Goals

In this assignment, the goal is to learn the basics of adaptive time-stepping methods for initial value ODEs. There are three distinct learning and training topics here:

1. **Explicit Runge–Kutta methods and automatic step size selection.** Here you will implement your own ODE solver, based on an explicit Runge–Kutta method with error estimator. The error estimator is used to adjust the time step $h$ along the integration so that the error is always kept close to a prescribed accuracy tolerance $\text{tol}$. You will use your method to solve a few nonlinear oscillatory systems.

2. **Stiff vs. nonstiff problems.** Here we focus on understanding what a “stiff” problem is, and why they have to be integrated using implicit methods. We will study this question for a nonlinear oscillatory system, the van der Pol equation. You will work with your own solver, and one of MATLAB’s built-in stiff ODE solvers, `ode15s`, which is based on the Backward Differentiation Formulas (BDFs).

3. **The Lotka–Volterra equation.** Here you will investigate a periodic problem with invariants, and compare the explicit Euler method to your own adaptive Runge–Kutta solver.

**Your written project report should not exceed 10 pages.** The report will be graded; your individual performance counts as part of the final exam. You are encouraged to discuss problems and technicalities with your fellow students, but in your report you must acknowledge in writing those persons, including the instructors, who have contributed to your understanding and results.

**Part 1. Explicit Runge–Kutta methods**
Theory

An explicit Runge–Kutta method for the initial value problem \( y' = f(t, y) \) is a method of the form exemplified by the classical 4th order Runge–Kutta method (also known as “RK4”)

\[
\begin{align*}
Y_1' &= f(t_n, y_n) \\
Y_2' &= f(t_n + h/2, y_n + hY_1'/2) \\
Y_3' &= f(t_n + h/2, y_n + hY_2'/2) \\
Y_4' &= f(t_n + h, y_n + hY_3') \\
y_{n+1} &= y_n + \frac{h}{6} (Y_1' + 2Y_2' + 2Y_3' + Y_4').
\end{align*}
\]

A single step of the method can then be described as follows. The method “samples” the right-hand side \( f(t, y) \) at four different points by computing the four “derivatives” \( Y_1', Y_2', Y_3' \) and \( Y_4' \). Then it forms a linear combination of these derivatives to obtain the “average derivative” over the step from \( y_n \) to \( y_{n+1} \). This method of order \( p = 4 \) was first derived in 1895 by Karl Runge and Wilhelm Kutta.

A general Runge–Kutta method depends on its coefficients \( a_{ij} \) for evaluating the \( Y_i' \), and the coefficients \( b_j \) for forming the linear combination of these derivatives to update the solution in a single step. The method can be written

\[
\begin{align*}
Y_i' &= f(t_n + c_i h, y_n + \sum_{j=1}^{s} a_{ij} hY_j'); \quad i = 1, \ldots, s \\
y_{n+1} &= y_n + \sum_{j=1}^{s} h b_j Y_j'.
\end{align*}
\]

We see that the method is represented by two coefficient vectors, \( c \) and \( b \), and the coefficient matrix \( A \), arranged in the Butcher tableau

\[
\begin{array}{c|cccc}
c & A \\
y & b^T
\end{array}
\]

In the particular case of the classical RK4 method, the Butcher tableau is

\[
\begin{array}{c|cccc}
0 & 0 & 0 & 0 & 0 \\
1/2 & 1/2 & 0 & 0 & 0 \\
1/2 & 0 & 1/2 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 \\
y & 1/6 & 1/3 & 1/3 & 1/6
\end{array}
\]

Task 1.1

Write a function

\[
\text{RK4step}(f, uold, told, h) \rightarrow \text{unew}
\]
that takes a single step with the classical RK4 method.
Here $f$ is the function defining differential equation.
Then test it by solving a simple problem of your choice. For example, you could try the linear test equation $y' = \lambda y$, and you can try to verify that the error is $O(h^4)$ by plotting the error in a log–log diagram, like you did in the first computer exercise for the Euler methods. Note that you now need to define the differential equation in a separate MATLAB m-file.

**Theory**

In a similar manner, a 3rd order RK method called RK3 is defined by the Butcher tableau

\[
\begin{array}{c|cccc}
0 & 0 & 0 & 0 & \\
1/2 & 1/2 & 0 & 0 & \\
1 & -1 & 2 & 0 & \\
z & 1/6 & 2/3 & 1/6 & \\
\end{array}
\]

Here we see that some of the evaluations of the right-hand side $f$ are the same as for the classical RK4 method. In fact, we can write both methods simultaneously as

\[
\begin{align*}
Y'_1 &= f(t_n, y_n) \\
Y'_2 &= f(t_n + h/2, y_n + hY'_1/2) \\
Y'_3 &= f(t_n + h/2, y_n + hY'_2/2) \\
Z'_3 &= f(t_n + h, y_n - hY'_1 + 2hY'_2) \\
Y'_4 &= f(t_n + h, y_n + hY'_3) \\
y_{n+1} &= y_n + \frac{h}{6} (Y'_1 + 2Y'_2 + 2Y'_3 + Y'_4) \\
z_{n+1} &= y_n + \frac{h}{6} (2Y'_2 + Z'_3 - 2Y'_3 - Y'_4).
\end{align*}
\]

The interesting property about this is that with five evaluations of the right-hand side $f$ instead of four (the extra evaluation is $Z'_3$), we can obtain both a 3rd order approximation $z_{n+1}$ and a 4th order approximation $y_{n+1}$ from the same starting point $y_n$.

**This has the very important implication that we can estimate a local error by the difference** $e_{n+1} := z_{n+1} - y_{n+1}$.

When two methods use the same function evaluations like above, we say that we have an embedded pair of RK methods. The embedded pair above is called RK34. In practical use, one usually doesn't compute $z_{n+1}$, but one computes the error estimate directly from

\[
e_{n+1} := \frac{h}{6} (2Y'_2 + Z'_3 - 2Y'_3 - Y'_4).
\]

**Task 1.2**

Write a function
RK34step(f, uold, told, h) → unew, err

that takes a single step with the classical RK4 method and puts the result in unew and uses the embedded RK3 to compute a local error estimate in err as described above.

**Question 1.3**

Apply, by hand calculations, the RK4 and RK3 methods to the linear test equation $y' = \lambda y$. Derive the stability polynomials $P(h\lambda)$ and $Q(h\lambda)$ respectively. Find the difference $Q(h\lambda) - P(h\lambda)$, representing the local error estimate, and in particular, find its order, i.e., the power $k$ in $Q(h\lambda) - P(h\lambda) = O((h\lambda)^k)$, by analyzing your results.

**Theory**

Let us use the Euclidean norm throughout and assume that the local error $r_{n+1} := \|e_{n+1}\|_2$ is of the form

$$r_{n+1} = \varphi_n h_n^k$$

if the step size $h_n$ was used. The coefficient $\varphi_n$ is called the principal error function and depends on the method as well as on the problem.

Our goal is now to keep $r_n = \text{TOL}$ for a prescribed accuracy tolerance TOL. The idea is to vary the step size so that the local error stays of the same magnitude. In order to do this, we assume that $\varphi_n$ varies slowly (in other words, we will treat it as if it were “constant”).

Now suppose that $r_n$ was a bit off, i.e., there is a deviation between $r_n$ and the desired value TOL. How would we change the step size in order to eliminate this deviation? If $\varphi$ is constant we seek a step size $h_n$ such that

$$r_n = \varphi h_{n-1}^k$$

$$\text{TOL} = \varphi h_{n}^k.$$

Here we know the old step size $h_{n-1}$, the estimated error $r_n$ and the tolerance TOL. This implies that we can solve for the next step size $h_n$. Thus, by eliminating $\varphi$ from the equations above (just divide the equations) we find

$$h_n = \left(\frac{\text{TOL}}{r_n^k}\right)^{1/k} \cdot h_{n-1}.$$

This is the simplest recursion for controlling the step size and is the desired algorithm for making the RK34 method adaptive.

**Task 1.4**

Write a function

```python
newstep(tol, err, hold, k) → hnew
```
which, given the tolerance $\text{tol}$, a local error estimate $\text{err}$, the old step size $\text{hold}$, and the order $k$ of the error estimator, computes the new step size $\text{hnew}$.

**Task 1.5**

Combining RK34 and newstep, write an adaptive ODE solver

$$ \text{adaptiveRK34}(f, y_0, t_0, t_f, \text{tol}) \rightarrow t, y $$

which solves $y' = f(t, y); \quad y(t_0) = y_0$ on the interval $[t_0, t_f]$, while keeping the error estimate equal to $\text{tol}$ using the step size control algorithm you implemented above. In the vector $t$ you store the time points the method uses, and in $y$ you store the corresponding numerical approximation. See below how you need to arrange the function $f$.

**Task 1.6**

Use your solver to solve the van der Pol equation

$$y_1' = y_2$$
$$y_2' = \mu \cdot (1 - y_1^2) \cdot y_2 - y_1$$

with initial condition $y(0) = (1, 1)$. In order to do this, you have to write a function that computes the right-hand side of the ODE in the following format

$$ \text{vdp1}(t, y) \rightarrow \text{dydt} $$

where $\text{dydt}$ is the value computed from (♣).

Note that the argument $t$ is necessary, although it will not be used by your solver in this case. The reason is that we will also solve this and other problems with built-in solvers, and then the function is required to have this format. If you follow the format you will be able to reuse your function in the tasks that follow.

The parameter $\mu = 1$ is to be used for the time being; it will take other values later on.

The solution is oscillatory. Plot the solution over a few periods. Use various tolerances for your computations. Plot also $y_2$ as a function of $y_1$, the so-called phase portrait. Try using a few different initial values, and check that the solution tends to the same oscillation, a limit cycle, after a short time.

**Part 2. Stiff and nonstiff problems**

**Theory**

In the first few lectures, we saw both the explicit Euler method,

$$ y_{n+1} = y_n + hf(t_n, y_n) $$

and the implicit Euler method,

\[ y_{n+1} = y_n + hf(t_{n+1}, y_{n+1}) . \]

In order to understand the distinction between stiff and nonstiff initial value problems, one needs to understand the following question.

**Q:** If the two methods have similar accuracy, then why would one consider using an implicit method, in which every step is more expensive due to the necessary (nonlinear) equation solving?

**A:** It can pay off if one can take much longer steps with the implicit method.

This happens in **stiff problems**. All explicit methods have bounded stability regions, and this limits the maximum stable step size. A well designed implicit method, however, can have an unbounded stability region, meaning that there is no stability restriction on the step size. The method can take much larger steps and this makes up for the extra work required for the equation solving.

---

**Task 2.1**

Go back to the van der Pol equation. Solve it for \( \mu = 10 \) on the interval \([0, 10]\) with your own explicit RK34 code. Then go on to solve it for \( \mu = 100 \) on the interval \([0, 100]\), and, if at all possible, for \( \mu = 1000 \) and the range of integration \([0, 1000]\). What happens?

---

**Task 2.2**

Read the documentation in MATLAB (using `help` and other sources) on how to use `ode15s` to solve differential equations. Repeat the experiments from the last Task using `ode15s`. What happens? Which code performs better, and why? Compare, if necessary, step sizes used. Note that in order to obtain the step sizes from `ode15s`, which outputs the time points \( \{t_n\} \) in a vector \( t \), you can use the command, `diff(t)`, which produces the difference sequence, \( \{t_{n+1} - t_n\} \), which is identical to the step sizes used.

You may also read more about stiff differential equations at Cleve’s corner, see [http://www.mathworks.com/company/newsletters/news_notes/clevescorner/may03_cleve.html](http://www.mathworks.com/company/newsletters/news_notes/clevescorner/may03_cleve.html)

If you like, you can try other solvers in MATLAB too. They have the same calling sequence, so it does not take a lot of time to start working with another type of solver.

---

**Part 3. The Lotka–Volterra equations: an application**

**Theory**

A classical model in population dynamics is the Lotka–Volterra equation,

\[
\begin{align*}
\dot{x} &= ax - bxy \\
\dot{y} &= cxy - dy,
\end{align*}
\]
where $a, b, c, d$ are positive parameters. The equation is used to model the interaction of a predator species, $y$, with a prey, $x$, e.g. foxes and rabbits. If there are no foxes present ($y = 0$) the rabbits multiply and grow exponentially. On the other hand, if there are no rabbits ($x = 0$), then the foxes have no food supply and die at a rate determined by $d$. The product term, $xy$, represents the probability that a fox encounters a rabbit. This benefits the fox, so the product term is positive in the second (fox) equation. Because the fox eats the rabbit, it is negative for the rabbits, however, as shown by the product term entering the first (rabbit) equation with a negative sign.

The Lotka–Volterra equation is separable. By dividing the two equations, we get

\[
\frac{dx}{dy} = \frac{ax - bxy}{cxy - dy} = \frac{x(a - by)}{y(cx - d)}.
\]

Written in terms of differentials, we have

\[
(c - d)dx = \left(\frac{a}{y} - b\right)dy,
\]

and by integration we obtain

\[
cx - d\log x = a\log y - by + K.
\]

Hence the function $H(x, y) = cx + by - d\log x - a\log y$ remains constant at all times, i.e., $H(x, y)$ is invariant along solutions. This will have the consequence that the Lotka–Volterra equation has periodic solutions.

**Task 3.1**

Choose some suitable positive values for the parameters $a, b, c, d$, such as $(0.1, 0.3, 0.5, 0.5)$ and some suitable positive initial values, preferably not too far from the equilibrium position $(\frac{a}{c}, \frac{b}{d})$. Take a reasonably small step size and solve the problem using the explicit Euler method. What happens? Are the solutions periodic as claimed? Plot $x$ and $y$ as functions of time, and also plot $y$ as a function of $x$, i.e., the phase portrait. It is often easier to check periodicity there.

Can you find some possible explanation of your observation, such as what is the shortcoming of the method? Could you speculate what would happen if you instead used the implicit Euler method? (Note: do not try to implement the implicit Euler method, as you will have to implement nonlinear equation solving too.)

**Task 3.2**

Use your own adaptive RK34 to solve the problem. Run with a tolerance of (say) $10^{-4}$. Simulate the system for at least 10 full periods. In what way is the solution different now? Can you conclude that your method is better? How do you interpret a periodic solution biologically?

If you change the initial conditions, does the solution go to the same periodic solution? Compare this to what happened in the van der Pol oscillator, which had a limit cycle. Does the Lotka–Volterra equation have a limit cycle?
Task 3.3

The model is idealized, inasmuch as the rabbits are supposed to increase exponentially in the absence of foxes. Modify the problem to also include a “logistic” factor, representing that food supply is an issue also for the rabbits, by replacing the first equation by

\[ \dot{x} = ax(1 - x/10) - bxy. \]

(This implies that we force \( x \leq 10 \); the number 10 has been chosen quite arbitrarily, but ought to be a parameter in the equation, just like \( a, b, c \) and \( d \).) Does this make the solution qualitatively different? Are the solutions periodic? Are they stable? What happens after a “long time” (i.e., as \( t \to \infty \))? If you check the derivations demonstrating the separability of the Lotka–Volterra equation, can you still use that trick when the logistic factor is present?

Note concerning the documentation of all tasks

- Reports are individual.
- The written report must not exceed 10 pages, including plots and codes.
- The report will be graded; your performance counts as part of the final exam.
- Be sure to identify every plot with a title, axis labels and a caption explaining what you see in the plot and what conclusions you can draw from it.
- Don’t forget to acknowledge discussions with fellow students or instructors in your report.
- Summarize what you have learned in this computer project.
- Please feel free to also give positive as well as negative feedback on the project for future use in the course.