Practical 06: Plotting and the Verlet integrator Documentation

Release 1.0

Oliver Beckstein

February 07, 2013

CONTENTS

Contents:

CHAPTER

ONE

PRACTICAL 06

1.1 IPython and pylab

Start the IPython Python shell with

```
ipython
```

If you want to do interactive plotting, start it with the `--pylab` flag:

```
ipython --pylab       # default
ipython --pylab=osx   # Mac OS X "backend"
ipython --pylab=qt    # QT backend
```

IPython is incredibly useful and can do many, many useful and cool things (see their web page). IPython is tightly integrated with NumPy and matplotlib. A small selection that you can use right away:

- help: `?`
- source code: `??`
- TAB completion
- special "magic" functions:
  - `%history` (`%history -f output.py` writes session to file)
  - `%run` (runs a python file)
  - `%pdb` (switches on debugger)
  - `%time` (timing)

Example for plotting with the `plot()` function:

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

X = np.arange(-5,5,0.1)
Y = np.sin(X**2)/X**2

plt.plot(X, Y)
plt.xlabel("$x$")
plt.ylabel("sinc$^{2}x$")
plt.title("using matplotlib")
plt.savefig("sinc2.pdf")    # pdf format
plt.savefig("sinc2.png")    # png format
```

```
plt.clf()

plt.close()
```

See Also:

- `numpy.arange()` and `numpy.linspace()`
- `numpy.sin()` is a NumPy Universal function (ufunc); get help with `help(numpy.sin)` or `numpy.info(numpy.sin)` or in ipython, `numpy.sin ?`
- `matplotlib.pyplot.savefig()`
- `matplotlib.pyplot.clf()`
- Text rendering with LaTeX

Look at the figure from the command line; in Mac OS X you can use the `open` command

```
open sinc2.pdf
open sinc2.png
```

In Linux different commands are available, depending on yur distribution, e.g. `display`, `eog`, ... (for images), `xpdf`, `okular`, ... (for pdf).

1.2 Potential and forces

Harmonic force on a particle, 1D:

- Force F = -kx
- Potential energy: U = k/2 x**2
- equations of motion:
  - a = F/m = -k/m x
  - d2xdt2 + k/m x = 0 (harmonic oscillator)
  - d2xdt2 + omega**2 x = 0
  - omega = sqrt(k/m)

  Note: Hamiltonian (*not needed*)

  H = p**2/2m + 1/2 k x**2 = p**2/2m + 1/2 m omega**2 x**2

1.2.1 Preparation

Download the file `integration_v0.py` from http://becksteinlab.physics.asu.edu/pages/courses/2013/SimBioNano/06/

```
curl -O http://becksteinlab.physics.asu.edu/pages/courses/2013/SimBioNano/06/integration_v0.py
```

or

```
cdl 06 integration_v0.py
```

and rename to `integration.py`. Work in this file (see the comments in the file).

You can open two terminals in parallel, one for a vi session to edit the file, the other one running `ipython`. In `ipython` you can load your `integration.py` as a module:

```
import integration

# example
U = integration.U_harm(1.0,  integration.kCC)
```

Note that changes in the file are *not automatically picked up in ipython*, you have to use the special `reload()` command:

```
reload(integration)
```

If you put commands in another file, e.g. `plot_U.py` you can run this file from `ipython` with

```
%run plot_U.py
```

1.2.2 Task 1

Sketch a point mass with a spring. What is `x` in our definition of the potential `U` above? How does this relate to our previous picture that includes the equilibrium bond/spring length b0?

- write a function `U_harm(x, k)` which returns the potential energy (in kJ/mol).

  Plot the function with `plot()` for the two values of k:

  - `kCC = 1860e2` (in kJ/(mol*nm**2))
  - `kHC = 2760e2`

  over `-0.15 <= x <= 0.15` (nm). Label the axes.

- write a function `F_harm(x, k)` that returns the force (in kJ/mol*nm)

  Plot it for a range of x values.

1.3 Integrators

Euler:

> xnew = x + v dt + F/(2m) dt**2

Verlet:

> xnew = 2x - x0 + F/m dt**2

1.3.1 Task 2

Write functions

- `verlet(x, x0, F, dt, m)`
- `euler(x, v, F, dt, m)` [* only if time, see *Task 5*]

which return the new position. (We'll test them in *Task 3* but you should make sure that they return some sensible values for simple input e.g. x=0, v=0. Masses will be taken in `u` and are on the order of 1 to 10.)

1.3.2 Task 3

Use the function `integrate_verlet()` in file `integration_v0.py` (available from) as a basis and fill in the missing parts.

```
# some predefined values for masses and bond force constants
#-------------------------------------------------------------
# masses in atomic units u (note: 1 u = 1g/mol)
mCC = 6.
mHC = 12./13.

# force constants in kJ/(mol*nm**2)
kCC = 1860e2
kHC = 2760e2

def integrate_verlet(x0, v0, dt, nsteps=100, m=mCC, k=kCC):
    """Integrate harmonic oscillator x.. + k/m x = 0.

    :Arguments:

    * x0: starting position (in nm)
    * v0: starting velocities (in nm/ps)
    * dt: time step (in ps)

    :Keywords:

    * nsteps: number of integration steps (100)
    * m: reduced mass in atomic mass units u (default is for a C-C
      bond)
    * k: harmonic force constant in kJ/(mol*nm**2) (default is for
      a simple C-C bond)

    Returns trajectory as NumPy array:

       [time/ps, x/nm, U(x)/kJ/mol]
    """

    print("Starting Verlet integration: nsteps = %d" % nsteps)
    x = x0
    # bootstrap: generate previous point
    x0 = x - v0*dt
    # store coordinates and time: trajectory
    trajectory = []
    for istep in xrange(nsteps):
        # --- calculate force F ----
        # --- calculate new position xnew ---
        trajectory.append([istep*dt, x, U_harm(x, k)])
        x0 = x
        x = xnew
    print("Integrator finished.")
    return np.array(trajectory)
```

In the following we are always using m=mCC and k=kCC.

- Run for 1000 steps with a timestep of 0.001 ps = 1 fs:

  ```
  trj1 = integration.integrate_verlet(0.1, 0, 0.001, nsteps=1000)
  ```

  Plot (t, x(t)):

```
plt.plot(trj1[:,0], trj1[:,1], linewidth=2, label="dt=1fs")
```

Zoom in on the region:

```
plt.xlim(0, 0.2)
```

Add labels:

```
plt.xlabel(r"time $t$ in ps")
plt.ylabel(r"position $x$ in nm")
```

and a title:

```
plt.title("Harmonic oscillator integrated with Verlet")
```

Add legend:

```
plt.legend(loc="best")
```

Save figure:

```
plt.savefig("verlet.pdf")
plt.savefig("verlet.png")
```

Look at the figure from the command line; in Mac OS X you can use the `open` command

```
open verlet.pdf
open verlet.png
```

1.3.3 Task 4

1. If you have time, compute the potential, kinetic and total energy at each step and plot them.
2. Plot the phase space trajectory.

1.3.4 Task 5

If you have time, investigate the Euler integrator in the same way as in Tasks 3 and 4.