The First-Order Initial Value Problem

Elijan Jakob Mastnak
Student ID: 28181157

November 2020

Contents

1 Theory **2**
1.1 The Temperature Equation . . . 2
1.2 Overview of Numerical Methods for Linear ODEs . . . 2
1.2.1 Explicit and Implicit Methods . . . 3
1.2.2 Error . . . 3
1.3 Descriptions of a Few Common Methods . . . 3
1.3.1 Explicit Euler Method . . . 3
1.3.2 Trapezoid Method . . . 4
1.3.3 Heun's Method . . . 4
1.3.4 Explicit Midpoint Method . . . 4
1.3.5 Implicit Midpoint Method . . . 5
1.3.6 Runge-Kutta RK4 . . . 5

2 Solution **5**
2.1 Methods Used in This Report . . . 5
2.2 The Solution to the Differential Equations . . . 6

3 Accuracy **7**
3.1 Error in the Time Domain for Fixed-Step Methods . . . 8
3.2 Reliability of Embedded Error Estimates . . . 9
3.3 Reliability of the Tolerance in Adaptive Step Methods . . . 9
3.4 Case Study: Small Step Size Is Not Always Better . . . 10

4 Efficiency **12**
4.1 Computation Time for Fixed Step Size Methods . . . 12
4.2 Computation Time for Adaptive Step Size Methods . . . 13

5 Stability **13**
5.1 Stability of Fixed-Step Method for Large Step Sizes . . . 14
5.2 Stability of Adaptive-Step Methods for Large Tolerances . . . 15

Assignment

1. Investigate the numerical solutions of the first-order differential equation (1) with the initial temperatures $T(0) = 21$ or $T(0) = -15$ and an external temperature $T_{\text{ext}} = -5$ with the parameter $k = 0.1$ using as many different numerical methods as possible. How large a step size is needed for each method? Compare efficiency, accuracy and stability of the methods among each other, using the analytic solution (1.1) for reference.

2. *Optional*: Perform a similar analysis of Equation 2, which includes a sinusoidal term to model temperature variation across the day, using the parameters $k = 0.1$ and $\delta = 10$. Experiment with varying the periodic term's amplitude A. Which differential equation method is best suited to accurately determining the value and time of the maximum daily temperature?

---

1 Theory

To jump right to the solution, see Section 2.
Note that this section is rather long, largely to solidify my own understanding.

1.1 The Temperature Equation

For this report we consider the first-order linear differential equation

$$\frac{\mathrm{d}T}{\mathrm{d}t} = -k(T - T_{\text{ext}}) \tag{1}$$

which is a simple model of the time dependence of the temperature in a room whose walls have thermal conductivity modeled by k, with an external temperature T_{ext}. The equation's analytic solution is

$$T(t) = T_{\text{ext}} + e^{-kt}\big[T(0) - T_{\text{ext}}\big]$$

A better model for the temperature in a room might include a periodic term to model change in temperature throughout the day, of the form

$$\frac{\mathrm{d}T}{\mathrm{d}t} = -k(T - T_{\text{ext}}) + A\sin\left[\frac{2\pi}{24}(t - \delta)\right] \tag{2}$$

1.2 Overview of Numerical Methods for Linear ODEs

This report focuses on numerically solving the first-order initial value problem

$$y' \equiv \frac{\mathrm{d}y}{\mathrm{d}t} = f(y, t)$$

with the initial condition $y(t_0) = y_0$ on the interval $t \in [a, b]$. This is done by first dividing the time interval $[a, b]$ into n partition points $\{t_i\}_{i=0}^{n-1}$ where

$$a = t_0 < t_1 < \cdots < t_{n-1} < t_n = b$$

In theory, the spacing between individual points could be arbitrary, but in practice we choose uniformly spaced points separated by a *time step* $h = t_{i+1} - t_i$. The problem reduces to finding n numerical approximations y_i to the analytic solution y at each of the n partition points t_i. We then approximate the solution y by the values y_i at the partition points.

1.2.1 Explicit and Implicit Methods

The approximations y_i are found either with *explicit* or *implicit* methods. An explicit method starts with the initial value y_0 and calculates successive approximations directly on the basis of the previous approximations, for example

$$y_{i+1} = \Phi(h, x_i, y_{i-k}, \dots, y_{i-1}, y_i)$$

An implicit method calculates approximations by solving a (generally non-linear) equation involving previous approximations and the next approximation, e.g.

$$y_{i+1} = \Phi(h, x_i, y_{i-k}, \dots, y_i, y_{i+1})$$

Both explicit and implicit methods can be either *single step* or *multi-step.* In single step methods, the next approximation y_{i+1} is found only from the previous approximation y_i. In a multi-step, e.g. k-step method, the next approximation y_{i+1} is found with the k previous approximations $y_i, y_{i-1}, \dots y_{i-k-1}$.

1.2.2 Error

Estimates for local truncation error come from comparing the method to the function's Taylor expansion. Error is expressed in powers of the step size h, e.g. $\mathcal{O}_\mathrm{l}(h^p)$. Higher order methods have lower error. *Local error* is the difference between the numerical approximation and analytic solution at a single point while *global error* is sum of all local errors. Since we make N steps $N = \frac{b-a}{h}$ which scale as $\sim h^{-1}$, the global error of a $\mathcal{O}_\mathrm{l}(h^p)$ method scales as

$$\mathcal{O}_\mathrm{g} = N \cdot \mathcal{O}_\mathrm{l} \sim \frac{1}{h} \cdot h^p = \mathcal{O}_\mathrm{g}(h^{p-1})$$

The global error is one order less than the local truncation error. When error is given without reference to local or global, it is usually refers to global error.

1.3 Descriptions of a Few Common Methods

In all cases below, the function f gives the value of the derivative y'.

1.3.1 Explicit Euler Method

The simplest and most primitive method. A nice way to remember it is as a rearranged version of the first-order Taylor series approximation for the derivative:

$$y' \approx \frac{1}{h}\big[y(x+h) - y(x)\big] \implies y(x+h) = y(x) + h\frac{\mathrm{d}y}{\mathrm{d}x}\bigg|_x$$

A numerical implementation starts with the initial condition $y(t_0) = y_0$ and calculates successive points using

$$y_{i+1} = y_i + hf(t_i, y_i) \qquad \text{for} \qquad i = 0, 1, \ldots, n-1$$

The explicit Euler method has local order $\mathcal{O}(h^2)$ and global order $\mathcal{O}(h)$.

1.3.2 Trapezoid Method

The trapezoid method reads

$$y_{i+1} = y_i + \frac{h}{2}\big[f(t_i, y_i), f(t_{i+1}, y_{i+1})\big]$$

It is an implicit, linear multistep method. The method comes from the trapezoid method for integration; integrating the differential equation from t_i to t_{i+1} and applying the trapezoid rule gives

$$y(t_{i+1}) - y(t_i) = \int_{t_i}^{t_{i+1}} f\big(t, y(t)\big)\,\mathrm{d}t \approx \frac{h}{2}\big[f(t_i, y(t_i)), f(t_{i+1}, y(t_{i+1}))\big]$$

Using the approximations $y_i \approx y(t_i)$ and solving for y_{i+1} gives

$$y_{i+1} = y_i + \frac{h}{2}\big[f(t_i, y_i), f(t_{i+1}, y_{i+1})\big]$$

1.3.3 Heun's Method

Heun's method is a 2nd order explicit method and reads

$$\begin{aligned}\tilde{y}_{i+1} &= y_i + hf(t_i, y_i)\\ y_{i+1} &= y_i + \frac{h}{2}\big[f(t_i, y_i) + f(t_{i+1}, \tilde{y}_{i+1})\big]\end{aligned}$$

This is a simple *predictor-corrector* method. The first step uses Euler's method to predict the value $\tilde{y}_{i+1}$. This intermediate value is then used with the trapezoid method to generate an improved guess y_{i+1}. Two evaluations of f are required per step, once for $f(t_i, y_i)$ (used twice) and once for $f(t_{i+1}, \tilde{y}_{i+1})$.

1.3.4 Explicit Midpoint Method

The explicit midpoint method reads

$$y_{i+1} = y_i + hf\left(t_i + \tfrac{h}{2}, y_i + \tfrac{h}{2}f(t_i, y_i)\right)$$

The method uses the value of the derivative at the midpoint $t_i + \frac{h}{2}$ between t_i and t_{i+1}. In equivalent, Runge-Kutta form, the midpoint method reads

$$\begin{aligned}k_1 &= f(x_i, y_i), \qquad k_2 = f\left(x_i + \tfrac{h}{2}, y_i + \tfrac{h}{2}k_1\right)\\ y_{i+1} &= y_i + hk_2\end{aligned}$$

In this form the midpoint method is often called the RK2 method. The method has global order $\mathcal{O}(h^2)$. Two function evaluations are required per step. The midpoint method is a single-step method with an intermediate half-step stage, which is discarded after each complete step.

1.3.5 Implicit Midpoint Method

The implicit midpoint method reads

$$y_{i+1} = y_i + hf\left(t_i + \tfrac{h}{2}, \tfrac{1}{2}(y_i + y_{i+1})\right)$$

The derivative is evaluated at the t coordinate corresponding to the midpoint $t_i + \frac{h}{2}$ and the y coordinate corresponding to the average of the y values at the start and end of the step.

1.3.6 Runge-Kutta RK4

A canonical choice for its combination of efficiency and accuracy. Starting with $y(t_0) = y_0$, we calculate the coefficients

$$k_1 = f(t_i, y_i) \qquad k_2 = f\left(t_i + \tfrac{h}{2}, y_i + \tfrac{h}{2}k_1\right)$$
$$k_3 = f\left(t_i + \tfrac{h}{2}, y_i + \tfrac{h}{2}k_2\right) \qquad k_4 = f(t_i + h, y_i + hk_3)$$

and calculate the next term with the formula

$$y_{i+1} = y_i + \frac{h}{6}\big(k_1 + 2k_2 + 2k_3 + k_4\big) + \mathcal{O}(h^5), \quad i = 0, 1, \dots, n-1$$

The method has global order $\mathcal{O}(h^4)$ and requires four function evaluations per step. It is a single-step method with four intermediate stages; these are discarded after each complete step.

2 Solution

2.1 Methods Used in This Report

As encouraged in the instructions, I tried to test many methods, so this list is rather long. I include a brief description and the `function_name` used in this report.
Since the methods are all well-known, I chose not to include code in the report itself, but each function appears in the attached source code file `ivp.py`.

Single-step methods from the Runge-Kutta family with fixed step size:

- Euler method `euler`
- Heun's method `heun`
- Explicit midpoint method `rk2a`
- Alternate 2nd order RK2 method `rk2b`
- Ralston's 3rd Runge-Kutta method `rk3r`
- 3rd order strong stability preserving Runge-Kutta `rk3ssp`
- Classic 4th order Runge-Kutta `rk4`
- The 3/8 4th order Runge-Kutta method `rk438` (which was included along with the canonical RK4 in Kutta's original 1901 paper)

- Ralston's 4th order Runge-Kutta method `rk4r` (supposedly minimizes local truncation error)
- Fixed-step embedded 4-5th order Runge-Kutta with error estimate `rk45`

Single-step, embedded adaptive-step Runge-Kutta methods:

- Adaptive 2-3rd order Bogacki–Shampine method `bs23` (supposedly used in Matlab's `ode23`)
- Adaptive 4-5th order Runge-Kutta-Fehlberg method `rkf45`
- Adaptive 4-5th order Cash-Karp method `ck45`
- Adaptive 4-5th order Dormand–Prince method `dp45` (supposedly used in Matlab's `ode45`)

Multistep methods

- Alas, only the single 4th order multi-step predictor-corrector Adams-Bashforth-Moulton method `pc4`. I ran out of time and energy to implement more of my own, despite the importance of multistep methods in scientific computing.

Interesting ideas for another day

- Investigate the backward differentiation family of multistep methods; use more multistep methods in general.
- Investigate the behavior of explicit versus implicit methods when applied to stiff equations.
- Solving implicit equations with fixed-point iteration.

2.2 The Solution to the Differential Equations

Before wading knee-deep in the details of numerical methods, it feels appropriate to first show the solution to the temperature in the room! Figure 1 shows both the simple exponentially decaying solution and the oscillating solution for positive and negative initial temperature T_0. The plotted solutions were found with the Adams-Bashforth-Moulton predictor-corrector `pc4`, but on the macroscopic scale on display in Figure 1 all methods give essentially indistinguishable results.
As seen in Figure 1, the basic exponential model decays to the external temperature T_{ext} on a time scale determined by the parameter k. The oscillating model also approaches the external temperature T_{ext}, about which it oscillates with amplitude A over a period of 24 hours upon reaching the steady state at large t.

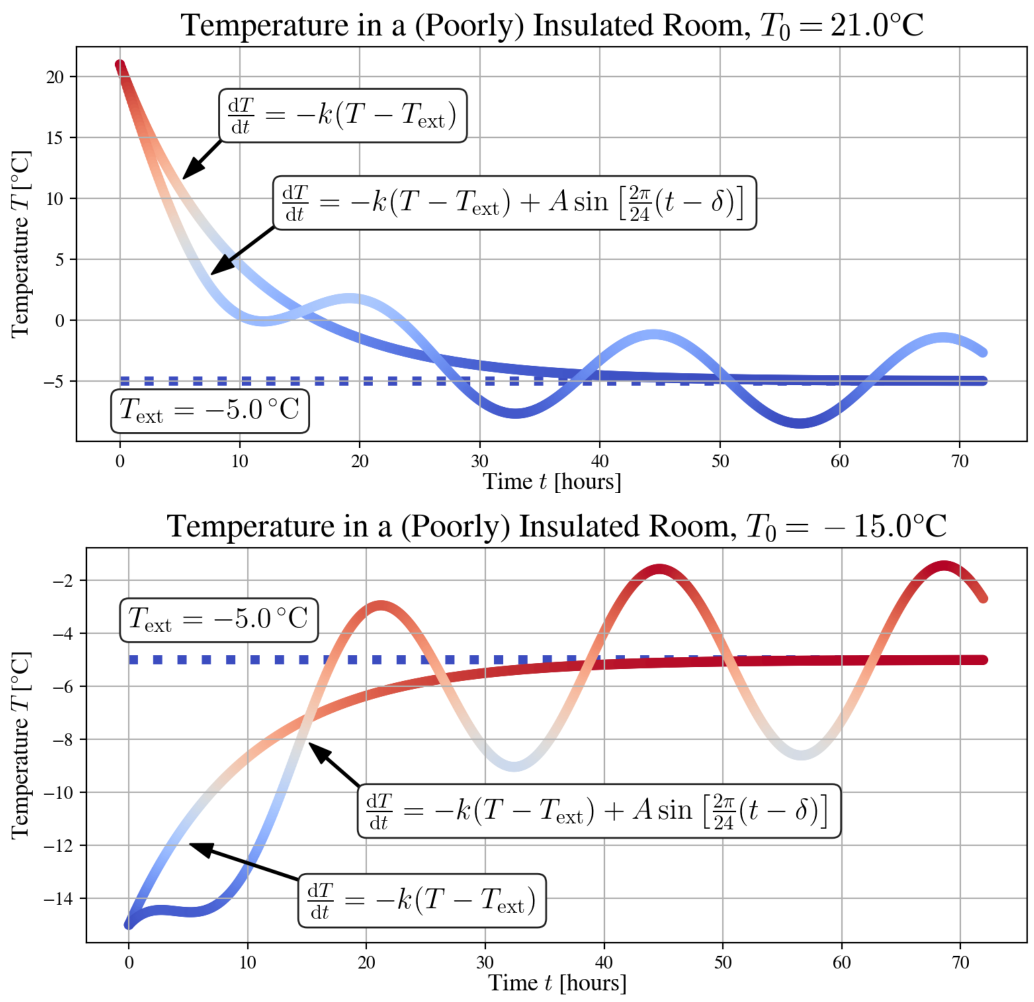

Figure 1: Exponentially decaying and oscillating solutions with initial conditions $T_0 = 21\,°\mathrm{C}$ and $T_0 = -15\,°\mathrm{C}$ found with the 4th order Adams-Bashforth-Moulton predictor-corrector `pc4`. Tested with $k = 0.1$, $A = 1$ and $\delta = 10$.

3 Accuracy

I used the absolute error of each numerical solution relative to the analytic solution in Equation 1.1 as a measure of each method's accuracy—I worked with the simpler model in Equation 1 when testing accuracy because this method has a convenient analytic solution as a reference for error. I focused on the following:

1. Dependence of error on time at a fixed time step h (for fixed time step methods).
2. Agreement (or disagreement) between the error predicted by embedded methods and error with respect to the analytic solution.
3. The reliability of the tolerance parameter for adaptive-step methods.
4. Dependence of error on step size for fixed time step methods.

First Note: I encountered a dilemma when plotting error: I could either present error in a linear scale and preserve the sign (which could be either positive or negative with respect to the analytic solution, depending on the method) or plot the absolute value of error on a better-suited logarithmic scale (naturally, I couldn't plot negative values on a logarithmic scale). I opted for the latter, so keep in mind: all error quantities are absolute values, which hides information about the error's sign.

Second Note: I was motivated by the Airy function report to also investigate relative error with respect to the analytic solution and found an understandably large spike whenever temperature crossed zero. Since this spike completed dominated the error in the rest of the time domain, I focused only on absolute error, which naturally avoids the risk of divergence near zero. Absolute error would be more interesting in most practical applications anyway.

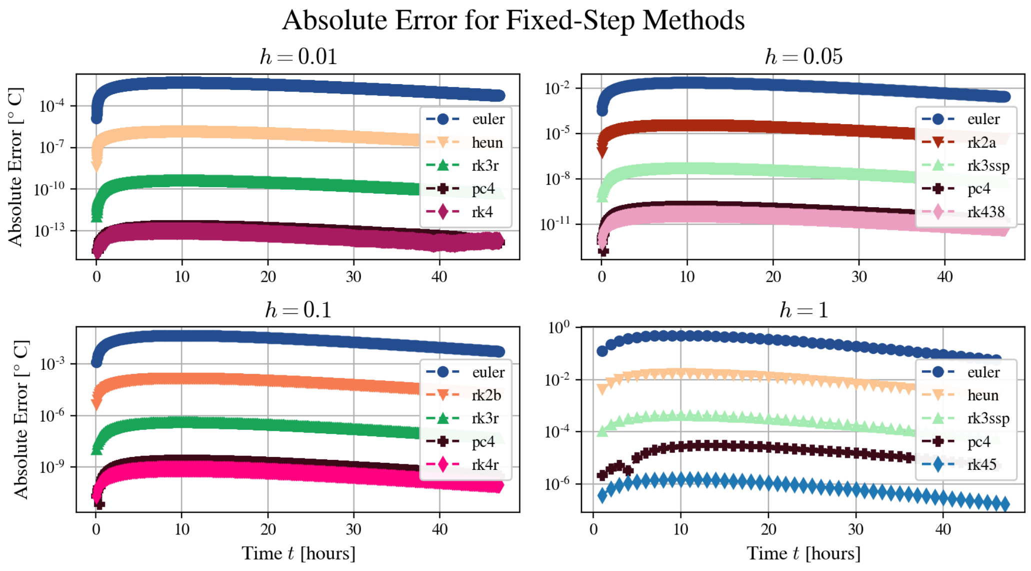

Figure 2: Error of fixed-step methods (progressing vertically from 1st order Euler to 4th order Runge-Kutta) in the time domain over the course of a 48 hour simulation for various time steps h. Tested with $k = 0.1$ and $T_0 = 21\,°\mathrm{C}$.

3.1 Error in the Time Domain for Fixed-Step Methods

Figure 2 shows the error of solutions found with the fixed-step methods in the time domain for various time steps h. Accuracy improves in roughly uniform steps of two to three orders of magnitude with each method order. Note the vast numerical range across the various methods—over 10 orders of magnitude for $h = 0.01$, from the 1st order Euler method with $\epsilon \sim 10^{-3}$ to the 4th order `rk4` and `pc4` methods with $\epsilon \sim 10^{-14}$. I was surprised to find the 4th order Runge-Kutta methods slightly but consistently outperformed the multistep `pc4` on a basis of accuracy.

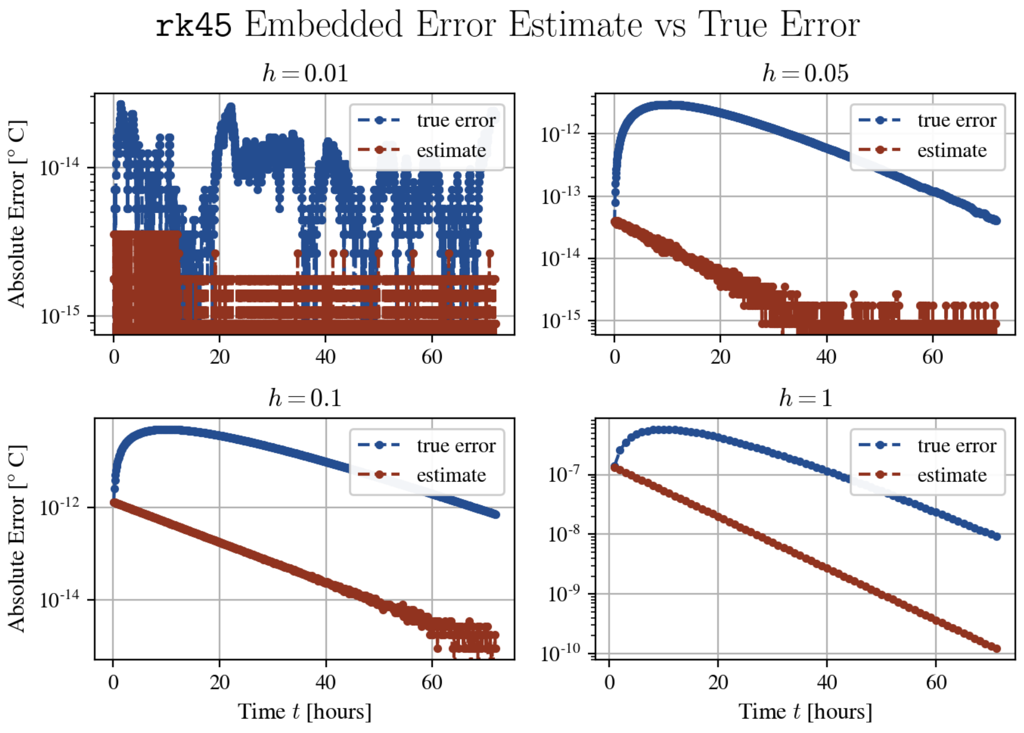

Figure 3: Error estimated by the embedded `rk45` method compared to the true error with respect to the analytic solution. Note that the embedded method consistently underestimates its error! Tested with $k = 0.1$ and $T_0 = 21\,°\mathrm{C}$ over 72 hours.

3.2 Reliability of Embedded Error Estimates

Figure 3 compares the error estimated by the `rk45` method—a 4-5th order embedded method—to the true error with respect to the analytic result. I was surprised by onset of local oscillations in error for small step sizes (see the $h = 0.01$ and $h = 0.05$ subplots). Conspicuously, the embedded method underestimates its error across a range of step sizes, and, although I only plotted the `rk45` method, other embedded methods show similar behavior.

3.3 Reliability of the Tolerance in Adaptive Step Methods

Note: I implemented three adaptive-step methods by hand (the 2-3rd order Bogacki–Shampine method `bs23`, the 4-5th order Cash-Karp method `ck45` and the 4-5th order Dormand–Prince method `dp45`), in addition to the provided `rkf45`. Of these, it appears only `ck45` works as it should. Although both `bs23` and `dp45` give "macroscopically" acceptable results (they appear identical to the analytical solution on a scale of e.g. $T \sim 1\,°\mathrm{C}$), *I have strong reason to believe my implementations contain bugs*, which remained stubbornly hidden over the course of my unsuccessful debugging process. Rather than hide this semi-failure by leaving the offending methods out, I included `bs23` and `dp45` to show what could go wrong.
Figure 4 shows error in the time domain for solutions found with the four adaptive-step methods for various tolerances ϵ. Note how `rkf45` and `ck45` work properly; `bs23`

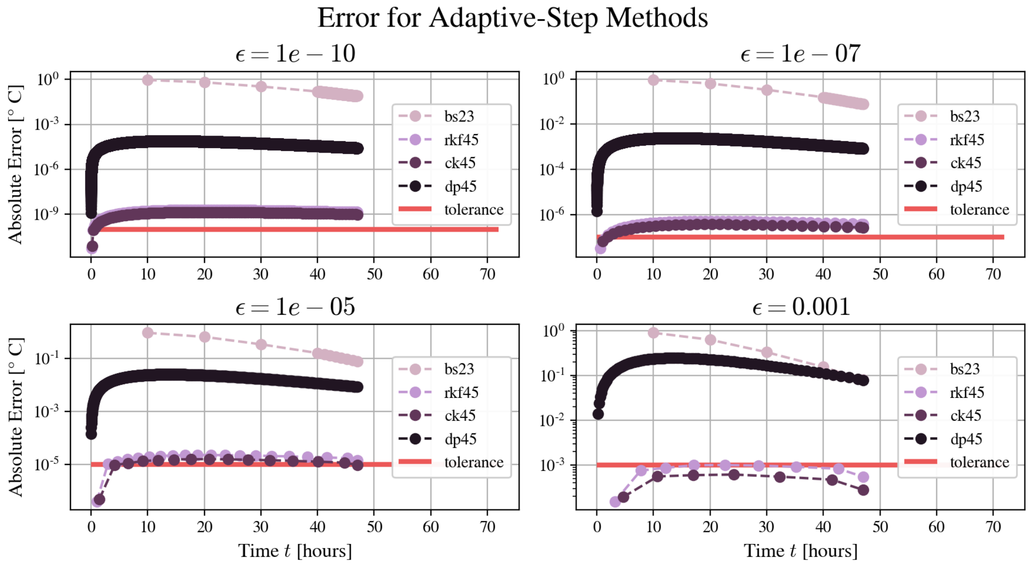

Figure 4: Error for four adaptive-step methods for various tolerances ϵ, shown with a red line for reference. `rkf45` and `ck45` work as intended, while my implementations of `bs23` and `dp45` are likely "buggy", i.e. the poor performance is my responsibility and not the fault of method in principle. Tested with $k = 0.1$ and $T_0 = 21\,°\mathrm{C}$.

drastically underestimates its error with respect to the inputted tolerance ϵ; `dp45` both underestimates its error and computes *far* too many than should be necessary, e.g. compare `dp45`'s densely-spaced points in the $\epsilon = 0.001$ subplot to the other methods' sparse solutions. These extremely dense points imply my `dp45` needs a much smaller step size than it should to meet the inputted tolerance; evidently the partial steps don't converge properly to a correct approximation for the next point. Perhaps a coefficient in my Butcher tableau is slightly off, but I could not for the life of me find which one.

3.4 Case Study: Small Step Size Is Not Always Better

Intuition suggests that an arbitrarily small step size could produce an arbitrarily accurate numerical solution. Unfortunately, floating point arithmetic has other thoughts. As the step size (and thus the solution increments T_{i+1}) grow very small, the increments T_{i+1} become comparable to and eventually overshadowed by the computer's limit of floating point precision. Worse yet, a small step size generates an enormous number of approximation points and thus an accelerated accumulation of numerical error. The end result, at least for higher-order methods, is a an increase in error for excessively small step size, as shown in Figure 5.

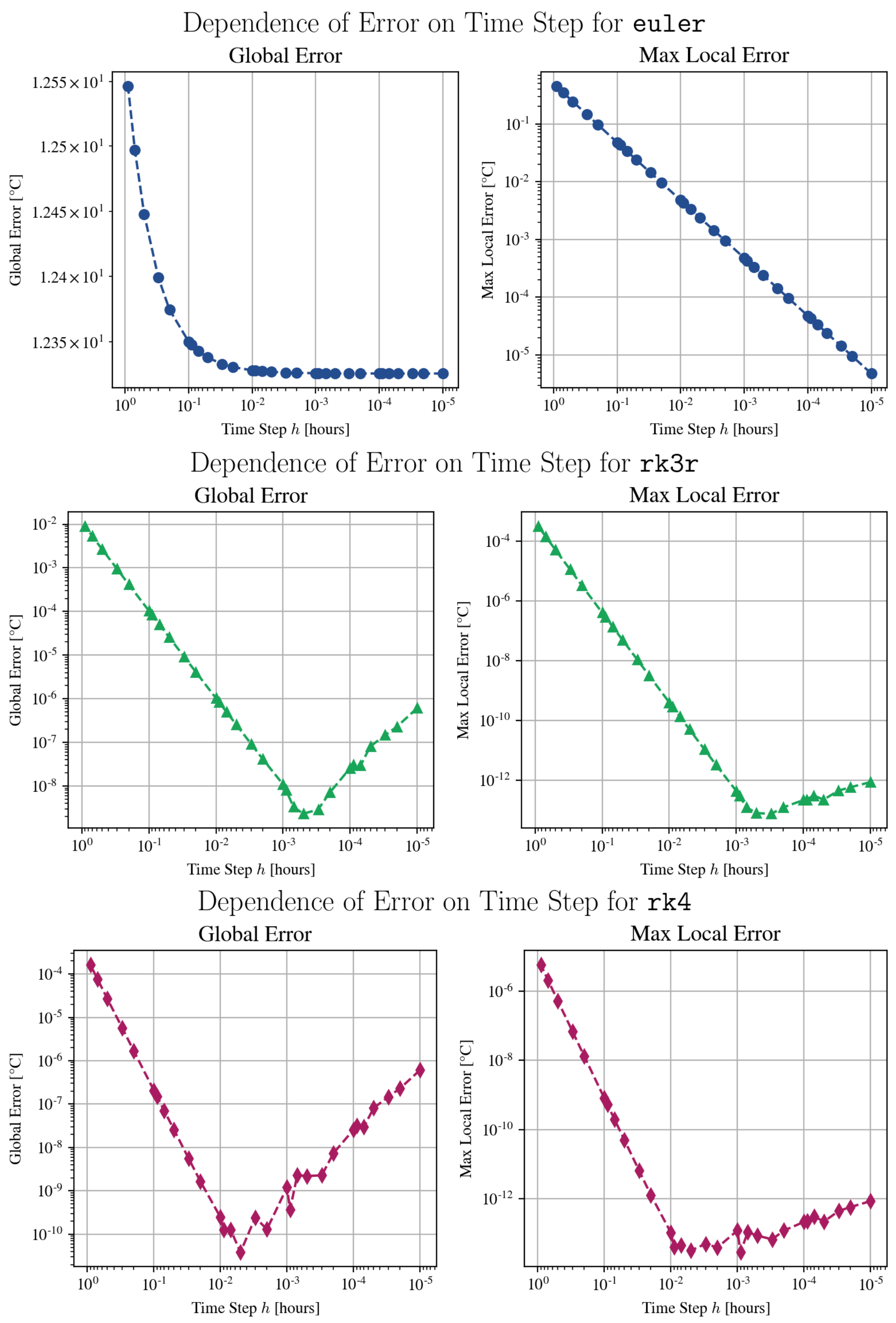

Figure 5: Dependence of global and maximum local error on step size h for `euler`, `rk3r` and `rk4`—note that h decreases from left to right. The 1st order `euler` does not reach an optimal step size, although the global error fails to improve appreciably beyond $h \sim 10^{-2}$. The higher-order `rk3r` and `rk4` methods reach an optimal step size near $5 \cdot 10^{-4}$ and $5 \cdot 10^{-3}$. Tested over 48 hours with $k = 0.1$ and $T_0 = 21\,°\mathrm{C}$.

4 Efficiency

I evaluated efficiency by measuring the time required by each method to produce a solution with a given error tolerance ϵ (for adaptive-step methods) or a given upper bound on maximum local error (for fixed-step methods). Another metric for efficiency might be computation time needed to find a solution for a given step size h. Since, in practice, we are more interested in the accuracy of the solution than the step size needed to achieve it, I decided it made more sense to measure computation time versus tolerance/error. Since the computation times for this problem were short even for small step sizes, I measured each computation time over an average of 10 runs to get a more precise result than a single run could give.

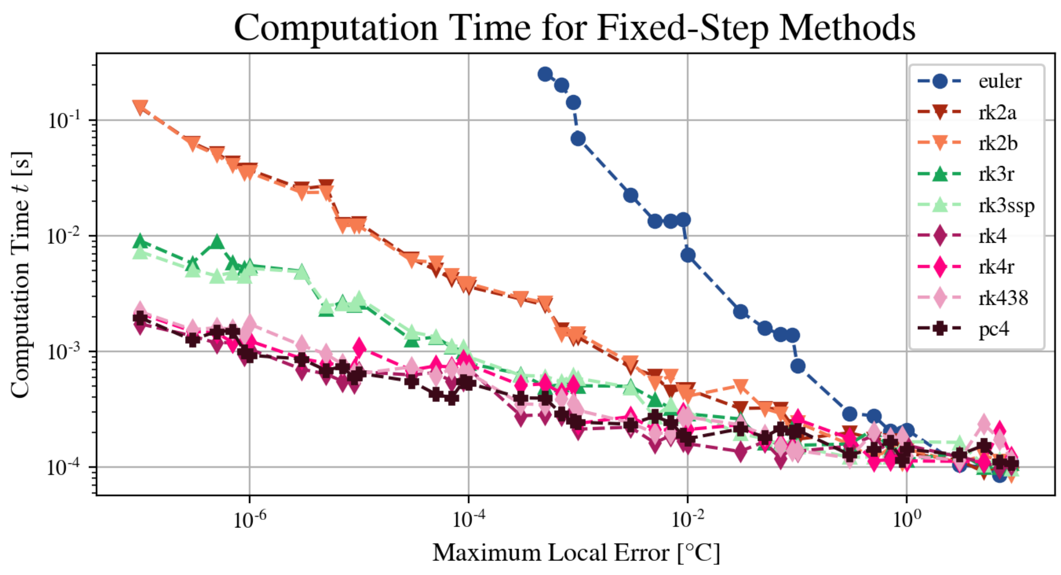

Figure 6: Computation time as a function of maximum permissible local error for each fixed-step method. Note the decrease in computation time with method order for a given tolerance.

4.1 Computation Time for Fixed Step Size Methods

I found computation time versus max local error for fixed-step methods in two steps:

1. Calculate solutions with each fixed-step method over a large range of step sizes h and record the maximum local error in each solution.

2. Use the above data to create a bijective map between step size and maximum local error for each method. With the step sizes spaced closely enough, this map gives a good estimate of the step size needed for a given tolerance.

I then used this map to measure computation time t as a function of step size h as a function of tolerance ϵ, i.e. $t\big(h(\epsilon)\big)$. Figure 6 shows the results; naturally, smaller error corresponds to longer computation time. Computation time improved with method order for each method tested; Although higher-order methods would take longer for a fixed step size h because of the larger number of calculations per

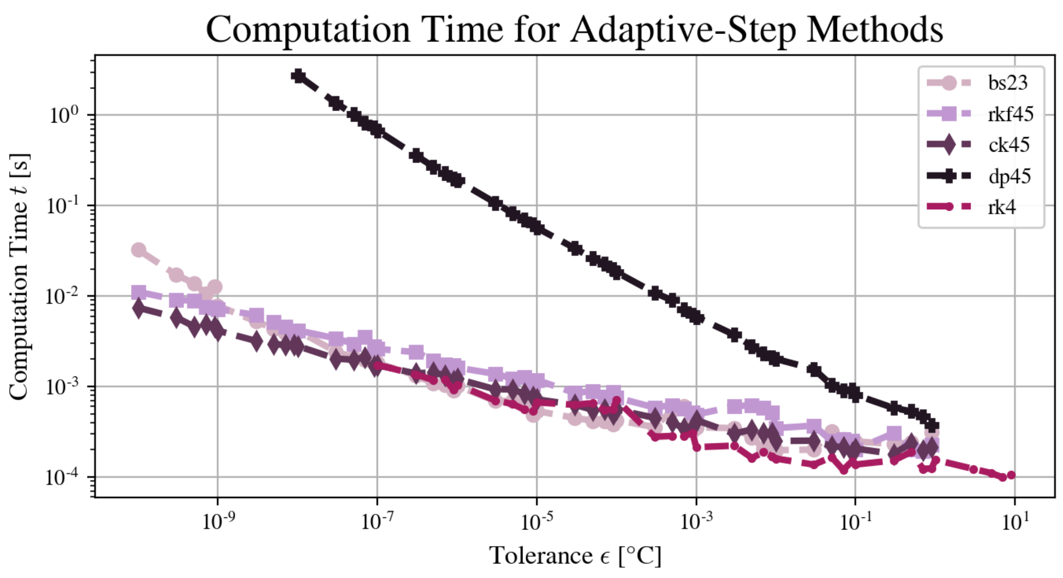

Figure 7: Computation time as a function of tolerance for each adaptive-step method. Aside from the "buggy" `dp45` (see the discussion in Subsection 3.3) all adaptive-step methods are comparably efficient and stack up favorably against even the fastest fixed-step methods (`rk4` shown for reference).

step, they nonetheless come out more efficient than low-order methods on a basis of tolerance or maximum local error.

4.2 Computation Time for Adaptive Step Size Methods

Adaptive-step methods allow a direct specification of tolerance, so measuring computation time versus tolerance is straightforward. Figure 7 shows the results with the fixed-step `rk4` for reference. Note the (relatively) terrible performance of my `dp45` implementation, which is another good indication of the unresolved bugs mentioned in Subsection 3.3). The ill-fated `dp45` aside, all adaptive-step methods perform on-par, if not better than, the fastest fixed-step methods in Figure 6.

5 Stability

I focused on the following when measuring stability:

1. Convergence (or divergence) of the fixed-step methods for progressively increasing step size h
2. Convergence (or divergence) of the adaptive-step methods for progressively increasing tolerance ϵ
3. The stability of both method families for varying values of the parameter k and initial temperature T_0.

I used both the simple model in Equation 1 and the oscillating model in Equation 2 to test stability and ran measurements over a 10 day (240 hour) period.

5.1 Stability of Fixed-Step Method for Large Step Sizes

Figures 8 and 9 show the solutions of selected fixed-step methods for increasing step sizes. Naturally, as h increased the methods began to diverge. Some observations:

- Of the fixed-step methods tested, Euler's method impressed me with its stability even for moderate step sizes of $h = 15$ hours, while the multistep predictor-corrector `pc4` surprised me with its relative instability.
- Second-order methods, e.g. `rk2a` and `rk2b` were relatively unstable, while the 4th order Runge-Kutta methods `rk4`, `rk4r` and `rk438` performed best out of the fixed-step methods I tested, with `euler` also reliable.
- Small changes in initial temperature T_0 did not noticeably affect the methods' stability. Meanwhile, all fixed-step methods grew unstable with an increasing exponential parameter k, which leads to a rapidly-changing initial part of the solution. The fixed-step methods tended to fail for $k > 0.3$ for step sizes larger than five hours and remained stable for $k > 1.0$ only for step sizes below one hour, which is quite small relative to the 240 hour simulation time.

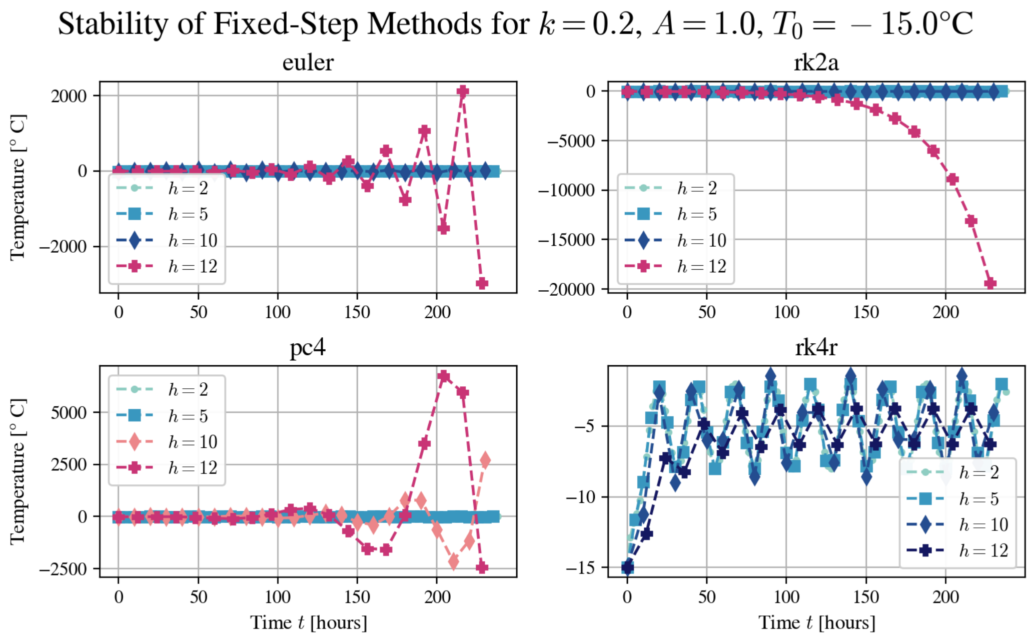

Figure 8: Stability of fixed-step methods for various values of h with the oscillating model. `euler` and the fourth-order methods `rk4`, `rk4r` and `rk438` performed best.

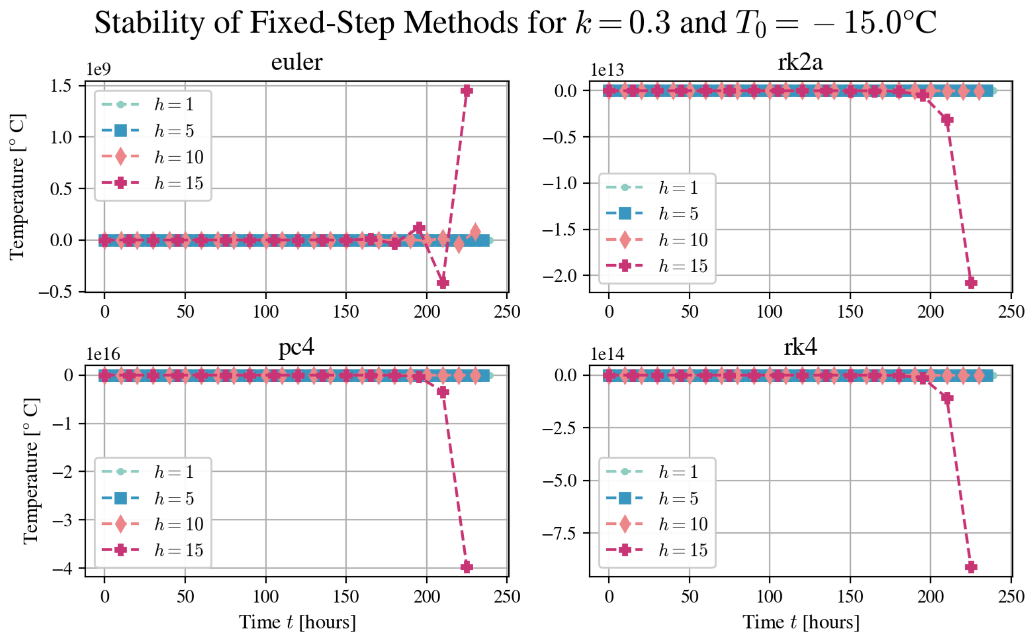

Figure 9: Testing stability of fixed-step methods for various values of h with the basic exponential model. All methods fail for $h > 5$ at $k = 0.3$.

5.2 Stability of Adaptive-Step Methods for Large Tolerances

Figure 10 shows the solutions of selected adaptive-step methods for progressively larger tolerances—I only tested `rkf45` and `ck45`, which I was fairly certain worked properly. My takeaways:

- As for fixed-step methods, changes in initial temperature T_0 did not noticeably affect the methods' stability, so I did not include graphs.
- Although step-size h and tolerance ϵ cannot be directly compared, the results in Figures 10 suggest that adaptive-step methods outperform fixed-step methods on grounds of stability. Even for absurdly large tolerances (e.g. $\epsilon = 50\,°\mathrm{C}$ on a temperature scale with amplitudes $A = 1\,°\mathrm{C}$), the adaptive-step methods remain relatively stable for k values at which the fixed-step methods consistently diverged.
- Note that both adaptive-step understandably suffer in accuracy for the solutions shown in Figure 10. This is understandable, since the tolerance is intentionally made excessively large, and I did not count this inaccuracy against the methods, since the purpose of this section was to study stability. Despite considerable oscillations about the correct value, both adaptive-step methods remain consistently stable throughout the 10-day simulation.

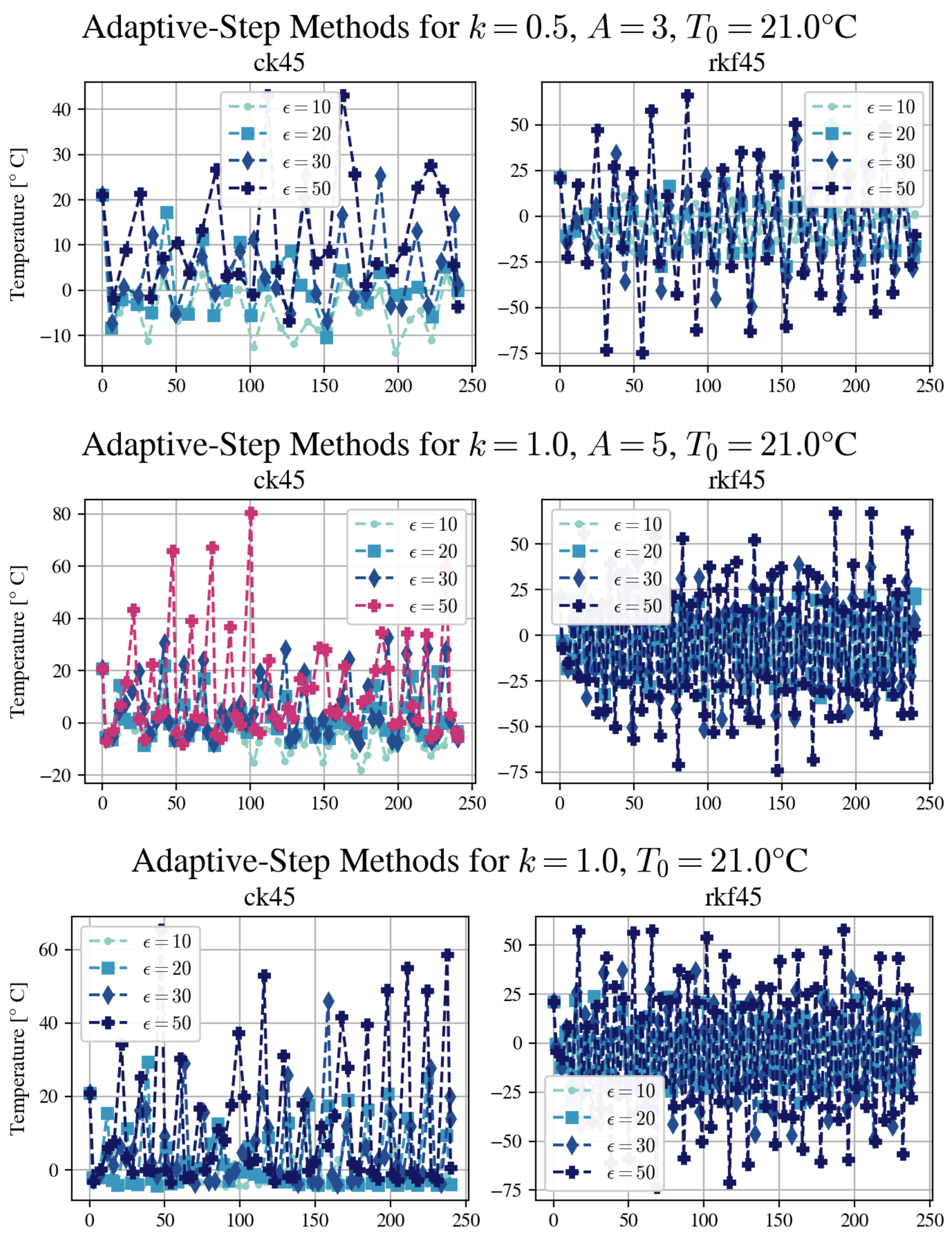

Figure 10: Testing stability of adaptive-step methods for various values of ϵ and k; the figure shows solutions for both the oscillating model (top two graphs) and simple exponential model (bottom graph). Although both methods understandably suffer in accuracy as ϵ increases, they remain stable even for large k.

References

[1] Wikipedia contributors. “List of Runge–Kutta methods.” *Wikipedia, The Free Encyclopedia*. 28 August 2020. https://en.wikipedia.org/wiki/List_of_Runge-Kutta_methods

[2] Press, William and Teukolsky, Saul. “Adaptive Stepsize Runge-Kutta Integration”. *Computers in Physics* **6**, 188 (1992); doi: 10.1063/1.4823060. https://aip.scitation.org/doi/pdf/10.1063/1.4823060.

[3] Deturck, Dennis and Wilf, Herbert. *Lectures on Numerical Analysis*. Department of Mathematics, University of Pennsylvania. Philadelphia, 2002. https://www.math.upenn.edu/~wilf/DeturckWilf.pdf.