Supplemental Material for "Cosmology-informed neural networks to solve the background dynamics of the Universe"

Augusto T. Chantada,[1,*] Susana J. Landau,[1,2] Pavlos Protopapas,[3]
Claudia G. Scóccola,[4,5] and Cecilia Garraffo[6,7]

[1]*Departamento de Física, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires, Av. Intendente Cantilo S/N 1428 Ciudad Autónoma de Buenos Aires, Argentina*
[2]*IFIBA - CONICET - UBA, Av. Intendente Cantilo S/N 1428 Ciudad Autónoma de Buenos Aires, Argentina*
[3]*John A. Paulson School of Engineering and Applied Sciences, Harvard University, Cambridge, Massachusetts 02138, USA*
[4]*Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET), Godoy Cruz 2290, 1425, Ciudad Autónoma de Buenos Aires, Argentina*
[5]*Facultad de Ciencias Astronómicas y Geofísicas, Universidad Nacional de La Plata, Observatorio Astronómico, Paseo del Bosque, B1900FWA La Plata, Argentina*
[6]*Center for Astrophysics | Harvard & Smithsonian, 60 Garden Street, Cambridge, Massachusetts 02138, USA*
[7]*Institute for Applied Computational Science, Harvard University, 33 Oxford St., Cambridge, Massachusetts 02138, USA*

In this text, we give a description of SciPy's implementation of the explicit adaptive Runge-Kutta 5(4) [RK5(4)] algorithm with quartic interpolation for a given initial value problem. After we describe the algorithm, we estimate a formula for the amount of floating point operations (FLOPs) that it takes to compute the solutions.

I. THE ALGORITHM

A system of differential equations falling in the category of initial value problem is of the following form:

$$\frac{d\boldsymbol{u}}{dt} = f\left(t, \boldsymbol{u}\right), \ \boldsymbol{u}\left(t\right)\big|_{t=t_0} = \boldsymbol{u}_0, \tag{1}$$

where $\boldsymbol{u} = (x, y, z, \dots)$ represents the vector of the M dependent variables, the function $f : \mathbb{R}^{M+1} \to \mathbb{R}^M$ that defines the problem with the initial condition $\boldsymbol{u}_0$. SciPy's `solve_ivp` class contains the RK5(4) method, which is defined by the coefficients a_{ij}, b_i, b_i^*, c_i provided in Ref. [1]. In addition, the SciPy implementation defines P_{ij} to use the quartic interpolation proposed in Ref. [2]. We want the solutions to Eq. (1) for each of the M dependent variables in $\boldsymbol{u}$ at $\boldsymbol{t}_{\text{out}} = \left(t_{\text{out}}^{(1)}, \dots, t_{\text{out}}^{(l)}, \dots, t_{\text{out}}^{(N_{\text{out}})}\right)$ which we can represent as $\boldsymbol{U}_{\text{sol}} = \left(\boldsymbol{u}_{\text{sol}}^{(1)}, \dots, \boldsymbol{u}_{\text{sol}}^{(l)}, \dots, \boldsymbol{u}_{\text{sol}}^{(N_{\text{out}})}\right)$ with relative tolerance $\boldsymbol{r}_{\text{tol}} = (r_{\text{tol}}^x, r_{\text{tol}}^y, r_{\text{tol}}^z, \dots)$ and absolute tolerance $\boldsymbol{a}_{\text{tol}} = (a_{\text{tol}}^x, a_{\text{tol}}^y, a_{\text{tol}}^z, \dots)$ such that at any step n the following relation holds true:

$$R\left\{\boldsymbol{e}_n \oslash \left[\boldsymbol{a}_{\text{tol}} + \boldsymbol{r}_{\text{tol}} \odot \max\left(\boldsymbol{u}_{n-1}^{\text{abs}}, \boldsymbol{u}_n^{\text{abs}}\right)\right]\right\} < 1, \tag{2}$$

where $\boldsymbol{e}_n = (e_n^x, e_n^y, e_n^z, \dots)$ are the estimated errors at step n, and $\boldsymbol{u}_n^{\text{abs}} = (|x_n|, |y_n|, |z_n|, \dots)$ are the absolute values of each variable at step n. In addition, we denote the element-wise product as $\odot$, the element-wise division as $\oslash$, and the function $R : \mathbb{R}^M \to \mathbb{R}$ takes the root mean square of a given vector. In the following subsections we give a step by step description of the main operations that the algorithm takes to compute $\boldsymbol{U}_{\text{sol}}$ (assuming $t_0 < t_{\text{out}}^{(1)} < \cdots < t_{\text{out}}^{(N_{\text{out}})}$).

[*] augustochantada01@gmail.com

A. Runge-Kutta step calculation

In this Runge-Kutta method, the number of stages of each step is effectively six, although there are seven $\boldsymbol{k}$'s (an explanation of this is found below). The quantities relevant for the step calculations are as follows:

$$\boldsymbol{k}_1 = f\left(t_n, \boldsymbol{u}_n\right), \tag{3a}$$

$$\boldsymbol{k}_i = f\left(t_n + c_i h, \boldsymbol{u}_n + h\sum_{j=1}^{i-1} a_{ij}\boldsymbol{k}_j\right), \ \text{(for: } i = 2, \dots, 6) \tag{3b}$$

$$\boldsymbol{u}_{n+1} = \boldsymbol{u}_n + h\sum_{i=1}^{6} b_i\boldsymbol{k}_i, \tag{3c}$$

$$\boldsymbol{k}_7 = f\left(t_n + h, \boldsymbol{u}_{n+1}\right) = \boldsymbol{k}_1^{(n+2)}. \tag{3d}$$

When the step $n+1$ is accepted, $\boldsymbol{k}_7$ in $n+1$ is used as $\boldsymbol{k}_1$ in step $n+2$. Then, $n+2$ only has six stages, because $\boldsymbol{k}_1$ was not computed there. This is why this particular method is effectively a six stage one, the amount of function evaluations per step done (so both the accepted and rejected ones) is six.

B. Error

Once all the values of the RK step are done, the method checks if the estimated error meets the criteria set by the tolerances. This is done by calculating a $\boldsymbol{u}^*_{n+1}$ as

$$\boldsymbol{u}^*_{n+1} = \boldsymbol{u}_n + h\sum_{i=1}^{7} b^*_i \boldsymbol{k}_i. \tag{4}$$

Then, the error can estimated as

$$\boldsymbol{e}_{n+1} = \boldsymbol{u}_{n+1} - \boldsymbol{u}^*_{n+1} = h\sum_{i=1}^{7} \left(b_i - b^*_i\right) \boldsymbol{k}_i. \tag{5}$$

It is important to note that SciPy does not calculate $(b_i - b^*_i)$ each time, but rather defines $E_i = b_i - b^*_i$, which is stored in the code making it more efficient. This way, the actual calculation per step is just

$$\boldsymbol{e}_{n+1} = h\sum_{i=1}^{7} E_i \boldsymbol{k}_i. \tag{6}$$

Next, we need to calculate the argument of R in Eq. (2). For this let us define $\hat{\boldsymbol{e}}_{n+1}$ as

$$\begin{aligned}\hat{\boldsymbol{e}}_{n+1} =& \boldsymbol{a}_{\text{tol}} + \boldsymbol{r}_{\text{tol}} \odot \max\left(\boldsymbol{u}^{\text{abs}}_n, \boldsymbol{u}^{\text{abs}}_{n+1}\right) \\ =& \left[a^x_{\text{tol}} + r^x_{\text{tol}} \max\left(|x_n|, |x_{n+1}|\right), \dots\right].\end{aligned} \tag{7}$$

Then, to close out the error, the algorithm calculates the following:

$$R\left(\boldsymbol{e}_{n+1} \oslash \hat{\boldsymbol{e}}_{n+1}\right) = \sqrt{\frac{1}{M}\left[\left(\frac{e^x_{n+1}}{\hat{e}^x_{n+1}}\right)^2 + \cdots\right]}. \tag{8}$$

If the value that Eq. (8) yields is greater than one, the step is rejected. Then, the method starts the step again with a lower value for h. On the other hand, if Eq. (8) outputs a value lower than one, the step is considered accepted. In this case, the value of h is increased a small amount for the next step to optimize the speed of the algorithm.

C. Interpolation

When the step $n+1$ is accepted, if $\exists\, t^{(l)}_{\text{out}} \in \boldsymbol{t}_{\text{out}}$ such that $t_n < t^{(l)}_{\text{out}} < t_{n+1}$, then the interpolation process must be executed. To do this, first we need to calculate $\boldsymbol{Q}_j$ as

$$\boldsymbol{Q}_j = \sum_{i=1}^{7} \boldsymbol{k}_i P_{ij}. \tag{9}$$

Then, compute

$$x^{(l)} = \frac{t^{(l)}_{\text{out}} - t_n}{h}, \tag{10}$$

and subsequently calculate $y^{(l)}_j$ as

$$\left(y^{(l)}_1, y^{(l)}_2, y^{(l)}_3, y^{(l)}_4\right) = \left(x^{(l)}, \left(x^{(l)}\right)^2, \left(x^{(l)}\right)^3, \left(x^{(l)}\right)^4\right) \tag{11}$$

Then, the interpolated value can be obtained from

$$\boldsymbol{u}^{(l)}_{\text{sol}} = \boldsymbol{u}_n + h\sum_{j=1}^{4} \boldsymbol{Q}_j y^{(l)}_j. \tag{12}$$

To demystify this step, one could write an equivalent version as: $\tilde{b}_i = \sum_{j=1}^{4} P_{ij} y^{(l)}_j$, with $\boldsymbol{u}^{(l)}_{\text{sol}} = \boldsymbol{u}_n + h\sum_{i=1}^{7} \tilde{b}_i \boldsymbol{k}_i$. Once the interpolation has been done for all the elements in $\boldsymbol{t}_{\text{out}}$ that are greater than t_n and less than t_{n+1}, then the RK step $n+1$ can be considered finished, and $n+2$ starts. Once all the N_{out} points have been calculated for each of the M dependent variables, the algorithm stops.

II. FLOPS

To find out the computational cost of this specific method, one can calculate the amount of FLOPs that are required for its completion. Given that we can estimate the FLOPs of each of the three main parts of the algorithm, all that is needed is the number of times each of the parts are executed in order to obtain the desired output. For the case of the first two parts, it is the number of times a RK step was computed, which we denote as N_{step} (note that this includes both rejected and accepted steps). Then, for the interpolation part, if we assume that most steps are accepted, and that in each step there is a value of $\boldsymbol{t}_{\text{out}}$ to interpolate, the calculation of the four $\boldsymbol{Q}_j$'s is executed roughly for each RK step. The other part of the interpolation left is executed N_{out} times. With this in mind, the total amount of FLOPs, $\mathcal{F}_{\text{RK5(4)}}$, that the algorithm needs to do in order to solve a given problem can be estimated as

$$\mathcal{F}_{\text{RK5(4)}} \simeq N_{\text{out}}\mathcal{F}_{\text{int}} + N_{\text{step}}\left(\mathcal{F}_{\text{step}} + \mathcal{F}_{\text{error}} + 4\mathcal{F}_{\boldsymbol{Q}_j}\right), \tag{13}$$

where $\mathcal{F}_{\text{int}}$ is the FLOPs of the interpolation part, $\mathcal{F}_{\text{step}}$ is the amount of FLOPs of the calculation of the RK step, $\mathcal{F}_{\text{error}}$ the amount for the error calculation, and $\mathcal{F}_{\boldsymbol{Q}_j}$ the FLOPs required to calculate $\boldsymbol{Q}_j$. In the following subsections, the FLOPs of each part of the algorithm are calculated. To do that, a FLOP is counted for every addition or multiplication, for example, to calculate $d = ab + c$, 2 FLOPs are required. A convenient formula is utilized throughout the calculations for the dot product is as follows:

$$\mathcal{F}\left(\sum_{i=1}^{N} a_i b_i\right) = 2N - 1. \tag{14}$$

A. Step calculation FLOPs

Because $\boldsymbol{k}_1$ comes from the preceding step's $\boldsymbol{k}_7$, the amount of stages per step is six and the calculation of $\boldsymbol{k}_1$ is not included. Therefore, one needs to sum the FLOPs associated to each $\boldsymbol{k}_i$, where $i = 2, \ldots, 7$, and the FLOPs needed to calculate $\boldsymbol{u}_{n+1}$:

$$\mathcal{F}_{\text{step}} = \left(\sum_{i=2}^{7} \mathcal{F}_{\boldsymbol{k}_i}\right) + \mathcal{F}_{\boldsymbol{u}}. \tag{15}$$

The expression for $\mathcal{F}_{\boldsymbol{k}_i}$ is easy to formulate as a function of i for $i = 2, \ldots, 6$, using Eq. (3b). Each of these calculations are comprised of the calculation $t_n + hc_i$, which is two FLOPs, the sum $\boldsymbol{u}_n + h\sum_{j=1}^{i-1} a_{ij}\boldsymbol{k}_j$ which is $M\left[1 + 1 + (2(i-1) - 1)\right]$ FLOPs, and one function evaluation which costs $\mathcal{F}_{\text{eval}}$. Adding each term for $i = 2, \ldots, 6$:

$$\begin{aligned}\sum_{i=2}^{6} \mathcal{F}_{\boldsymbol{k}_i} &= \sum_{i=2}^{6} 2 + M\left[1 + 1 + (2(i-1) - 1)\right] + \mathcal{F}_{\text{eval}} \\ &= 5\left(\mathcal{F}_{\text{eval}} + 2\right) + M\sum_{i=2}^{6}(2i - 1) \\ &= 5\left(\mathcal{F}_{\text{eval}} + 2\right) + 35M.\end{aligned} \tag{16}$$

Then, as we see in Eq. (3d), the FLOPs for $\boldsymbol{k}_7$ are just a single FLOP for a sum, and a function evaluation resulting in $\mathcal{F}_{\boldsymbol{k}_7} = \mathcal{F}_{\text{eval}} + 1$. Then, for the calculation of $\boldsymbol{u}_{n+1}$, one has to do $M\left[1 + 1 + (2 \times 6 - 1)\right] = 13M$ FLOPs [see Eq. (3c)]. Adding all of these quantities together we obtain

$$\begin{aligned}\mathcal{F}_{\text{step}} &= 5\left(\mathcal{F}_{\text{eval}} + 2\right) + 35M + \mathcal{F}_{\text{eval}} + 1 + 13M \\ &= 6\mathcal{F}_{\text{eval}} + 11 + 48M.\end{aligned} \tag{17}$$

B. Error FLOPs

In the error part of the algorithm, we start by looking at the calculation of $\boldsymbol{e}_{n+1}$ as in Eq. (6). This requires $M\left[1 + (2 \times 7 - 1)\right]$ FLOPs. Then, we need to compute $\hat{\boldsymbol{e}}_{n+1}$, which we can see from Eq. (7) that it takes about 2M FLOPs. Finally, Eq. (8) requires $3M + 1$.[1] Adding all of these together, the equation for the FLOPs of the error is

$$\mathcal{F}_{\text{error}} = M\left[1 + (2 \times 7 - 1)\right] + 2M + 3M + 1 = 19M + 1. \tag{18}$$

C. Interpolation FLOPs

1. *Polynomial factors*

For each successful RK step, if interpolation is required, the $\boldsymbol{Q}_j$'s have to be calculated. The FLOPs required to calculate Eq. (9) are

$$\mathcal{F}_{\boldsymbol{Q}_j} = M\left(2 \times 7 - 1\right) = 13M. \tag{19}$$

2. *Performing the interpolation*

Calculating $x^{(l)}$ just takes two FLOPs [see Eq. (10)], then six FLOPs to calculate $y_2^{(l)}$, $y_3^{(l)}$, and $y_4^{(l)}$ [see Eq. (11)]. Finally, the calculation of $\boldsymbol{u}_{\text{sol}}^{(l)}$ using Eq. (12), which takes $M\left[1 + 1 + (2 \times 4 - 1)\right]$ FLOPs to compute. The addition of these results gives

$$\mathcal{F}_{\text{int}} = 2 + 6 + M\left[1 + 1 + (2 \times 4 - 1)\right] = 8 + 9M. \tag{20}$$

D. Final expression FLOPs

Each step, either rejected or accepted does roughly six function evaluations per iteration, therefore the total number of steps N_{steps} can be replaced for $N_{\text{eval}}/6$, where N_{eval} is the amount of function evaluations done during the execution of the algorithm. With this, the final expression for the FLOPs can be obtained by using Eqs. (17) to (20) in Eq. (13):

$$\begin{aligned}\mathcal{F}_{\text{RK5(4)}} &\simeq N_{\text{out}}\mathcal{F}_{\text{int}} + \frac{N_{\text{eval}}}{6}\left(\mathcal{F}_{\text{step}} + \mathcal{F}_{\text{error}} + 4\mathcal{F}_{\boldsymbol{Q}_j}\right) \\ &= N_{\text{out}}\left(8 + 9M\right) + \frac{N_{\text{eval}}}{6}\left(6\mathcal{F}_{\text{eval}} + 12 + 119M\right) \\ &\simeq 9N_{\text{out}}\left(M + 1\right) + N_{\text{eval}}\left(\mathcal{F}_{\text{eval}} + 2 + 20M\right).\end{aligned} \tag{21}$$

Finally, assuming $\mathcal{F}_{\text{eval}} + 20M \gg 2$, we obtain

$$\mathcal{F}_{\text{RK5(4)}} \simeq 9N_{\text{out}}\left(M + 1\right) + N_{\text{eval}}\left(\mathcal{F}_{\text{eval}} + 20M\right). \tag{22}$$

[1] J. Dormand and P. Prince, A family of embedded Runge-Kutta formulae, Journal of Computational and Applied Mathematics **6**, 19 (1980).

[2] L. F. Shampine, Some practical Runge-Kutta formulas, Mathematics of Computation **46**, 135 (1986).

[1] Here we estimate the cost of taking the square root as a single FLOP.