs = np.array(Xs)

plt.figure()
plt.plot(ts,Xs[:,0])
plt.plot(ts,Xs[:,1])

plt.figure()
plt.plot(Xs[:,0],Xs[:,1])

from scipy.linalg import expm
A = np.array([[0,1,0],[0,0,1],[0,0,0]])
X0 = np.array([1,0,-9.81])
t = 0.25
M = expm(A*t)
Xt = M @ X0
```

Lecture 3.3 2019-11-18 Mon

Assignment 2

Well done. That's (probably) Peak Workload for PH204.
You're able to do something meaningful, and you're not yet overloaded with exams.
I'll send some general feedback once I've had a chance to mark.
Next assignment out tomorrow. Is important stuff, and hopefully less work that Assignment 2.

Stability

Recap method using undamped harmonic oscillator
$\ddot{x} + \omega^2 x = 0$
Introduce $v = \dot{x}$ and $X = (x, v)$ therefore $\dot{X} = (\dot{x}, \dot{v}) = (v, -\omega^2 x)$.
For this "state vector", write a function which returns the rate of change. That captures the physics.

```
def f(t,X):
    x,v = X
    xdot = v
    vdot = -omega**2 * x
    Xdot = np.array([xdot,vdot])
    return Xdot
```

We can then use any of the solvers, essentially `euler`, `rk4`, or `rkscipy` (there are others!).
Glue it together like so: `thing = euler(f,X0,dt)` where `X0` is the initial state vector, and `dt` is the time-step.
Let's look at this system's behaviour with `euler`
Blows up: see Material "Stability of numerical methods".
Euler is not stable for the undamped harmonic oscillator.
Proper analysis is beyond the scope of this course.
Briefly:

- Local Truncation error – how wrong it is per step
  - proportional to dt^n to some power ($n = 2$ for Euler; $n = 4$ for RK4)
- Global error – error at some fixed simulation time t
  - error per step of dt^2; number of steps $\propto 1/dt$;
  - global error therefore $\propto dt$ and Euler is called "first order".
- Stability
  - whether it diverges from the correct solution
- Bias
  - consistent under- or over-estimates?

Let's see another example. Simpler one. $\dot{x} + \alpha x = 0$.
Analytical solution $x(t) = x(0)e^{-\alpha t}$. Euler stable if $\delta t < 2/\alpha$.
Show example from Material 3. Show what happens if we switch to RK4.

Other methods

Methods we've seen so far are **explicit** methods: calculate state of system at later time given state now

Implicit methods find a solution by solving the equation at both current and later times i.e. X_{n+1} appears on the RHS as well as the LHS.

There are **specialist** methods with optimizations specific to the equations being solved. e.g. Gravity, or specifically to conserve energy in some system.

There are other ways to code this. See `scipy.integrate` examples for `odeint`. Chose this framework because it makes clear **how** it's being solved, and as a **physicist** you need to know this. It's not magic; it's a method/approach/machine and you need to know when this is an appropriate way to solve a problem.

Frequency response

Can **use** this as a tool to simulate a system. e.g. frequency response of a DHO.

Driven, damped harmonic oscillator: $\ddot{x} + \Gamma\dot{x} + \omega_0^2 x = a\sin\omega_D t$.

Solve with RK.

THEN: wrap in function which takes frequency ω_D and returns the amplitude after some time.

Use this to plot steady-state amplitude as a function of ω_D.

Code illustrating instability of Euler for Harmonic Oscillator

```
import numpy as np
import matplotlib.pyplot as plt

def f(t,X):
    x,v = X
    xdot = v
    vdot = -omega**2 * x
    Xdot = np.array([xdot,vdot])
    return Xdot

omega = 2*np.pi

from solvers import euler, rk4

X0 = np.array([1,0])
dt = 0.002
thing = euler(f, X0, dt) # replace euler with rk4 to compare

ts = list()
Xs = list()
for t,X in thing:
    ts.append(t)
    Xs.append(X)
    if t > 100:
        break

Xs = np.array(Xs)
plt.plot(ts,Xs[:,0],label='x(t)')
```

Code illustrating bias of methods for exponential decay

```
def f(t,x):
    xdot = -alpha * x
    return xdot

alpha = 1
x0 = 1

plt.figure()
```

```
for dt in [0.1,1.9,2.1]:
    thing = euler(f,x0,dt)     # or replace euler with rk4
    ts = list(); xs = list()
    for t,x in thing:
        ts.append(t)
        xs.append(x)
        if t > 10:
            break
    plt.plot(ts,xs,'*-',label='dt={:.2f}'.format(dt))

def analytical(t):
    return np.exp(-alpha*t)
ts = np.linspace(0,10,1001)
ys = [analytical(ti) for ti in ts]
plt.plot(ts,ys,'k',label='Analytical')
```

DHO Case study code

```
import numpy as np
import matplotlib.pyplot as plt
from solvers import rk4

pi = np.pi

def DHO(omegaD):
    print(omegaD)
    def f(t,X):
        from numpy import array, cos
        x,v = X
        xdot = v
        vdot = a0*cos(omegaD*t) - Gamma*v - omega0**2 * x
        return array([xdot, vdot])

    omega0 = 2*pi*10
    Gamma = 2
    a0 = 100
    dt = 1e-3
    X0 = np.array([0,0]) # start the oscillator at rest
    solver = rk4(f, X0, dt)
    ts = list()
    Xs = list()
    for t,X in solver:
        ts.append(t)
        Xs.append(X)
        if t > 10:
            break
    ts = np.array(ts)
    Xs = np.array(Xs)

    Energy = Xs[-1,0]**2 + (Xs[-1,1]/omega0)**2
    return Energy

omegas = np.linspace(2*pi*5,2*pi*15,101)
ys = [DHO(omega) for omega in omegas]
```

Lecture 4.1 2019-11-25 Mon

Assignment 3

ODEs.

Chaos: straightforward

Kapitza: be sure to sample with sufficient time resolution, use an appropriate method, and think about what the result means

Penning: combines vectors (esp. cross-product) and ODE solving. Use a 6-element state-space (x, y, z, v_x, v_y, v_z). E is a function of position.

Penning is technologically important. Kapitza is surprising and interesting.

Emphasis in this Assignment is the core ideas. You've seen that we can do much more. For example, once we have a working Penning trap simulator, we could ask what *temperature* of particle it could confine, by running $\gtrsim 10^4$ simulation with particles picked from a thermal distribution and seeing what fraction remain after some time; and then doing this for a range of different temperatures.

Onwards

SDEs

- describe *thermal* systems from microscopic
- thermal is link from neat theories to real world
- can e.g. simulate particle in potential well and find thermal distribution
- also used extensively in financial modelling

Boundary value problems

- sketch trajectory
- constrained by angle; misses too short; too long; use `newton`
- there's more than one solution

Partial differential equations

- Laplace with votlage known on surfaces
- sneakily link to Penning trap electric field

Functional

- lambda might make more sense to some for mathematical expressions
- show in `quad` or `newton`
- `map` is like list comp
- `filter`; `reduce` (e.g. factorial)

itertools - much neater than nested for loops

multiprocessing - 16x speed up!

classes

- everything is an object
- this is how to make our own
- key "private" functions: init, repr, str

Lecture 4.2 2019-12-02 Mon

Module feedback

Outlook

Nearly finished.
Today: symbolic.
Final lecture: case-study showing `class`, `multiprocessing`, and MCMC.

Broadcasting

Broadcasting: hand as much to `numpy` as possible.
e.g. Δx for all combinations in a 1D array to make a 2D array.
Introduce `%timeit` in IPython

```
import numpy as np
A = np.random.random(1000)
# The following timed by prefixing with "%timeit" in IPython
D = A[:,None] - A[None,:]                    #   1.2ms
D = np.array([A - a for a in A])             #   2.9ms
D = np.array([[b-a for a in A] for b in A]) # 240  ms
```

Symbolic

Change of style. Doing not just numerical, but **symbolic** mathematics.

```
import sympy as sp
x,y,z = sp.symbols('x y z')
```

Create `sympy` objects to represent x, y, z. n.b.

```
q = x
q
```

```
x
```

because `q` points to the same thing as `x` and `x.__repr__` returns the string "x".

- Solve quadratic
- illustrate $f == g$ and `simplify(f-g)`
- differentiation
- integration
- limits
- N
- replace
- summation
- combine with usual python e.g. `lambda`
- solving differential equations

Lecture 4.3 2019-12-09 Mon

Assessment

Assignment 3 finished. Only Comp4 remains. Essentially some basic sympy. Integration. Differentiation. Solving. Matrix.

Example code for Assignment 3 Q3 starting with $2x3$ state-space for $[[x, y, z], [v_x, v_y, v_z]]$.

Randomness

Explore extremely large configuration space by sampling.

Simple example. Peer marking. Class size N, what's the probability that someone gets their own script?

Larger N means less chance that any individual gets their own script, but there are more individuals. What's the probability oveall?

Can't enumerate all possibilities, because for $N = 70$ there are $\sim 10^{100}$ configurations.

Make a function which returns numbers in a random order:

```
from random import shuffle
def mixed(N):
    cards = list(range(N)) # list of numbers 0 to N-1
    shuffle(cards) # shuffle in-place!
    return cards   # return the shuffled list

mixed(10)
```

```
[3, 5, 9, 1, 0, 2, 4, 6, 8, 7]
```

Test to see whether any number n occurs at the n^{th} position:

```
N = 10
m = mixed(N)
[n == c for n,c in enumerate(m)] # describe enumerate
```

```
[False, False, False, False, False, False, False, False, False, False]
```

Wrap this in a function which returns `True` if any card is at its n^{th} position:

```
def hit(N):
    m = mixed(N)
    return any([n == c for n,c in enumerate(m)])

def hit(N):
    m = mixed(N)
    for n,c in enumerate(m):
        if n == c:
            return True
    return False
```

Average over lots of trys:

```
N = 100
sum([hit(N) for i in range(1000)])/1000
```

```
0.641
```

Analytical result: $1 - 1/e \approx 0.63212056$.

Very often there is no known analytical result!

Ising case study

More generally, have a system described by a state and some transition probability to a new state.

Markov Chain Monte Carlo explores this system by considering changes to the state and accepting them depending on the transition probability.

Ising model: 2D model of a magnet. Square grid of "spins" up or down. Energy associated with a spin depends on neigbours.

Define as a class. Random 2D square array. Compute magnetisation. Produce readable description. Flip a spin.

Algorithm: pick a spin at random. Flip if reduces energy. If flipping would increase energy, accept with probability depending on temperature.

Iterate lots of times. Make animation by combining lots of PNGs. (Maybe 100 iterations per frame?)

Much faster algorithms. Challenge is to find an algorithm which is efficiency and still accurately captures the physics.

Just for fun

Write a Python script which prints its own source code.

Write a sudoku solver.

Write solvers for the Countdown numbers and/or letters games.

See Project Euler.

Calculate the number of unique ways to make £2 using the coins currently in UK circulation. (more)