Numerical simulations of gravitational dynamics

Eugene Vasiliev

Oxford University &
Lebedev Physical Institute

XII School of modern astrophysics 2016

Seminar 1.

Overview of *N*-body simulation software and the AMUSE framework

Types of software for performing N-body simulations

Typical workflow for numerical experiments:

0. Formulate the physical problem to explore.
1. Set up the initial conditions.
2. Run the dynamical simulation.
3. Analyze the results (consistency checks, scientific output).
4. Prepare the data for publication (tables, figures and movies).

Each of these steps, in general, requires a different software,
and the data needs to be stored and exchanged between steps.

Preparing initial conditions

- Planetary dynamics:
  small number of bodies – initialize directly (e.g., using JPL ephemerides for Solar system).
- Star clusters:
  typically use analytic spherical models (Plummer, King, etc.) with a known distribution function, and the rejection sampling method to draw particles from it; often this is built into the N-body integrator.
- Galaxy evolution or mergers:
  various methods for creating the initial conditions in dynamical equilibrium, with specific tools for each method (example: Schwarzschild orbit-superposition method – `smile` software).
- Cosmological simulations:
  compute the power spectrum of initial fluctuations (e.g., `cmbfast`, `camb`), then convert it to the density and velocity fields (e.g., `grafic`).

N-body simulation codes

- Planetary dynamics:
  Swift; Mercury; Rebound.
- Star clusters:
  NBODYx series of codes from Sverre Aarseth
  ($x = 1..7$, the parallel version is NBODY6++);
  kira from the Starlab framework; HiGPUs and ϕGRAPE
  (both require hardware acceleration, included in AMUSE framework).
- Galaxy and cosmological simulations
  (collisionless, often include hydrodynamics):

  GADGET-2; PKDGrav2/Gasoline;
  gyrfalcON (included in NEMO framework);
  Bonsai (GPU-accelerated, included in AMUSE);
  RAMSES; Enzo; FLASH; Athena; Gizmo;

  non-public but well-known:
  ART, GADGET-3, AREPO, HACC, GOTPM, 2HOT, ...

N-body analysis tools and frameworks

- `yt` – Python-based visualization package.
- `pynbody` – another Python-based environment mainly for analysis of cosmological simulations.
- `Starlab` – framework for collisional dynamics (collection of individual programs coupled in a UNIX toolchain style).
- `NEMO` – framework for collisionless dynamics with a similar architecture; see also a stand-alone interactive 3d visualization program `glnemo2`.
- `AMUSE` – Python-based meta-framework that enables interaction between many separate simulation codes.

Python as the glue language

+ De-facto standard in the wider astronomical community, alternative to proprietary software such as IDL or MATLAB.
+ Widespread also in a more general context, plenty of resources in the internet.
+ Well suited both for quick coding of short one-time tasks and for development of complex applications.
– Intrinsically less efficient in CPU- or memory-intensive tasks,
+ however this is compensated by third-party libraries (notably `numpy`), and by the possibility of coupling with closer-to-hardware languages.
+ High-quality plotting facilities (`matplotlib`).

- Most convenient in data analysis and visualization tasks, but also can be used as the glue language (e.g., in `AMUSE`).
- Can be used both in the interactive or scripting modes.

Architectures of simulation software

1. Historically, most scientific software was developed as monolithic codes including all required physical processes, quickly growing in complexity.

2. Alternatively, software frameworks such as `NEMO` and `Starlab` provided a large collection of relatively small tools for data manipulation, common data exchange formats and parameter handling conventions. But still the core simulation programs are monolithic.

3. Analysis and visualization of existing simulation data can be done within self-contained environments such as `yt`, `pynbody`.

4. Run-time coupling of *existing independent* simulation codes is a very challenging task – but it has been successfully implemented in the AMUSE framework.

AMUSE framework: rationale

Task:

- To have a unified interface for a variety of "community" codes, with the possibility of changing a single line in the script to use a different simulation code.
- To enable the run-time interaction of several independent codes from different domains (e.g., coupling of gravity and hydrodynamic solvers, or a collisional simulation of a star cluster moving in the gravitational field of two merging galaxies).

Challenges:

- Different data representation strategies, unit systems, coding conventions and languages.
- Large spatial and temporal dynamic range, different time integration approaches.
- Reasonably low overhead (both CPU and communication).

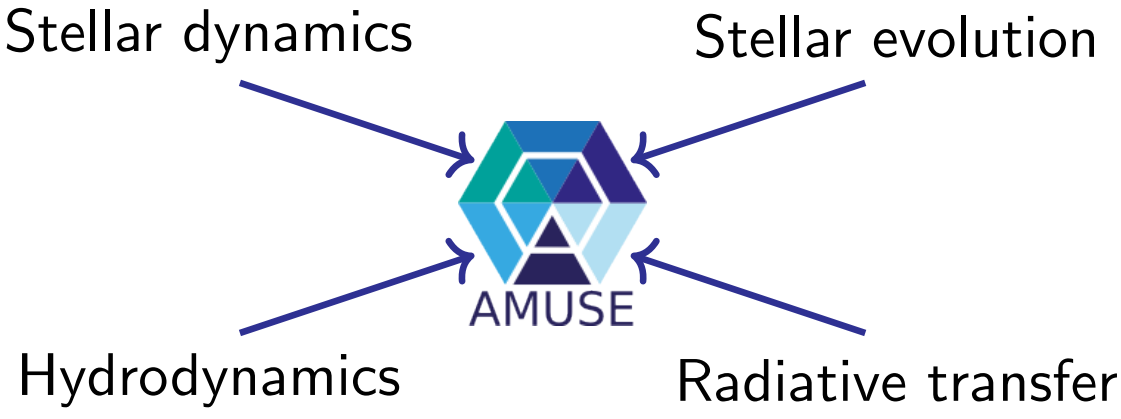
AMUSE framework: architecture

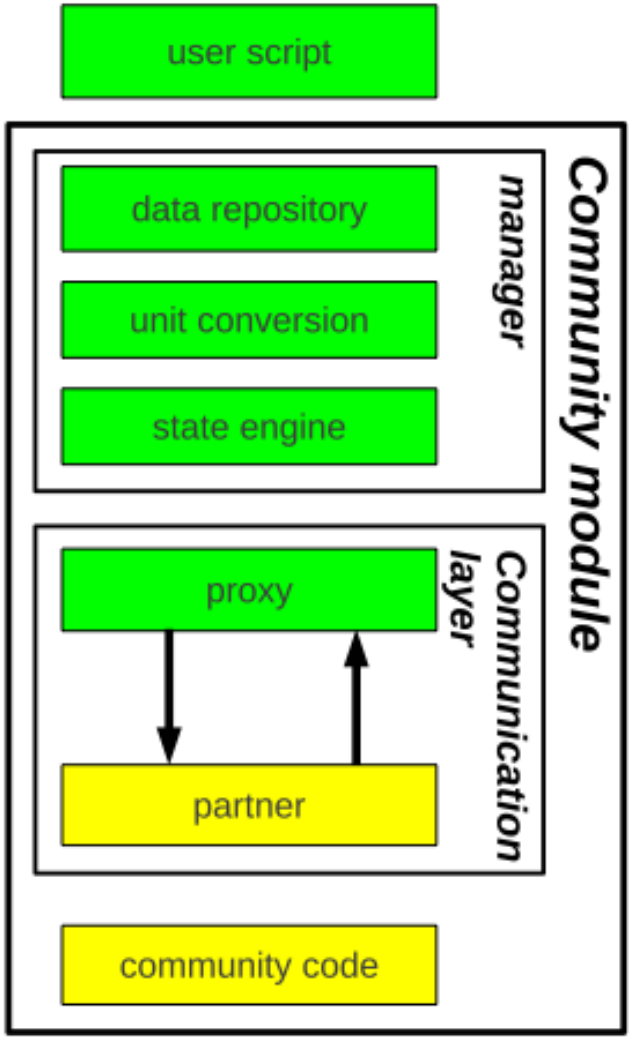
- User script or interactive session in Python.
- The library data handling layer.

MPI

- Wrapper module for the community code.
- The actual community code (with minor modifications).

Physical domains:





Interlude: guidelines for [scientific] programming

- Coherent coding style
  (programming paradigm, naming conventions, choice of libraries, . . .)
- In-code documentation
  for both the interfaces and most important implementation details.
- Test suites and examples of usage.
- Layered approach for the design of the program;
  clear and efficient definition of abstractions and interfaces,
  allowing for interchangeability of individual blocks (loose coupling).

Hands-on session #1
Installing and running AMUSE

(with updates following the chaotic experience of the first day).

Basic facts about Python

Python can be used in either interactive or scripting modes:

- `bash$ python`
  `Python 2.X.X blabla`
  `>>> _`
  now you may type your commands here and they will be executed one-by-one.
  More conveniently, you may start the interpreter with the paths to the AMUSE library correctly set up, by typing `amuse` instead of `python` (this runs a simple shell script that you may find in the AMUSE root directory), or you may simply add the relevant directories to your PYTHONPATH environment variable permanently (e.g. in bash.rc).
- `bash$ python script.py`
  runs the given script non-interactively (or do it with `amuse` command).
- Finally, you may load and execute the script from within the python interactive session by typing
  `>>> execfile("script.py")`

Basic facts about Python (2)

- It is also possible to run an interactive Python script from a browser (called IPython notebook – it is launched e.g. by `amuse-tutorial` program).
- Note that either way, when you start `python` or `amuse`, you still need to import AMUSE and other libraries at the beginning of your script, e.g., as
  `>>> from amuse.lab import something`
  (then you may use `something` without a namespace prefix), or
  `>>> import amuse.lab, numpy, matplotlib, ...`
  (then you will need to add `amuse.lab.` prefix to `something`).
- Python commands need to fit into a single line, or alternatively continue from the following line after a backslash put at the end of the previous one (as used in the first example scripts to break up long lines). If you merge the split lines, you need to delete the \ symbol.
- # is the comment symbol (until the end of line), triple quotes """ open and close a multi-line comment block.

Basic facts about Python plotting library

The standard plotting library is `matplotlib`, which has a module named `pyplot` with a very similar syntax to MATLAB. So you will typically use the sequence of commands like

```
import matplotlib.pyplot as plt
plt.figure(figsize=(10,10))
plt.plot(xcoords, ycoords, '.')  # plot points not lines
plt.show()
```

with more formatting commands possibly inserted before `show()`.

The `show()` command opens an interactive plotting window that will block any further execution of the script until it is closed.

Alternatively, you may save the plot into a png or pdf file:

```
plt.savefig("figure.png", dpi=100)
```

Or if you want to use Gnuplot from within Python, you need the `gnuplot-py` package (download and install separately);
finally, you may export the data to a text file and plot it with Gnuplot or your favourite plotting program.

Troubleshooting

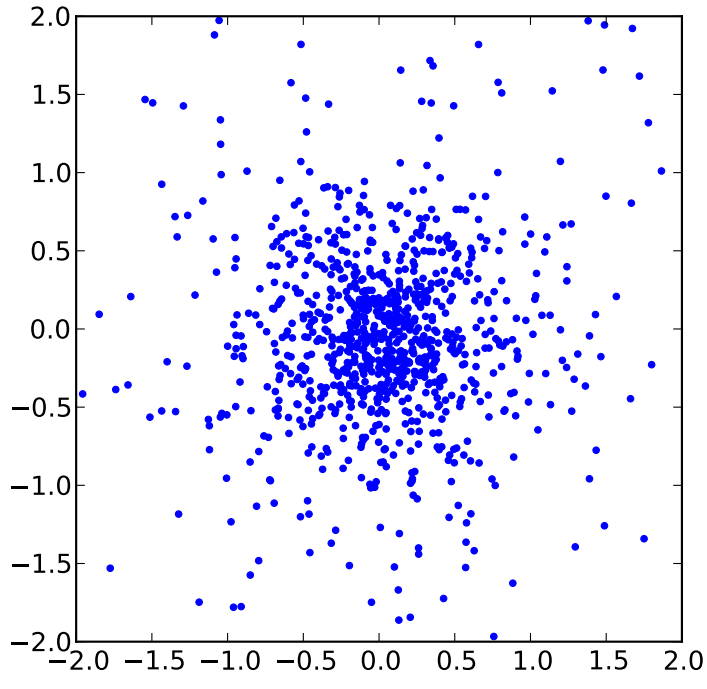
- Some people have reported that `matplotlib.pyplot.show()` command does nothing. This may be fixed by choosing a different output back-end, adding the following lines at the beginning of the script:

```
import matplotlib
matplotlib.use('webagg')  # or maybe 'tkagg'
```

- Scripts further down in this pdf file may mess up some symbols (quotes and extra spaces) – you may need to edit it after copypasting.
- The AMUSE module for exporting binary NEMO files is buggy. Delete the files `nemobin.py` and `nemobin.pyc` in `amuse/lib/python2.7/site-packages/amuse/io/` and replace the former with the corrected file attached at the school webpage. You only need it if you want to use the `glnemo` program for visualizing the simulation data (quite handy).

Example AMUSE script (1a)

```
#!/usr/bin/python
"""
Example AMUSE script to plot a Plummer sphere
"""
import amuse.lab, amuse.plot, pylab, matplotlib
bodies = amuse.lab.new_plummer_model(1000)
pylab.figure(figsize=(5,5))
amuse.plot.plot(bodies.x, bodies.y, '.')
pylab.xlim(-2, 2)
pylab.ylim(-2, 2)
pylab.show()
```



Same example using Gnuplot (1b)

```
#!/usr/bin/python
"""
Example AMUSE script to plot a Plummer sphere
"""
import amuse.lab, Gnuplot
bodies = amuse.lab.new_plummer_model(1000)
plot=Gnuplot.Gnuplot()
plot("set terminal x11")  # enable interactive
plot("set ticslevel 0")   # 3d plotting window
plot("set xrange [-2:2]")
plot("set yrange [-2:2]")
plot("set zrange [-2:2]")
# need to manually convert an array of dimensional
# quantities into a simple python array of numbers
plot.splot(bodies.position.value_in \
    (amuse.lab.nbody_system.length))
raw_input()  # pause execution
```

Working with units (2)

```
#!/usr/bin/python
"""
Create a Plummer model in physical units
"""
from amuse.lab import units, \
    nbody_system, new_plummer_model
Mass   = 1e5 | units.MSun
Radius = 10  | units.parsec
bodies = new_plummer_model(1000, \
    nbody_system.nbody_to_si(Mass, Radius))
Ekin   = bodies.kinetic_energy()
Epot   = bodies.potential_energy()
print "Ekin = %f Msun*(km/s)^2" % \
    Ekin.value_in(units.MSun*units.kms**2)
print "Epot = %f Msun*(km/s)^2" % \
    Epot.value_in(units.MSun*units.kms**2)
print "Virial ratio = %f" % (-2*Ekin/Epot)
```

Assignments

0. Download and install AMUSE from http://amusecode.org
Run the example scripts (1a and 2).

A*. (optional) Download glnemo2 visualization program from https://projets.lam.fr/projects/glnemo2,
export the generated Plummer model to a NEMO file using

```
amuse.io.write_set_to_file(bodies, \
        "plummer.nemo", format="nemobin")
```

then load and display this file in glnemo2.

1. Estimate the density profile of the generated Plummer model, compare to the expected functional form

$$\rho(r) = \frac{3}{4\pi}\frac{M}{r_0^3}\left(1+\frac{r^2}{r_0^2}\right)^{-5/2} \qquad \text{with } M=1, r_0 = 3\pi/16$$

either by eye, or using a rigorous statistical test (e.g., Kolmogorov–Smirnov).

Notes

- Assignments with a letter and an asterisk are optional (i.e. do not bring bonus points, but only the customer satisfaction...)
- You may use any of your favourite plotting programs to display results (`gnuplot` and `glnemo` are just two suggestions, but they are optional). Make sure to learn how to export particles at least to a text file (`format="txt"`).
- When estimating the density profile from particles using a histogram, remember that this is a 3d distribution (hint: need to divide the number of particles by the volume of the bin, not by the radius).
- K–S tests are designed for the cumulative mass profile, and work best without any binning.

Numerical simulations of gravitational dynamics

Eugene Vasiliev

Oxford University &
Lebedev Physical Institute

XII School of modern astrophysics 2016

Seminar 2.

First experiments with N-body integrators in AMUSE

Some of the N-body integrators available in AMUSE

name	type	timestepping	language	CPU/GPU	parallel[1]
Hermite	direct	shared, var.	C++	+/−	+
ph4	direct	block	C++	+/+	+
NBODY6++	direct	block+reg	F77	+/?	?
PhiGRAPE	direct	block	F77	+/+	+
HiGPUs	direct	block	C++	−/+	+
Huayno	sympl.	block	C	+/+	−
Mercury	MVS	fixed/adaptive	F77	+/−	−
BHTree	tree	shared, fixed	C++	+/−	−
Gadget2	tree	block	C	+/−	+
Bonsai	tree	block	C++	−/+	+
Ramses	grid	block	F90	+/−	+

Note: not all integrators are available on all systems, and not all of them are suitable for all problems.

[1] by default, all codes are run in a single-thread mode; to enable parallel execution, you need to construct the instance of the code with an extra parameter such as `number_of_workers=4`.

Assignments – star clusters

2. Evolve a Plummer model with $N = 1000$ for several $T_{\rm dyn}$, using as many variants of N-body integrators as possible.

Experiment with the parameters of integrators as well (fixed timestep size or accuracy parameter η for variable timestep, smoothing length, etc.), determine the order of integrator(s).

Perform basic consistency checks: accuracy of energy and momentum conservation, virial ratio.
What are the suitable values of timestep for different methods?
Which integrator is the most efficient?

B*. Modify the example script to evolve the system for several intervals of time, recording the diagnostic information after each sub-interval. Explore the growth of error as a function of time for various methods, explain your findings.

C*. Display the evolution of the system as a movie or a sequence of images, or using `glnemo` visualization program
(will need to save the output as a series of text or NEMO files, or create an animation directly from Python).

Assignments – planetary systems

3. Create an N-body model of the Solar system, using the positions/velocities of 8 planets at the present time (take from JPL website, or from the AMUSE module `amuse.ext.solarsystem`). Evolve it forward for 1000 years with a suitable integration method(s), check the energy conservation and make sure that the system remains stable. Plot the evolution of eccentricities of all planets (use the `Kepler` module from `amuse.lab` to compute the orbital elements, see the examples in AMUSE primer).

D*. Now let's play god and imagine that the planets have been formed more massive. (It's unlikely from astrophysical grounds but we'll ignore it). Multiply planetary masses (but not the mass of Sun) by a factor of 10,100,... and repeat the integration for 100–1000 years. At which point does the system become unstable? what happens to it?
(Hint: consider the Hill radii of each planet).

Basic script for simulating a *N*-body system

```
#!/usr/bin/python
import amuse.lab
bodies   = amuse.lab.new_plummer_model(100)
totaltime = 1.0 | amuse.lab.nbody_system.time
gravity = amuse.lab.ph4()
gravity.parameters.timestep_parameter = 0.01
gravity.particles.add_particles(bodies)
Etot_init = gravity.kinetic_energy + \
            gravity.potential_energy
gravity.evolve_model(totaltime)
Ekin = gravity.kinetic_energy
Epot = gravity.potential_energy
Etot = Ekin + Epot
print "Time=", gravity.get_time()
print "Virial ratio=", (-2*Ekin/Epot)
print "dE/E=", (Etot_init-Etot)/Etot
gravity.stop()
```

Notes

- **Hint:** read the AMUSE primer for more details and examples.
- If you use different *N*-body integrators, you will discover that they often have inconsistent parameter naming schemes, e.g. what is `timestep_parameter` in `ph4`, becomes `dt_param` in `hermite` (use `print(gravity.parameters)` to find it out, or check the description on the website).
- Also, make sure that you use the same initial snapshot for all of them (since `new_plummer_model` creates a different realization each time).
- All dimensional quantities in AMUSE must be explicitly assigned a correct dimension (even if using "standard" *N*-body units), as follows:
  `time = 0 | amuse.lab.nbody_system.time`
  When dealing with physical units, you need to create an appropriate converter object and provide it as an extra parameter:

```
converter = amuse.lab.nbody_system.nbody_to_si( \
    1 | amuse.lab.units.MSun, 1 | amuse.lab.units.AU)
gravity   = amuse.lab.Mercury(converter)
```

  To get an ordinary number from a dimensional quantity, use `position.value_in(amuse.lab.units.km)`

Notes

- Remember that the AMUSE framework and the N-body codes live in separate worlds: particles created in the script and fed to the integrator using `add_particles` are not automatically updated when the N-body system is evolved. You need to open a communication channel and update them manually:

```
gravity.particles.add_particles(bodies)
channel = gravity.particles.new_channel_to(bodies)
gravity.evolve_model(time)
channel.copy()  # now bodies contain updated data
```

- The `evolve_model(time)` method of a gravity code advances the system *up to the given time*, not *by the given time*, i.e. if you call it once again with the same argument, it does nothing since the internal clock is already at the required position.

Numerical simulations of gravitational dynamics

Eugene Vasiliev

Oxford University &
Lebedev Physical Institute

XII School of modern astrophysics 2016

Seminar 3.

Simulating a sinking satellite

Dynamical friction

Drag force acting on a massive body M moving with velocity v in the background of lighter objects with density ρ:

$$M\dot{v} = -\frac{4\pi\, G^2\, M^2\, \rho\, \ln\Lambda}{v^2}.$$

We simulate the sinking of a satellite to the center of a galaxy.

The main galaxy is spherical with a "broken power-law" density profile $\rho \propto r^{-\gamma}\,(r^\alpha + r_0^\alpha)^{(\gamma-\beta)/\alpha}$;
the mass of the satellite $M_{\rm sat}$ is $M_{\rm gal} \gg M_{\rm sat} \gg M_{\rm gal}/N \equiv M_{\rm part}$.

In the first example, we model the satellite as a point mass.

Dynamical friction: initial conditions

```
#!/usr/bin/python
from amuse.lab import *
converter = nbody_system.nbody_to_si( \
    1e11 | units.MSun, 10. | units.kpc)
# initial conditions for the main galaxy
galaxy    = new_halogen_model(10000, \
    converter, alpha=1, beta=6, gamma=1)
# initial position and velocity of the satellite
satellite = Particle()
satellite.mass = 5e9 | units.MSun
satellite.position = [10., 0, 0] | units.kpc
satellite.velocity = [0, 160, 0] | units.kms
particles = Particles()
particles.add_particle(satellite)
particles.add_particles(galaxy)
```

Dynamical friction: parameters of the integrator

```
# N-body integrator: use a tree code
gravity = BHTree(converter)
# set the softening length
gravity.parameters.epsilon_squared = \
    (0.5 | units.kpc)**2
# set the (constant) timestep
gravity.parameters.timestep = 0.5 | units.myr
gravity.particles.add_particles(particles)
# open a communication channel from code to AMUSE
channel = gravity.particles.new_channel_to( \
    particles)
```

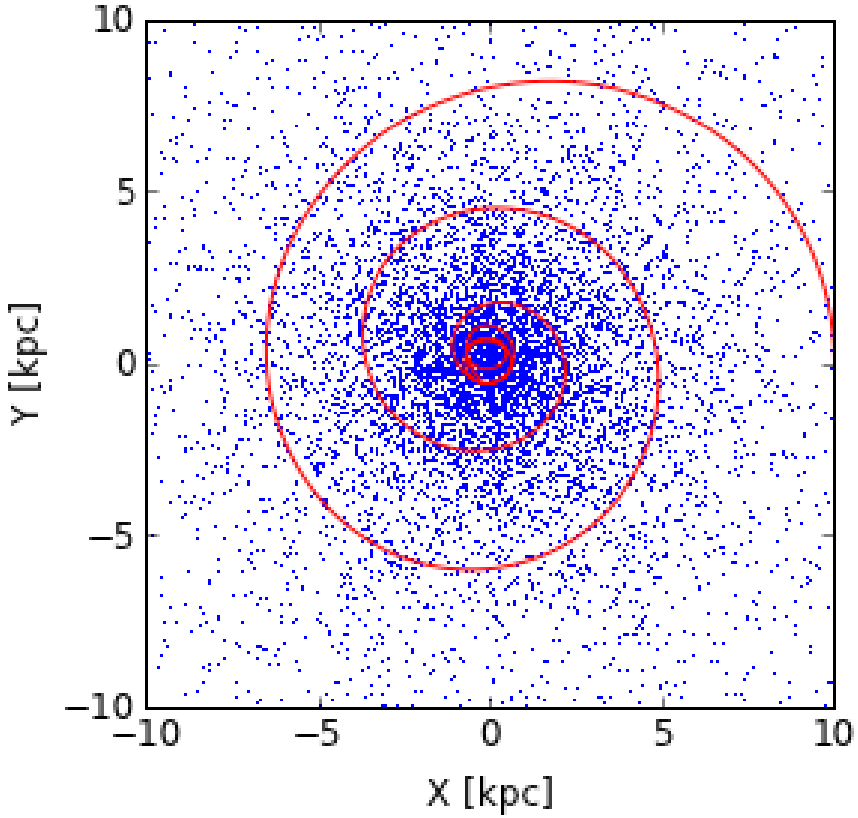
How to choose an appropriate softening length and timestep?
$\epsilon \simeq (\rho/M_{\rm part})^{-1/3}$, $\Delta t \lesssim \epsilon/v$, $v \lesssim v_{\rm escape} \equiv \sqrt{-2\Phi(0)}$

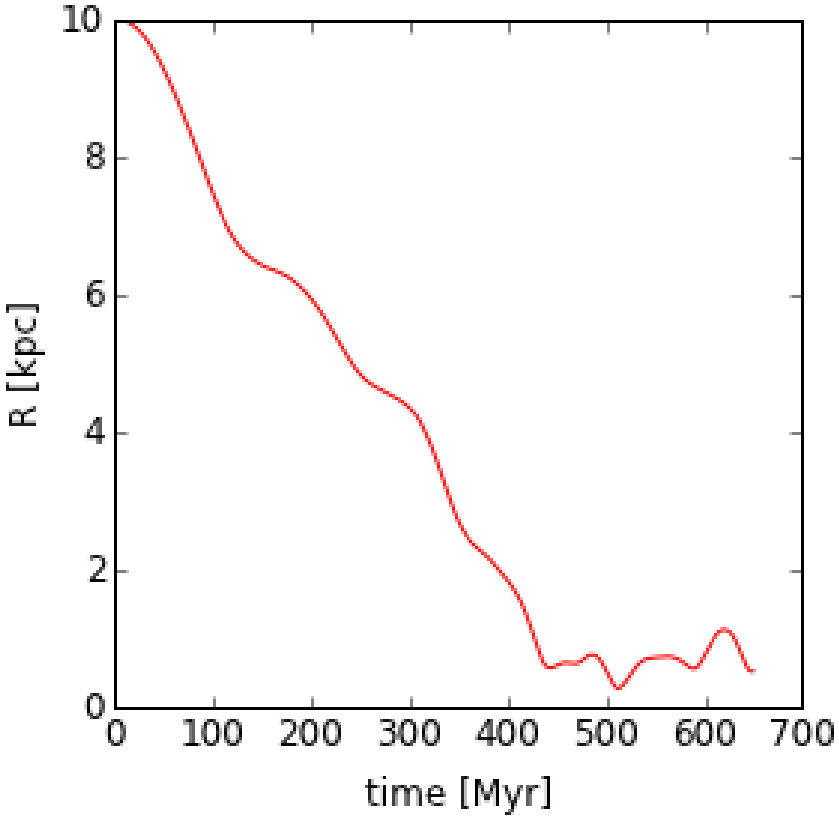
```
print "Escape velocity:  ", \
((-2*min(galaxy.potential()))**0.5).value_in(units.kms)
```

This gives $v_{\rm escape} \simeq 500$ km/s and thus Δt must be $\lesssim 1$ Myr.

Dynamical friction: assignment

4. Complete the script with the loop that evolves the model for few$\times 10^8$ yr (split into many smaller intervals of time, storing the position of the satellite and the galaxy center after each interval). Plot the trajectory of the satellite and distance from the galaxy center as a function of time.
   Compare with the predictions of Chandrasekhar's formula.





Numerical simulations of gravitational dynamics

Eugene Vasiliev

Oxford University &
Lebedev Physical Institute

XII School of modern astrophysics 2016

Seminar 4.

Simulating a tidally-disrupted satellite

Code coupling in AMUSE

We want to model a globular cluster or a satellite galaxy moving in the potential of the main galaxy.

The internal dynamics of the cluster should be followed with a high-accuracy direct-summation code, and the dynamics of the galaxy – with a more approximate tree code, and we want them to interact.

In AMUSE this can be realized with the concept of bridge integrator, which uses operator-splitting approach (similar to a symplectic integrator):
$\mathcal{H}_{\mathsf{full}} = \mathcal{H}_{\mathsf{first}} + \mathcal{H}_{\mathsf{second}}$, where each part is evolved by an appropriate integrator, and the interaction between subsystems is implemented by kicking particles of `first` subsystem periodically with the force from the `second` subsystem and vice versa.

Bridge simulation: initial conditions

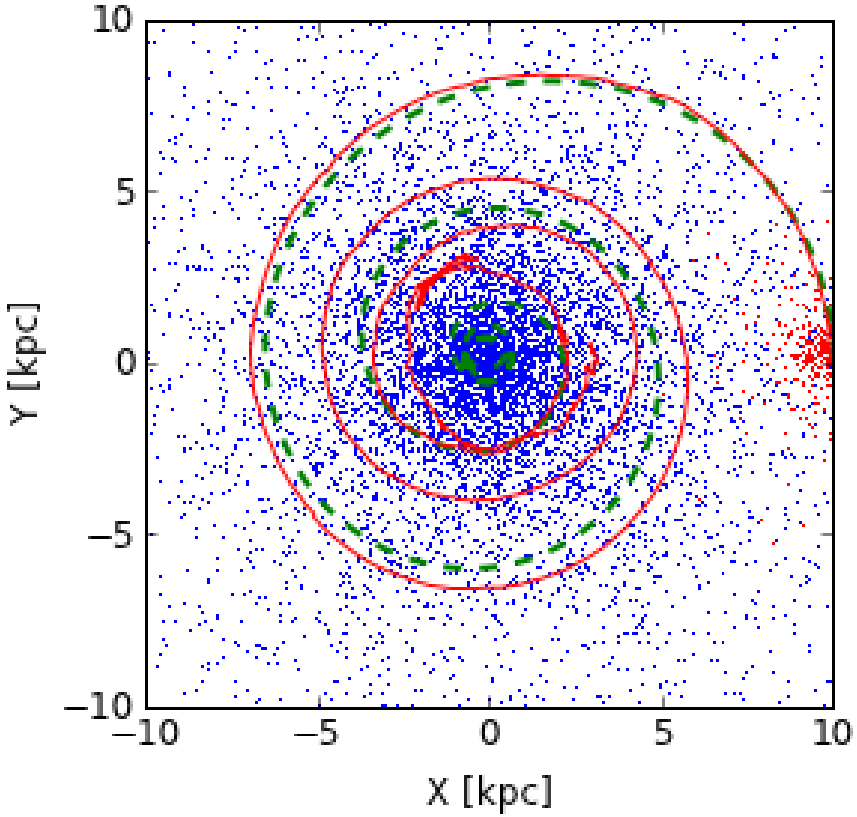
```
from amuse.lab import *
converter_gal = nbody_system.nbody_to_si( \
    1e11 | units.MSun, 10. | units.kpc)
particles_gal = new_halogen_model(10000, \
    converter_gal, alpha=1, beta=6, gamma=1)
gravity_gal = BHTree(converter_gal)
gravity_gal.parameters.epsilon_squared = \
    (0.5 | units.kpc)**2
gravity_gal.parameters.timestep = 0.5 | units.myr
gravity_gal.particles.add_particles(particles_gal)
converter_sat = nbody_system.nbody_to_si( \
    5e9  | units.MSun, 2.  | units.kpc)
particles_sat = new_king_model(500, \
    6.0, converter_sat)  # IC for the satellite
particles_sat.x  += 10. | units.kpc
particles_sat.vy += 160 | units.kms
gravity_sat = ph4(converter_sat)
gravity_sat.particles.add_particles(particles_sat)
```

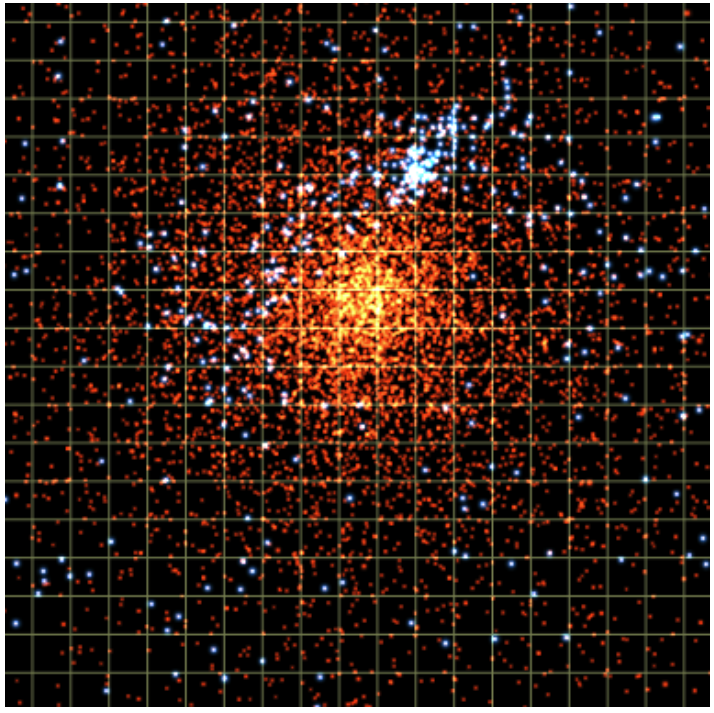
Bridge simulation: defining the bridge

```
from amuse.couple import bridge
gravity_all = bridge.Bridge(timestep=0.5|units.myr)
# define interactions between subsystems
gravity_all.add_system(gravity_gal, (gravity_sat,))
gravity_all.add_system(gravity_sat, (gravity_gal,))
for i in range(1000):
  gravity_all.evolve_model(gravity_all.timestep*i)
  # compute the center-of-mass of the galaxy...
  cm_gal = gravity_gal.particles.center_of_mass()
  # ... and the (bound part) of the satellite
  cm_sat = gravity_sat.particles.bound_subset( \
    unit_converter=converter_sat).center_of_mass()
  print "Time=", gravity_all.model_time, \
    "Offset=", cm_sat-cm_gal
```

Tidal disruption: assignment

5. Complete the script in a similar way to the previous one; compare the efficiency of the orbit decay rate between the cases of a point mass and a satellite with internal structure. Estimate the tidal disruption radius (distance from the galactic center at which the tidal force is strong enough to unbind the satellite; assume that it is a uniform-density sphere with radius $R = 1$ kpc and mass $M = 5 \times 10^9\, M_\odot$). Keyword: Hill or Roche radius.





Numerical simulations of gravitational dynamics

Eugene Vasiliev

Oxford University &
Lebedev Physical Institute

XII School of modern astrophysics 2016

Seminar 5.

Concluding remarks

Test (30 min)

1. Estimate the relaxation time for the nuclear star cluster at the center of the Milky Way, assuming $M = 4 \times 10^7\, M_\odot$, $R = 4$ pc. What method would you use to simulate its evolution?

2. You want to study the evolution of spiral structure in a disk galaxy (mass $M = 10^{11}\, M_\odot$, radius $R \simeq 5$ kpc, thickness $h \simeq 0.5$ kpc, time $T = 10^{10}$ yr). Which kind of method would you use? Motivate your choice of code and its parameters.

3. What are the reasons for using gravitational softening in N-body simulations, and when are they applicable?

4. Sagittarius dwarf galaxy is being tidally stripped by the Milky Way, producing a prominent tidal stream across the sky. Estimate the radius at which it will be entirely disrupted, assuming that the potential of the Milky Way halo has a form $\Phi(r) = \frac{1}{2} v_c^2 \ln(r)$, with $v_c = 200$ km/s, and that the density profile of the dwarf galaxy is given by the Plummer model $\rho(r) = \frac{3}{4\pi} \frac{M}{r_0^3} \left(1 + \frac{r^2}{r_0^2}\right)^{-5/2}$, with $M = 10^9\, M_\odot$ and $r_0 = 1$ kpc.