Computational Physics – Week 4

Andrei Alexandru

Feb 19, 2020

1 Introduction

This week will continue to discuss numerical integration of ordinary differential equations. We will specialize the code we developed last week to the case of N bodies interacting under the influence of gravity. The force then on any one body is the vector sum of the forces exerted by all the others:

$$\boldsymbol{F}_i = \sum_{j\neq i} \boldsymbol{F}_{i\leftarrow j} = \sum_{j\neq i} G\frac{m_i m_j}{|\boldsymbol{r}_j - \boldsymbol{r}_i|^3}(\boldsymbol{r}_j - \boldsymbol{r}_i)\,, \tag{1}$$

where $\boldsymbol{r}_i$ is the position for object i. The equations of motion are then

$$\ddot{\boldsymbol{r}}_i = \frac{1}{m_i}\boldsymbol{F}_i = \underbrace{\sum_{j\neq i} G\frac{m_j}{|\boldsymbol{r}_j - \boldsymbol{r}_i|^3}(\boldsymbol{r}_j - \boldsymbol{r}_i)}_{\boldsymbol{a}_i}\,. \tag{2}$$

To turn this in the first order equations we introduce the velocities as auxiliary variables and we have

$$\frac{d}{dt}\begin{pmatrix} \boldsymbol{r}_0 \\ \vdots \\ \boldsymbol{r}_{N-1} \\ \boldsymbol{v}_0 \\ \vdots \\ \boldsymbol{v}_{N-1} \end{pmatrix} = \begin{pmatrix} \boldsymbol{v}_0 \\ \vdots \\ \boldsymbol{v}_{N-1} \\ \boldsymbol{a}_0 \\ \vdots \\ \boldsymbol{a}_{N-1} \end{pmatrix} \tag{3}$$

To integrate the differential equation $\dot{\boldsymbol{y}} = \boldsymbol{f}(\boldsymbol{y}, t)$ numerically we will use Runge-Kutta method of order 2 (RK2):

$$\begin{aligned} \boldsymbol{k}_1 &= \boldsymbol{f}(\boldsymbol{y}_{n-1}, t_{n-1})\,, \\ \boldsymbol{k}_2 &= \boldsymbol{f}(\boldsymbol{y}_{n-1} + \boldsymbol{k}_1 \times \Delta t/2, t_{n-1} + \Delta t/2)\,, \\ \boldsymbol{y}_n &= \boldsymbol{y}_{n-1} + \boldsymbol{k}_2 \times \Delta t\,. \end{aligned} \tag{4}$$

This process involves two evaluations of the function $\boldsymbol{f}$ per step, but it generates an approximation that vanishes as $(\Delta t)^2$.

2 Implementation

We will discuss the implementation of this problem for two bodies ($N = 2$) moving in two dimensions. To completely specify this problem we will use $m_0 = m_1 = 1$ and initial conditions $\boldsymbol{r}_0(t = 0) = \{0, 0\}$, $\boldsymbol{v}_0(t = 0) = \{0, 0\}$, $\boldsymbol{r}_1(t = 0) = \{1, 0\}$ and $\boldsymbol{v}_1(t = 0) = \{0, 2\pi\}$. The masses are measure in units of solar masses, the distances in astronomical units, and time in measured in years. In this system of units $G = 4\pi^2$.

We will write a `main` routine to read in the time step Δt and the number of steps `nsteps`, integrate the equations of motion and print out the full state of the system at each time step. Below is the code that performs these steps:

```
1  int main(int argc, char** argv)
2  {
3    if(argc < 3)
4    {
5      printf("Usage: %s deltat nstep\n", argv[0]);
6      return -1;
7    }
8    double dt = atof(argv[1]);
9    int nstep = atoi(argv[2]);
10
11   double y[4*NBODY]={0,0,1,0,0,0,0,2*M_PI}, yn[4*NBODY], t=0;
12
13   double totmom[2] = {0,0};
14   double totmass = 0;
15   for(int i=0; i<NBODY; ++i)
16   {
17     totmom[0]   += mass[i]*y[2*NBODY+2*i];
18     totmom[1] += mass[i]*y[2*NBODY+2*i+1];
19     totmass += mass[i];
20   }
21   double vel[2] = {totmom[0]/totmass, totmom[1]/totmass};
22   for(int i=0; i<NBODY; ++i)
23   {
24     y[2*NBODY + 2*i]     -= vel[0];
25     y[2*NBODY + 2*i + 1] -= vel[1];
26   }
27
28   for(int i=0; i<nstep; ++i)
29   {
30     integrate_rk2(y, 4*NBODY, t, dt, yn, F);
31     printf("% .5f ", t+dt);
32     for(int k=0; k<4*NBODY; ++k) printf("%20.15e ", yn[k]);
33     printf("\n");
34     for(int k=0; k<4*NBODY; ++k) y[k] = yn[k];
35     t = t + dt;
36   }
37
38   return 0;
39 }
```

Lines 1–9 take care of reading the input from the command line, line 11 sets the initial conditions, lines 13–26 make sure that the center of mass of the system of N bodies is at rest so that the system does not drift (more about this below), and lines 28–36 execute the main loop of the program integrating the equations of motion and printing the results at every step.

Note that with most generic initial conditions the total momentum of the system $\boldsymbol{P}$ is non-zero, so that the center of mass of the system is moving at constant speed $\boldsymbol{v} = \boldsymbol{P}/M$, where M is the total mass of the system. If we plot the trajectory of the system we will see that superimposed on the orbital motion a trivial drift appears due to the motion of the center of mass. To remove this trivial motion we need to make sure that the initial condition is such that the center of mass is at rest at $t = 0$. This can be accomplished by subtracting the velocity of the center of mass from each particle: $\boldsymbol{v}_i \to \boldsymbol{v}_i - \boldsymbol{v}$. This is done in the lines 13–26 above: lines 13–20 compute the total momentum $\boldsymbol{P} = \sum_i m_i \boldsymbol{v}_i$ and the total mass $M = \sum_i m_i$; on line 21 the velocity $\boldsymbol{v} = \boldsymbol{P}/M$ is computed, and on lines 22–26 this velocity is subtracted from each velocity.

To integrate this we generalize the implementation of the `integrate_rk2` routine and include it in a separate file. The updated routine will be included in a file which we will call `integrator.c`, listed below

```c
#include <stdlib.h>


void integrate_rk2(double* y, int n, double t, double dt, double* yn,
    void (*f)(double*, double, double*))
{
  double *k1 = malloc(n*sizeof(double));
  double *k2 = malloc(n*sizeof(double));
  f(y, t, k1);
  double *yhalf = malloc(n*sizeof(double));
  for(int k=0; k<n; ++k) yhalf[k] = y[k] + 0.5*dt*k1[k];
  f(yhalf, t+0.5*dt, k2);
  for(int k=0; k<n; ++k) yn[k] = y[k] + k2[k]*dt;

  free(yhalf);
  free(k1);
  free(k2);
}
```

The main change to this routine is that we allow the state vector `y` to have arbitrary length. To implement this we need to pass also the length of the array, `n` as a parameter to the `integrate_rk2` routine. Finally, the problem to integrate is specified by the derivatives vector $\boldsymbol{f}$, which for this routine is passed as a parameter `f`. This is a special type of variable, a pointer to a function, and we need to specify the type of this function. Above we declare that `f` is a function that does not return anything (`void`), and takes an array of type `double*`, meant to represent the vector $\boldsymbol{y}$, a `double` that will be the time t, and an array `double*` to store the derivatives $\boldsymbol{f}(\boldsymbol{y}, t)$. Finally, we need memory to store the intermediate results of our calculations, the arrays `k1`, `k2`, and `yhalf`. Since the size of these arrays is not known at compile time we have to allocate dynamically memory on the *heap* using the standard library routine `malloc`. Since this is a finite resource, we have to remember to release this memory when we don't need it anymore using the `free` routine. This file can be integrated separately using `gcc -c integrator.c` which should produce an *object* file `integrator.o` that will be linked with the main file to produce the executable. To make it easy to use this routine we should create a *header* file, which we will call `integrator.h`, that includes the declaration of the `integrate_rk2` function. The contents of this file is listed below:

```c
void integrate_rk2(double* y, int n, double t, double dt, double* yn,
    void (*f)(double*, double, double*));
```

The final ingredient of this program is the implementation of the derivative function given in Eq. 3. The implementation requires to map the variables to the entries of the state array `y`. Given the chosen layout the coordinate x_i and y_i are stored at `y[2*i]` and `y[2*i+1]`, and the velocity components $(\boldsymbol{v}_i)_x$ and $(\boldsymbol{v}_i)_y$ are stored at `y[2*NBODY+2*i]` and `y[2*NBODY+2*i+1]`. The implementation is listed below

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "integrator.h"

#define NBODY 2
double mass[NBODY]={1,1};
double G = 4*M_PI*M_PI;

void F(double* y, double t, double* res)
{

  for(int i=0; i<2*NBODY; ++i) res[i] = y[2*NBODY+i];

```

```
15    for(int i=0; i<NBODY; ++i)
16    {
17      res[2*NBODY+2*i] = 0;    // vx_i
18      res[2*NBODY+2*i+1] = 0; // vy_i
19      for(int j=0; j<NBODY; ++j) if(j!=i)
20      {
21        double r[2] = { y[2*j] - y[2*i], y[2*j+1]-y[2*i+1] };
22        double magr = sqrt(r[0]*r[0]+r[1]*r[1]);
23        res[2*NBODY+2*i]   += G*mass[j]*r[0]/pow(magr,3);
24        res[2*NBODY+2*i+1] += G*mass[j]*r[1]/pow(magr,3);
25      }
26    }
27  }
```

Lines 1–8 do the housekeeping including the relevant header files and defining the `NBODY` macro and the `mass` array and the `G` constant. The function `F` is where the derivatives are computed: on line 13 the easy part of the equation, $\dot{\boldsymbol{r}}_i = \boldsymbol{v}_i$, is implemented. The more complex part, $\dot{\boldsymbol{v}}_i = \boldsymbol{a}_i$, is implemented in the lines 15–26. To compile the code we need to also link the object file `integrator.o` so the proper command is `gcc -o nbody nbody.c integrator.o -lm`.

3 Teaching objectives

- Numerical integration of ODEs using RK2 method.
- Breaking code in modules in separate files.
- Arrays and dynamical memory allocation.
- Function pointers used to pass functions as paramters to other functions.