Package: nbody (via r-universe)

August 21, 2024

Type Package

Title Gravitational N-Body Simulation

Version 1.41

Description Run simple direct gravitational N-body simulations. The package can access different external N-body simulators (e.g. GADGET-4 by Springel et al. (2021) <doi:10.48550/arXiv.2010.03567>), but also has a simple built-in simulator. This default simulator uses a variable block time step and lets the user choose between a range of integrators, including 4th and 6th order integrators for high-accuracy simulations. Basic top-hat smoothing is available as an option. The code also allows the definition of background particles that are fixed or in uniform motion, not subject to acceleration by other particles.

Depends R (>= 4.0.0)

Imports magicaxis, Rcpp (>= 1.0.0)

LinkingTo Rcpp

License GPL-3

Encoding UTF-8

RoxygenNote 7.3.2

NeedsCompilation yes

Author Danail Obreschkow [aut, cre] (<https://orcid.org/0000-0002-1527-0762>)

Maintainer Danail Obreschkow <danail.obreschkow@gmail.com>

Date/Publication 2024-08-20 14:50:05 UTC

Repository https://obreschkow.r-universe.dev

RemoteUrl https://github.com/cran/nbody

RemoteRef HEAD

RemoteSha 461f1023504af40a08b1432fb59481f17ef69bf7

Contents

`nbody-package` *Gravitational N-Body Simulation*

Description

Run simple direct gravitational N-body simulations. The package can access different external N-body simulators (e.g. GADGET-4 by Springel et al., 2021), but also has a simple built-in simulator. This default simulator uses a variable block time step and lets the user choose between a range of integrators, including 4th and 6th order integrators for high-accuracy simulations. Basic top-hat smoothing is available as an option. The code also allows the definition of background particles that are fixed or in uniform motion, not subject to acceleration by other particles.

Author(s)

Danail Obreschkow <danail.obreschkow@gmail.com>

`.nbody.env` *Package environment*

Description

Environment used to store paths and options for external code.

Usage

```
.nbody.env
```

Format

An object of class `environment` of length 0.

Author(s)

Danail Obreschkow

See Also

default.code

---

default.code *Set a default external simulation code*

Description

Set a default external simulation code

Usage

```
default.code(code = NULL)
```

Arguments

code	structured list specifying the default external simulation code used when calling run.simulation. This list has exactly the same format as the sub-list 'code' described in the documentation of run.simulation.

Value

Returns the current list 'code'. If no such last has been set and 'default.code()' is called without argument, an error is produced.

Author(s)

Danail Obreschkow

---

energy *Mechanical energy of an N-body system*

Description

Computes the instantaneous potential and kinetic energies of all particles in an N-body system. Here, the potential energy of a particle i means the potential energy it has with all other particles (sum_j -G*m[i]*[j]/rij). Hence the total potential energy of the system is half the sum of the individual potential energies.

Usage

```
energy(m, x, v, rsmooth = 0, G = 6.67408e-11, cpp = TRUE)
```

Arguments

m	N-vector with the masses of the N particles. Negative masses are treated as positive masses of same magnitude, since negative masses normally represent positive background masses in the nbody package.
x	N-by-3 matrix specifying the initial position in Cartesian coordinates
v	N-by-3 matrix specifying the initial velocities
rsmooth	top-hat smoothing radius.
G	gravitational constant. The default is the measured value in SI units.
cpp	logical flag. If TRUE (default), the computation is performed efficiently in C++.

Value

Returns a list with vector items `Ekin`, `Epot`, `Emec=Ekin+Epot`; and the associated total quantities `Ekin.tot`, `Epot.tot`, `Emec=Ekin+Epot.tot`.

Author(s)

Danail Obreschkow

---

plot.simulation *Visualize an N-body simulation*

Description

Basic routine to visualise the result of an N-body simulation, projected onto a plane.

Usage

```
## S3 method for class 'simulation'
plot(
  x,
  y,
  units = 1,
  index1 = 1,
  index2 = 2,
  xlim = NULL,
  ylim = NULL,
  center = c(0, 0, 0),
  cex = 0.3,
  pch = 20,
  title = "",
  asp = 1,
  pty = "m",
  col = "black",
  alpha.orbits = 1,
```

```
  alpha.snapshots = 1,
  lwd = 1,
  show.orbits = TRUE,
  show.snapshots = TRUE,
  show.ics = TRUE,
  show.fcs = TRUE,
  ...
)
```

Arguments

x	is a simulation-object as produced by run.simulation
y	deprecated argument included for consistency with generic plot function
units	length unit in SI units
index1	index of the dimension plotted on the x-axis
index2	index of the dimension plotted on the y-axis
xlim	2-vector specifying the plotting range along the x-axis
ylim	2-vector specifying the plotting range along the y-axis
center	3-vector specifying the plotting center in the specified units
cex	point size
pch	point type
title	title of plot
asp	aspect ratio of x and y axes
pty	character specifying the type of plot region to be used; "s" generates a square plotting region and "m" generates the maximal plotting region.
col	either (1) a single color, (2) a n-element vector of colors for each particle or (3) a function(n,...) producing n colors, e.g. 'rainbow'
alpha.orbits	opacity (0...1) of orbital lines.
alpha.snapshots	opacity (0...1) of snapshot points.
lwd	line width of orbital lines.
show.orbits	logical flag. If TRUE (default), the orbits are shown as straight lines between snapshots.
show.snapshots	logical flag. If TRUE (default), points are shown for each snapshot.
show.ics	logical flag. If TRUE (default), the initial positions are highlighted.
show.fcs	logical flag. If TRUE (default), the final positions are highlighted.
...	additional parameters for plot

Value

None

Author(s)

Danail Obreschkow

`reset.cm` *Move center of mass to the origin*

Description

Routine, designed to reset the center of mass (CM) of the initial conditions (ICs) of an N-body simulation. The CM position and velocity are both shifted to (0,0,0).

Usage

```
reset.cm(sim)
```

Arguments

`sim`	list of m, x, v or list with a sublist "ics", made of m, x, v, where `m` = N-vector with the masses of the N particles. Negative masses are treated as positive masses of same magnitude, since negative masses normally represent positive background masses in the nbody package. `x` = N-by-3 matrix specifying the initial position in cartesian coordinates `v` = N-by-3 matrix specifying the initial velocities

Value

Returns a structure of the same format as the input argument, but with re-centered positions and velocities.

Author(s)

Danail Obreschkow

`run.simulation` *Run a direct N-body simulation*

Description

Run direct N-body simulations using an adaptive block timestep.

Usage

```
run.simulation(sim, measure.time = TRUE, verbose = TRUE)
```

Arguments

`sim` structured list of simulation settings, which must contain the following sublists:

`ics` is the sublist of initial conditions. It must contain the items:
`m` = N-vector with the masses of the N particles. Negative mass values are considered as positive masses belonging to a background field, which is not subject to any forces. Therefore particles with negative mass will have a normal effect on particles with positive masses, but they will not, themselves, be accelerated by any other particle.
`x` = N-by-3 matrix specifying the initial position in Cartesian coordinates
`v` = N-by-3 matrix specifying the initial velocities

`para` is an optional sublist of optional simulation parameters. It contains the items:
`t.max` = final simulation time in simulation units (see details). If not given, a characteristic time is computed as `t.max = 2*pi*sqrt(R^3/GM)`, where `R` is the RMS radius and `M` is the total mass.
`dt.max` = maximum time step. If not given, no maximum time step is imposed, meaning that the maximum time step is either equal to `dt.out` or the adaptive time step, whichever is smaller.
`dt.min` = minimum time step used, unless a smaller time step is required to save an output or to land precisely on the final time `t.max`. `dt.out` = output time step, i.e. time step between successive snapshots in the `output` sublist returned by `run.simulation`. If not given, `dt.max=t.max/100` is assumed.
`eta` = accuracy parameter of adaptive time step. Smaller values lead to proportionally smaller adaptive time steps. Typical values range between 0.001 and 0.1. If not given, a default value of 0.01 is assumed. To use fixed time steps, set `eta=1e99` and set a time step `dt.max`.
`integrator` = character string specifying the integrator to be used. Currently implemented integrators are 'euler' (1st order), 'leapfrog' (2nd order), 'yoshida' (4th order), 'yoshida6' (6th order). If not given, 'leapfrog' is the default integrator.
`rsmooth` = optional smoothing radius. If not given, no smoothing is assumed.
`afield` = a function(x,t) of positions `x` (N-by-3 matrix) and time `t` (scalar), specifying the external acceleration field. It must return an N-by-3 matrix. If not given, no external field is assumed. If the external code "nbodyx" is used, then afield should be a vector of the parameters p1, p2, ... for the external acceleration field of "nbodyx".
`G` = gravitational constant in simulation units (see details). If not given, the measured value in SI units is used.
`box.size` = scalar>=0. If 0, open boundary conditions are adopted. If >0, the simulation is run in a cubic box of side length box.size with periodic boundary conditions. In this case, the cubic box is contained in the interval [0,box.size) in all three Cartesian coordinates, and all initial positions must be contained in this interval. For periodic boundary conditions, the force between any two particles is always calculated along their shortest separation, which may cross 0-3 boundaries. The exception is GADGET-4, which also evaluates the forces from

the periodic repetitions.
include.bg = logical argument. If FALSE (default), only foreground particles, i.e. particles with masses >=0, are contained in the output vectors x and v. If TRUE, all particles are included.

code is an optional sublist to force the use of an external simulation code (see details). It contains the items:
name = character string specifying the name of the code, currently available options are "R" (default), "nbodyx" (a simple, but fast N-body simulator in Fortran) and "gadget4" (a powerful N-body+SPH simulator, not very adequate for small direct N-body simulations).
file = character string specifying the path+filename of the external compiled simulation code.
interface = optional character string specifying a temporary working path used as interface with external codes. NOTE: All existing files in this directory are deleted! If not given, the current working directory is used by default.
kind = optional number of bytes per floating-point number used in nbodyx output files (has no bearing on computation accuracy)
gadget.np = number of processors used with GADGET-4 (defaults to 1, which is normally best for small direct N-body runs)

measure.time — logical flag that determines whether time computation time will be measured and displayed.

verbose — logical flag indicating whether to show console outputs from external codes. Ignored when using the in-built simulator.

Details

UNITS: The initial conditions (in the sublist ics) can be provided in any units. The units of mass, length and velocity then fix the other units. For instance, [unit of time in seconds] = [unit of length in meters] / [unit of velocity in m/s]. E.g., if initial positions are given in units of 1AU=1.49598e11m and velocities in units of 1km/s, one unit of time is 1.49598e8s=4.74yrs. Likewise, units of the gravitational constant G are given via [unit of G in m^3*kg^(-1)*s^(-2)] = [unit of length in meters] * [unit of velocity in m/s]^2 / [unit of mass in kg]. E.g., for length units of 1AU=1.49598e11m, velocity units of 1km/s=1e3m/s and mass units of 1Msun=1.98847e30kg, a unit of G is 7.523272e-14 m^3*kg^(-1)*s^(-2). In these units the true value of G is about 887.154.

NBODYX simulator:
Can be downloaded from github via
git clone https://github.com/obreschkow/nbodyx
Details on installing, compiling and running the code are given in the README file.
Note: To run very high-accuracy simulations, such as the Pythagorean three-body problem, you can use 128-bit floating-point numbers by compiling the code as
make kind=16

GADGET-4 simulator:
This his a very powerful N-body+SPH simulator used primarily for large astrophysical simulations. GADGET-4 is not particularly suitable for small direct N-body problems, but it can nonetheless be used for such simulations for the sake of comparison, at least if not too much accuracy is needed and if a massively increased computational overhead is acceptable. Please refer to https://wwwmpa.mpa-garching.mpg.de/gadget4 for details on how to download and compile the code. In order to use GADGET-4 with this R-package, it must be compiled with the following compile-time options (in the file Config.sh):
`NTYPES=2`
`GADGET2_HEADER`
`SELFGRAVITY`
`ALLOW_DIRECT_SUMMATION`
`HIERARCHICAL_GRAVITY`
`DOUBLEPRECISION=1`
`ENLARGE_DYNAMIC_RANGE_IN_TIME`
If and only if periodic boundary conditions are used, you also need to add the option
`PERIODIC`
If you plan to often switch between runs with open and periodic boundaries, it may be advisable to compile two versions of GADGET-4, with and without this option. To do so, one needs to create two sub-directories with the respective Config.sh files and compile them via
`make -j [number of cores] DIR=[path containing Config.sh with PERIODIC]`
`make -j [number of cores] DIR=[path containing Config.sh without PERIODIC]`
The runtime parameter file (param.txt) needed by GADGET-4 is written automatically when calling `run.simulation`. The gravitational softening length in GADGET-4 is computed as sim$para$rsmooth/2.8, which ensures that the particles behave like point masses at separations beyond sim$para$rsmooth. If rsmooth is not provided, it is computed as `stats::sd(apply(sim$ics$x,2,sd))*1e-5`. The accuracy parameter ErrTolIntAccuracy is set equal to sim$para$eta/sim$para$rsmooth*1e-3, which gives roughly comparable accuracy to in-built simulator for the Leapfrog integrator.

Value

The routine returns the structured list of the input argument, with one sublist `output` added. This sublist contains the items:

`t`	k-vector with the simulation times of the k snapshots.
`x`	k-by-N-by-3 array giving the 3D coordinates of the N particles in k snapshots.
`v`	k-by-N-by-3 array giving the 3D velocities of the N particles in k snapshots.
`n.snapshots`	total number of snapshots.
`n.iterations`	total number of iterations used to run the simulation.

Author(s)

Danail Obreschkow

Examples

```
sim = setup.halley()
sim = run.simulation(sim)
AU = 149597870700 # Astronomical unit in meters
```

```
plot(sim, units=AU, xlim=c(-20,60), ylim=c(-40,40), xlab='[AU]', ylab='[AU]')
cat(sprintf('This simulation was run with %d iterations.\n',sim$output$n.iterations))
```

`setup` *Initialize N-body simulation*

Description

Routines to generate the structured lists of initial conditions and simulation parameters required to run an N-body simulation with `run.simulation`.

Usage

```
setup()

setup.halley(
  t.max = NULL,
  nperiods = 1,
  dt.out = 1e+07,
  e = 0.96714,
  s = 2.667928e+12,
  ...
)

setup.sunearth(t.max = 31557600, dt.out = 86400 * 7, ...)

setup.ellipse(t.max = NULL, nperiods = 1, e = 0.9, s = 1, f = 0.5, ...)

setup.periodic.3body(
  v1 = 0.347112813567242,
  v2 = 0.532726851767674,
  t.max = 6.325,
  m3 = 1,
  ...
)

setup.pythagoras(t.max = 68, integrator = "yoshida6", eta = 0.002, ...)
```

Arguments

`t.max`	final simulation time
`nperiods`	number of orbital periods to be computed; ignored if `t.max` is specified.
`dt.out`	time step for simulation output
`e`	eccentricity in setup.ellipse()
`s`	semi-major axis in setup.ellipse()

...	other simulation parameters used by run.simulation
f	mass-ratio in setup.ellipse()
v1	first velocity parameter in setup.3body.periodic()
v2	second velocity parameter in setup.3body.periodic()
m3	mass of third body in setup.3body.periodic()
integrator	integrator used for N-body simulation, see run.simulation for details.
eta	accuracy parameter of adaptive time step, see run.simulation for details.

Value

Calling setup() is identical to calling setup.halley()

setup.halley() sets up a 2-body simulation of Halley's Comet around the Sun.

setup.sunearth() sets up a simple 2-body simulation of the Earth around the Sun, using only approximate orbital specifications.

setup.ellipse() sets up an elliptical Keplerian orbit in natural units

setup.periodic.3body() can be used to set up a planar zero angular momentum stable 3-body problem with two unit masses and a third mass m3 (maybe equal of different from unity). Such situations can be parameterized with two parameters v1 and v2, following the publications found at https://arxiv.org/abs/1709.04775 and https://arxiv.org/abs/1705.00527.
The default is the famous figure-of-eight, but try, for example, setup.3body.periodic(0.2034916865234370, 0.5181128588867190, 32.850, dt.out=0.02), setup.3body.periodic(0.2009656237, 0.2431076328, 19.0134164290, 0.5, dt.out=0.01) or setup.3body.periodic(0.991198122, 0.711947212, 17.650780784, 4, eta=0.005, dt.out=0.002).

setup.pythagoras() sets up the Pythagorean three-body problem consisting of three unit masses placed at the vertices of a right triangle with side lengths 3, 4 and 5. The masses are initially at rest and the gravitational constant is unity.

Author(s)

Danail Obreschkow

Examples

```
sim = setup.halley()
sim = run.simulation(sim)
plot(sim)
```

Index