`rkev`

General Purpose s-stage Runge-Kutta and Evolutionary Optimizer

Jackson L. Cole

Fall 2018

Middle Tennessee State University
`me@jacksoncole.io` • `jacksoncole.io`

What is a numerical integrator?

Let's say we are given a function describing the velocity of some object and an initial position.

$$\dot{x} = f\left(t, x\left(t\right)\right) \quad x\left(t_i\right) = x_i,$$

If we are interested in the position, we can can approximate the solution to the ODE by using numerical integration.

$$x_{i+1} = x_i + \int_{t_i}^{t_{i+1}} f(t, x(t))dt$$

The Workhorse: Fourth-order Runge-Kutta (RK4)

- Developed by Carl Runge and Wilhelm Kutta in early 1900s
- Most commonly used is the fourth-order Runge-Kutta (RK4)

$$
\begin{aligned}
k_1 &= f\left(t_i,\, x_i\right) \\
k_2 &= f\left(t_i + \frac{1}{2}h,\, x_i + \frac{1}{2}k_1\right) \\
k_3 &= f\left(t_i + \frac{1}{2}h,\, x_i + \frac{1}{2}k_2\right) \\
k_4 &= f\left(t_i + h,\, x_i + k_3\right) \\
x &= x_i + \mathbf{\frac{1}{6}}\left(k_1 + 2k_2 + 2k_3 + k_4\right)h \\
t &= t_i + h
\end{aligned}
$$

RK4 Scheme: Equations

$$
\begin{array}{llllllll}
k_1 = f(t_i & +(\mathbf{0})h, & x_i + & (\mathbf{0})k_1 & +(\mathbf{0})k_2 & +(\mathbf{0})k_3 & +(\mathbf{0})k_4 &) \\
k_2 = f(t_i & +(\mathbf{\frac{1}{2}})h, & x_i + & (\mathbf{\frac{1}{2}})k_1 & +(\mathbf{0})k_2 & +(\mathbf{0})k_3 & +(\mathbf{0})k_4 &) \\
k_3 = f(t_i & +(\mathbf{\frac{1}{2}})h, & x_i + & (\mathbf{0})k_1 & +(\mathbf{\frac{1}{2}})k_2 & +(\mathbf{0})k_3 & +(\mathbf{0})k_4 &) \\
k_4 = f(t_i & +(\mathbf{1})h, & x_i + & (\mathbf{0})k_1 & +(\mathbf{0})k_2 & +(\mathbf{1})k_3 & +(\mathbf{0})k_4 &) \\
x = & & x_i + & [(\mathbf{\frac{1}{6}})k_1 & +(\mathbf{\frac{2}{6}})k_2 & +(\mathbf{\frac{2}{6}})k_3 & +(\mathbf{\frac{1}{6}})k_4 &]h
\end{array}
$$

RK4 Scheme: Butcher Tableau

$$
\begin{array}{c|cccc}
0 & 0 & 0 & 0 & 0 \\
\frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\
\frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\
1 & 0 & 0 & 1 & 0 \\
\hline
 & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6}
\end{array}
$$

Figure 1: Full Butcher Tableau for explicit fourth-order Runge-Kutta (RK4). This form comes from an abstraction of the full RK4 scheme.

Butcher Tableaus and Runge-Kutta Geometry

$$
\begin{array}{c|cccccc}
0 & & & & & & \\
c_2 & a_{21} & & & & & \\
c_3 & a_{31} & a_{32} & & & & \\
\vdots & \vdots & \vdots & \ddots & & & \\
c_s & a_{s1} & a_{s2} & \dots & a_{s,s-1} & & \\
\hline
 & b_1 & b_2 & \dots & b_{s-1} & b_s &
\end{array}
$$

Figure 2: Butcher Tableau for explicit s-stage Runge-Kutta. This form of the tableau is found in many texts, so I will omit a reference here.

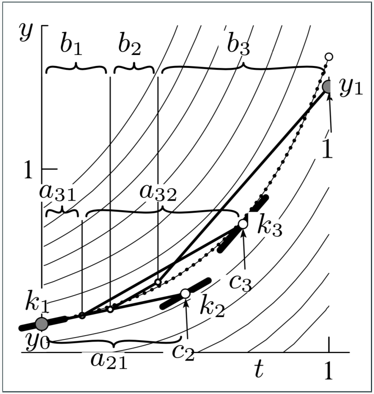


Figure 3: Geometrical diagram of the Runge-Kutta method; The explicitly defined rows of Figure 2 match with this diagram found in Hairer et al. (1).

Specific Issues with Runge-Kutta in Astronomy

In the orbits of gravitationally bound systems, the total energy of the system should be conserved.

Issue:

- Runge-Kutta methods do not preserve conserved quantities, and typically result in error that increases over long time integration.
- Essentially, RK methods integrate over a space in which a solution for the system can *only* be approximated.

Solution:

- Symplectic integrators (preserve solution structure)
- Optimization of the Runge-Kutta scheme itself

Goals and Benchmarks

Goals

- RK optimizer (i.e. Butcher Tableau optimizer)
- General-purpose RK Integrator

Benchmarks

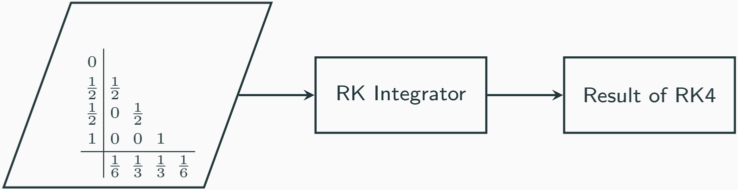
- Energy error improvement in gravitational n-body simulations

General-purpose Runge-Kutta Integrator Specifications

Requirements:

- Must be able to perform *any* RK integration scheme defined in a lower-triangular Butcher Tableau
- Must be callable in Python
- Must be able to call functions written in Python

If the appropriate initial conditions are defined, a standard fourth-order Runge-Kutta should be implemented as:



General-purpose Runge-Kutta Integrator Implementation

- Written in C++
- Wrapped for Python using the Boost.Python library
- Accepts any lower-triangular square matrix
  - i.e. this implementation can only handle **explicit** Runge-Kutta schemes
- Provides methods to run the full routine or step incrementally
- Provides methods to integrate a list of values (e.g. in an n-body simulation)
- Returns the result

`nbody` package

Born out of frustration with the overhead of n-body simulations of simple systems where randomizing masses is not feasible (i.e. specific two body systems)

- `Body` class
- `System` class

Reduces setup to initializing bodies, RKIntegrators, and calling the run method

n-body Simulation WITHOUT Energy Loss Correction

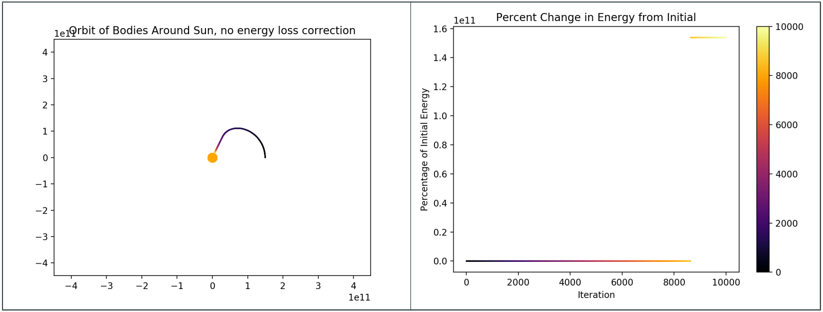


Figure 4: LEFT: Simulation of the Earth-Sun system showing heavy change in energy given the stability of the system. **RIGHT:** The change in energy of the system. The large "step" seen here could likely be improved by implementing adaptive step-size features in the RK Integrator.

Optimization with DEAP

DEAP $\rightarrow$ **D**istributed **E**volutionary **A**lgorithms in **P**ython

- Several generations of Butcher Tableaus are evaluated to minimize the change in energy of the system
- The individual Butcher Tableaus are seeded from the more successful RK4 and RK38 methods (both developed by Runge-Kutta).
- The most successful individuals are crossed with a probability of 0.7.

n-body Simulation WITH Energy Loss Correction

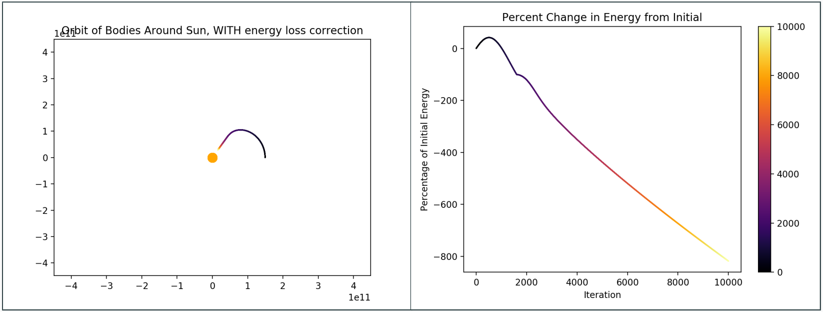


Figure 5: LEFT: Simulation of the Earth-Sun system still showing decay of the orbit, but the change can be really be seen in the energy plot. **RIGHT:** The change in energy of the system. This is significantly less of a change than the result of the prior integration.

Results, Conclusions

- This method appears to have potential as a case-by-case optimizer for ODEs that have an associated cost function.
- Even with a standard RK4 method, a non-structure-preserving n-body code has been developed that is more intuitive and user-friendly.

Improvements

- The results are, at most, promising. Although energy change is minimized, the resulting simulation still does not mirror the real system. Further work must be done to improve the n-body code *and* the RK optimizer.

References

[1] E. Hairer, Christian Lubich, and Gerhard Wanner. *Geometric numerical integration: structure-preserving algorithms for ordinary differential equations = Ji he shu zhi ji fen*. Shi jie tu shu chu ban gong si, 2017.