The Art of Computational Science

The Kali Code

vol. 6b

User Interface:

Command Line Arguments II

Piet Hut and Jun Makino

September 13, 2007

Contents

Preface

In the previous volume, a general framework has been sketched for a command line options parser, and a specific Ruby implementation has been provided. In the current volume, that implementation is used to provide a user interface for N-body experiments. Driven by user demand, a number of extensions are added to the basic program, to make it even more useful as a practical interface between the user and an N-body code.

0.1 Acknowledgments

We thank xxx, xxx, and xxx for their comments on the manuscript.

Piet Hut and Jun Makino

Chapter 1

The New Nbody Driver

1.1 Code Listing

Alice: With our new command line options parser in hand, it is time we see it at work for our N-body code.

Bob: That will be easy. Remember that I have already replaced my first attempt at parsing the command line, in `rkn1.rb`, by a new driver version, in `rkn2.rb`.

Alice: Yes, you showed me that it contained mainly the 'here document', which we specified before you even started writing the `Clop` class. Apart from them, it contained only a few lines of code, just enough to create an N-body system, read it in, and start the integrator.

Bob: Indeed. Let me list it again:

```
require "../command_line/clop.rb"

# This does not work, since there is a new "clop.rb" in this same directory
# that will be loaded anyway
#
# $acsdate="20050303"
# require "acsrequire"
# acsrequire "clop"

require "rknbody.rb"

options_definition_string = <<-END

  Description:          The simplest ACS N-body code
```

```
Long description:
  This is the simplest N-body code provided in the ACS environment
  (c) 2004, Piet Hut and Jun Makino; see ACS at www.artcompsi.org
  It offers a choice of integrators, for constant shared time steps.


Short name:           -m
Long name:            --integration_method
Value type:           string
Default value:        rk4
Global variable:      method
Print name:
Description:          Integration method
Long description:
  There are a variety of integration methods available, including:

    Forward Euler:            forward
    Leapfrog:                 leapfrog
    2nd-order Runge Kutta:    rk2
    4th-order Runge Kutta:    rk4


Short name:           -d
Long name:            --step_size
Value type:           float
Default value:        0.001
Global variable:      dt
Description:          Integration time step
Long description:
  In this code, the integration time step is held constant,
  and shared among all particles in the N-body system.


Short name:           -e
Long name:            --diagnostics_interval
Value type:           float
Default value:        1
Global variable:      dt_dia
Description:          Diagnostics output interval
Long description:
  The time interval between successive diagnostics output.
  The diagnostics include the kinetic and potential energy,
  and the absolute and relative drift of total energy, since
  the beginning of the integration.
      These diagnostics appear on the standard error stream.
  For more diagnostics, try option "-x" or "--extra_diagnostics".
```

```
Short name:           -o
Long name:            --output_interval
Value type:           float
Default value:        1
Global variable:      dt_out
Description:          Snapshot output interval
Long description:
  The time interval between output of a complete snapshot
  A snapshot of an N-body system contains the values of the
  mass, position, and velocity for each of the N particles.

      This information appears on the standard output stream,
  currently in the following simple format (only numbers):

    N:            number of particles
    time:         time
    mass:         mass of particle
    position:     x y z : vector components of position of particle
    velocity:     vx vy vz : vector components of velocity of particle
    mass:         mass of particle
    ...:          ...

  Example:

     2
     0
     0.5
    7.3406783488452532e-02  2.1167291484119417e+00 -1.4097856092768946e+00
    3.1815484836541341e-02  2.7360312082526089e-01  2.4960049959942499e-02
     0.5
   -7.3406783488452421e-02 -2.1167291484119413e+00  1.4097856092768946e+00
   -3.1815484836541369e-02 -2.7360312082526095e-01 -2.4960049959942499e-02


Short name:           -t
Long name:            --total_duration
Value type:           float
Default value:        10
Global variable:      dt_end
Description:          Duration of the integration
Long description:
  This option allows specification of the time interval, after which
  integration will be halted.
```

```
  Short name:           -s
  Long name:            --softening_length
  Value type:           float
  Default value:        0
  Global variable:      eps
  Description:          Softening length
  Long description:
    This option sets the softening length used to calculate the force
    between two particles.  The calculation scheme comforms to standard
    Plummer softening, where rs2=r**2+eps**2 is used in place of r**2.


  Short name:           -i
  Long name:            --init_out
  Value type:           bool
  Global variable:      init_out
  Description:          Output the initial snapshot
  Long description:
    If this flag is set to true, the initial snapshot will be output
    on the standard output channel, before integration is started.


  Short name:           -x
  Long name:            --extra_diagnostics
  Value type:           bool
  Global variable:      x_flag
  Description:          Extra diagnostics
  Long description:
    The following extra diagnostics will be printed:

      acceleration (for all integrators)
      jerk (for the Hermite integrator)


  END

parse_command_line(options_definition_string)

include Math

nb = Nbody.new
nb.simple_read
nb.evolve($method, $eps, $dt, $dt_dia, $dt_out, $dt_end, $init_out, $x_flag)
```

1.2 Testing

Alice: Let's try to get some simple help:

```
|gravity> ruby rkn2.rb -h
The simplest ACS N-body code
-m  --integration_method: Integration method [default: rk4]
-d  --step_size: Integration time step [default: 0.001]
-e  --diagnostics_interval: Diagnostics output interval [default: 1]
-o  --output_interval: Snapshot output interval [default: 1]
-t  --total_duration: Duration of the integration [default: 10]
-s  --softening_length: Softening length [default: 0]
-i  --init_out: Output the initial snapshot
-x  --extra_diagnostics: Extra diagnostics
```

Bob: As it should be. Why don't you try some of the long options?

Alice: I'll ask it about step size, since we've talked about that so much, and, let's see, how about a boolean variable? That one should not show any default values:

```
|gravity> ruby rkn2.rb --help --step_size -x
-d  --step_size: Integration time step [default: 0.001]

    In this code, the integration time step is held constant,
    and shared among all particles in the N-body system.

-x  --extra_diagnostics: Extra diagnostics

    The following extra diagnostics will be printed:

      acceleration (for all integrators)
      jerk (for the Hermite integrator)
```

Bob: Indeed, only the time step size gives a default value. Let's try some wrong options, to see whether we get helpful error messages:

```
|gravity> ruby rkn2.rb -q
../command_line/clop.rb:143:in `parse_command_line_options':  (RuntimeError)
  option "-q" not recognized; try "-h" or "--help"
from ../command_line/clop.rb:115:in `initialize'
from ../command_line/clop.rb:267:in `new'
from ../command_line/clop.rb:267:in `parse_command_line'
from rkn2.rb:144
```

Alice: That's pretty clear. How about leaving out a value:

```
|gravity> ruby rkn2.rb -d
../command_line/clop.rb:174:in `parse_option':  (RuntimeError)
  option "-d" requires a value, but no value given;
  option description: Integration time step
from ../command_line/clop.rb:141:in `parse_command_line_options'
from ../command_line/clop.rb:115:in `initialize'
from ../command_line/clop.rb:267:in `new'
from ../command_line/clop.rb:267:in `parse_command_line'
from rkn2.rb:144
```

Bob: Clear too. It seems that we have a working product!

1.3 Checking

Alice: Let's make a final check to see whether we get we still get our expected output for the figure three orbit. The first implementation you made gave:

```
|gravity> ruby rkn1.rb -o 2.1088 -e 2.1088 -t 2.1088 < figure8.in
eps = 0
dt = 0.001
dt_dia = 2.1088
dt_out = 2.1088
dt_end = 2.1088
init_out = false
x_flag = false
method = rk4
at time t = 0, after 0 steps :
  E_kin = 1.21 , E_pot =  -2.5 , E_tot = -1.29
            E_tot - E_init = 0
  (E_tot - E_init) / E_init = -0
at time t = 2.109, after 2109 steps :
  E_kin = 1.21 , E_pot =  -2.5 , E_tot = -1.29
            E_tot - E_init = -2e-15
  (E_tot - E_init) / E_init = 1.55e-15
3
  2.1089999999998787e+00
  1.0000000000000000e+00
 -1.6047303546488470e-04 -1.9320664965417420e-04
 -9.3227640249930266e-01 -8.6473492670753516e-01
```

```
  1.0000000000000000e+00
  9.7020367429337440e-01 -2.4296620300772800e-01
  4.6595057278750124e-01  4.3244644507801255e-01
  1.0000000000000000e+00
 -9.7004320125790211e-01  2.4315940965738195e-01
  4.6632582971180025e-01  4.3228848162952316e-01
```

Let us now check the newer version, with our implementation using the `clop.rb` code:

```
|gravity> ruby rkn2.rb -o 2.1088 -e 2.1088 -t 2.1088 < figure8.in
The simplest ACS N-body code
Integration method: rk4
Integration time step: dt = 0.001
Diagnostics output interval: dt_dia = 2.1088
Snapshot output interval: dt_out = 2.1088
Duration of the integration: dt_end = 2.1088
Softening length: eps = 0.0
at time t = 0, after 0 steps :
  E_kin = 1.21 , E_pot =  -2.5 , E_tot = -1.29
             E_tot - E_init = 0
  (E_tot - E_init) / E_init = -0
at time t = 2.109, after 2109 steps :
  E_kin = 1.21 , E_pot =  -2.5 , E_tot = -1.29
             E_tot - E_init = -2e-15
  (E_tot - E_init) / E_init = 1.55e-15
3
  2.1089999999998787e+00
  1.0000000000000000e+00
 -1.6047303546488470e-04 -1.9320664965417420e-04
 -9.3227640249930266e-01 -8.6473492670753516e-01
  1.0000000000000000e+00
  9.7020367429337440e-01 -2.4296620300772800e-01
  4.6595057278750124e-01  4.3244644507801255e-01
  1.0000000000000000e+00
 -9.7004320125790211e-01  2.4315940965738195e-01
  4.6632582971180025e-01  4.3228848162952316e-01
```

Good, all is well.

Chapter 2

XXX

2.1 xxx

first introduce the series `rkn1.rb` through `rkn4.rb`, the first two of which were already introduced in the previous volume, and all of which still depend on `clop.rb` as introduced in the previous volume.

Note that the change from `Nbody` to `NBody` is described in the text that is temporarily parked in the next chapter

Then start improving the `clop.rb` function.

Finally move from `rkn4.rb` to `nbody_cst0.rb`.

Note that `nbody_cst0.rb` still uses global variables; and so does `nbody_cst1.rb` in later volumes. This is finally changed when `nbody_sh1.rb` appears.

Chapter 3

XXX

3.1 xxx

3.2 No Retrofitting

Bob: Oh, by the way, I decided to make a slight change in notation. So far, we have called our N-body class `Nbody`. However, it seems that in the Ruby community people have developed a clear convention to name all classes using the socalled MixedCase notation: each word is capitalized. Since in our case *N* and *body* are really different words, it would be much more consisten to call our N-body class `NBody`.

Alice: That's a really good idea. It is important to stick to the conventions that are used in a community. That will make it easier for others to read our code, and to jump right in, making their own additions. I guess this means that you will go back and change `Nbody` into `NBody` everywhere in the volumes that we've already written?

Bob: That would be a lot of work. Besides, it might confuse students and colleagues who have already downloaded some of our earlier volumes, if we were to suddenly change our notation, retroactively.

Alice: Again, you have a good point. Retrofitting what we have done, in order to make the past consistent with the present, is a dubious idea, I agree. Do you think it would be better not to do that, in general?

Bob: I think so. And since you like rules and abstractions, how about making this into a rule of ours: thou shalt not retrofit. Or in plain English: we will consider our work up till now read-only. We can add to it, but we will resist the temptation to clean it all up.

Alice: Perhaps in the future, if and when we make everything available as a

book series, we could make a cleaning-up pass through the whole collection. But until that point, yes, I agree, let's just accumulate, rather than correct.

Bob: Of course, if we find plain mistakes, we should correct them, but other than that, we'll simply freeze each volume after we finish it.

Alice: The more I think about it, the more I realize that is the *only* way to go. After all, when we have N previous volumes, the probability that we find some sort of minor inconsistency is at least proportional to N. And if you take into account the way that inconsistencies may influence each other, you get something like compound interest, which was the way that the exponential function was first discovered! In the light of that, I guess the probability to be still consistent after N volumes must be something like $1 - \exp(-cN)$ for some constant c.

Even without taking into account compound effects, of changes triggering other changes, if one new volume gives rise to one incompatibility with each previous volume, we have N incompatibility, which should then be repaired in each of the N previous volumes, giving rise to an amount of work of order N^2. For N new volumes, the work will scale as N^3. And that is using only the most optimistic assumption! Okay, I'm convinced.

Bob: Good! That will save us a lot of work.

3.3 xxx

nil nil nil nil nil nil nil

Chapter 4

Literature References

[to be provided]