Python vs. MATLAB and RKMO4 vs. `ode45`

N. Bernstein

1 Introduction

In this report we will be analyzing the differences between Python and MATLAB as tools for computational mathematics. Within these two programming environments, we will also analyze compare a Runge-Kutta Order 4 functions I have created and established ODE solvers. For Python that will be `solve_ivp` that is part of the SciPy library and for MATLAB that will be `ode45`.

2 Problem

Each of these four functions will solve the following problem from page 219 of *Numerical Methods* by Fairs and Burden:

$$y'' - 2y' + 2y = e^{2t}\sin t \text{ for } 0 \leq t \leq 1 \text{ with } y(0) = -0.4, y'(0) = -0.6$$

Which gives us the system of equations:

$$\begin{aligned} u_1'(t) &= u_2(t) \\ u_2'(t) &= e^{2t}\sin t - 2u_1(t) + 2u_2(t) \\ &\text{where } u_1(0) = -0.4, u_2(0) = -0.6 \end{aligned}$$

3 MATLAB

Using MATLAB to solve this equation, we obtain the following results.

	RKMO4		`ode45`	
t	$w_1 \approx y(t)$	$w_2 \approx y'(t)$	$w_1 \approx y(t)$	$w_2 \approx y'(t)$
0	-0.4	-0.6	-0.4	-0.6
0.1	-0.46173334	-0.63163124	-0.46173296	-0.63163105
0.2	-0.52555988	-0.64014895	-0.52555904	-0.64014866
0.3	-0.58860144	-0.61366381	-0.58860004	-0.61366361
0.4	-0.64661231	-0.53658203	-0.64661028	-0.5365822
0.5	-0.69356666	-0.3887381	-0.69356394	-0.38873907
0.6	-0.7211519	-0.14438087	-0.72114849	-0.14438325
0.7	-0.71815295	0.22899702	-0.7181489	0.22899237
0.8	-0.66971133	0.7719918	-0.66970679	0.77198375
0.9	-0.5564429	1.53478148	-0.55643816	1.53476851
1	-0.35339886	2.57876634	-0.3533944	2.57874647

3.1 Speed

As you can see, both methods calculate approximately the same results. However, execution time is also important. After one million executions of each function on the above problem, my RKMO4 took an average of 0.14 milliseconds to execute while MATLAB's `ode45` took an average of 0.32 milliseconds. This must mean that my function is much simpler than MATLAB's.

3.2 Accuracy

Solving the above problem by hand yields the following equations for y and y':

$$y(t) = u_1(t) = 0.2e^{2t}(\sin t - 2\cos t)$$
$$y'(t) = u_2(t) = 0.2e^{2t}(4\sin t - 3\cos t)$$

Comparing the exact values to what we obtained from the functions is as follows:

	Exact		RKMO4		`ode45`	
t	y(t)	y'(t)	$\lvert y - w_1\rvert$	$\lvert y' - w_2\rvert$	$\lvert y - w_1\rvert$	$\lvert y - w_2\rvert$
0	-0.4	-0.6	0	0	0	0
0.1	-0.46173297	-0.63163105	3.72E-7	1.91E-7	7.08E-10	8.26E-10
0.2	-0.52555904	-0.64014866	8.36E-7	2.84E-7	1.21E-9	2.76E-9
0.3	-0.58860004	-0.6136636	1.39E-6	1.99E-7	1.29E-9	6.28E-9
0.4	-0.64661028	-0.53658219	2.02E-6	1.68E-7	6.60E-10	1.20E-8
0.5	-0.69356394	-0.38873905	2.71E-6	9.58E-7	1.08E-9	2.06E-8
0.6	-0.72114849	-0.14438322	3.41E-6	2.35E-6	4.45E-9	3.31E-8
0.7	-0.71814889	0.22899242	4.06E-6	4.59E-6	1.01E-8	5.06E-8
0.8	-0.66970677	0.77198382	4.55E-6	7.97E-6	1.90E-8	7.44E-8
0.9	-0.55643813	1.53476862	4.77E-6	1.29E-5	3.22E-8	1.06E-7
1	-0.35339435	2.57874662	4.50E-6	1.97E-5	5.11E-8	1.48E-7

Notice that `ode45` is more accurate by about two decimal places, which is why it takes longer than my RKMO4 function. Now whether or not two decimal places is worth roughly doubling the computation time, is purely situational.

4 Python

Running Python versions of these functions yields a similar result.

	RKMO4		`solve_ivp`	
t	$w_1 \approx y(t)$	$w_2 \approx y'(t)$	$w_1 \approx y(t)$	$w_2 \approx y'(t)$
0	-0.4	-0.6	-0.4	-0.6
0.1	-0.46173297	-0.63163105	-0.461729	-0.631622
0.2	-0.52555905	-0.64014867	-0.525403	-0.639762
0.3	-0.58860004	-0.61366361	-0.588487	-0.613278
0.4	-0.64661028	-0.53658221	-0.646756	-0.536693
0.5	-0.69356395	-0.38873908	-0.693838	-0.389169
0.6	-0.7211485	-0.14438325	-0.721212	-0.144506
0.7	-0.71814891	0.22899238	-0.718076	0.229089
0.8	-0.66970679	0.77198376	-0.66962	0.772087
0.9	-0.55643817	1.53476851	-0.556409	1.53477
1	-0.35339441	2.57874647	-0.35332	2.57881

4.1 Speed

Again, after one million executions of each function on the above problem, my RKMO4 took an average of 0.31 milliseconds to execute while SciPy's `solve_ivp` took an average of 0.44 milliseconds. This is much slower than the corresponding MATLAB functions, most likely due to the increased complexity with how Python stores data in memory.

4.2 Accuracy

Comparing the exact values to what we obtained from these Python functions yields a surprising result:

	Exact		RKMO4		`solve_ivp`	
t	y(t)	y'(t)	$\lvert y - w_1 \rvert$	$\lvert y' - w_2 \rvert$	$\lvert y - w_1 \rvert$	$\lvert y - w_2 \rvert$
0	-0.4	-0.6	0	0	0	0
0.1	-0.461733	-0.631631	3.72E-7	1.91E-7	3.83E-6	8.85E-6
0.2	-0.525559	-0.640149	8.36E-7	2.84E-7	1.56E-4	3.86E-4
0.3	-0.5886	-0.613664	1.39E-6	1.99E-7	1.13E-4	3.85E-4
0.4	-0.64661	-0.536582	2.02E-6	1.68E-7	1.46E-4	1.11E-4
0.5	-0.693564	-0.388739	2.71E-6	9.58E-7	2.74E-4	4.30E-4
0.6	-0.721148	-0.144383	3.41E-6	2.35E-6	6.34E-5	1.22E-4
0.7	-0.718149	0.228992	4.06E-6	4.59E-6	7.29E-5	9.65E-5
0.8	-0.669707	0.771984	4.55E-6	7.97E-6	8.66E-5	1.03E-4
0.9	-0.556438	1.53477	4.77E-6	1.29E-5	2.90E-5	4.49E-6
1	-0.353394	2.57875	4.50E-6	1.97E-5	7.41E-5	5.97E-5

Not only is the `solve_ivp` function slower, it is not as accurate as my simple RKMO4 function. This could be because it offers other information that we are ignoring for this exercise.

5 Comparison

Unsurprisingly, MATLAB's `ode45` is the most accurate function since the MATLAB environment is optimized for these types of calculations. However, it was surprising how poorly Python performed in comparison. The MATLAB version of my RKMO4 function was about twice as fast as the Python version and just as accurate. Additionally, the Python equivalent to `ode45`, solve_ivp, was both slower and less accurate than both my RKMO4 functions as well as it's MATLAB counterpart. It would be interesting to repeat this test with a significantly more complex system of equations to see how the relative performance of each function differs.