ENGSCI 233 - Lab Worksheet

Iteration and Stability

Date: April 5, 2022

Lab Introduction

Your objective in this lab is to implement explicit Runge-Kutta (RK) methods for the numerical solution of ordinary differential equations (ODEs), and then use these to explore the population dynamics of a fishery under different management strategies.

The following files have been made available to you:

- *functions_itlab.py*: partially completed RK functions.
- *test_itlab.py*: partially completed tests for your functions.
- *fisheries_itlab.py*: a script file in which to solve the fisheries scenario.

Any functions that you complete in *functions_itlab.py* should include a suitable docstring. Note that test cases, like those in *test_itlab.py*, will generally be self-documenting (i.e. the test name explains its purpose) or can be documented very briefly.

Task 1 - Improved Euler and Classic RK4 Methods

Background

You will be implementing both the Improved Euler and Classic RK4 methods. These are both explicit RK methods and therefore share a largely similar approach to iterating the numerical solution of an initial value problem. What differs is how each individual step is taken.

You will need to complete three functions for this task and test them:

1. `def step_ieuler`

   This function will need to be completed in the lab. It performs a single step of the Improved Euler method. The governing equations for this method are:

$$\begin{aligned} y^{(k+1)} &= y^{(k)} + h\left(\frac{f_0}{2} + \frac{f_1}{2}\right) \\ f_0 &= f\left(t^{(k)}, y^{(k)}\right) \\ f_1 &= f\left(t^{(k)} + h, y^{(k)} + hf_0\right) \end{aligned}$$

2. `def step_rk4`

   This function will need to be completed in the lab. It performs a single step of the Classic RK4 method. The governing equations for this method are:

$$\begin{aligned} y^{(k+1)} &= y^{(k)} + h\left(\frac{f_0 + 2f_1 + 2f_2 + f_3}{6}\right) \\ f_0 &= f\left(t^{(k)}, y^{(k)}\right) \\ f_1 &= f\left(t^{(k)} + \frac{h}{2}, y^{(k)} + \frac{hf_0}{2}\right) \\ f_2 &= f\left(t^{(k)} + \frac{h}{2}, y^{(k)} + \frac{hf_1}{2}\right) \\ f_3 &= f\left(t^{(k)} + h, y^{(k)} + hf_2\right) \end{aligned}$$

3. `def solve_explicit_rk`

   This function will need to be completed in the lab. We can generalise any explicit RK method to a form:

$$y^{(k+1)} = y^{(k)} + h\bar{f}^{(k)}$$

   where $\bar{f}$ is the weighted derivative across the step taken, which is dependent on the chosen RK method. However, our general stepping approach is independent of the chosen RK method, so a single function can be used.

   This solver will use the time span and time step to determine in advance how many iterations are required. A similar approach to that used in Python's built-in ODE solver, `scipy.integrate.solve_ivp`, can be used to select the RK method with which to step i.e. use a string argument to select the method.

In general, we will be passing a *function* (which itself returns the ODE derivative) as an argument to our RK functions. For example, we can pass to `def step_ieuler` the following function for an ODE derivative (simply by using the function name):

```
def dydt1(t, y):
    return y - np.sin(t)
```

In order to call this from within `def step_ieuler`, we would need to provide the two arguments, t and y. However, ideally we want our RK functions to handle a wide variety of different ODEs. For example, lets consider how to call an ODE derivative that requires additional parameters from within `def step_ieuler`:

```
def dydt2(t, y, a, b):
    return -np.sin(a*t+b)
```

This can be achieved through packing and unpacking arguments in Python. The two extra parameters, a and b, can be packaged together into a single list and provided as an argument to the function:

```
t = 0.
y = 0.
args = [2., 1.]
derivative = dydt2(t, y, *args)
```

This will call the function with $a = 2$ and $b = 1$ i.e. the $*$ will unpackage the list to provide additional arguments to the called function. A similar structure can also be used to call our first ODE derivative function, but now with an empty list (since we don't require any additional parameters):

```
t = 0.
y = 0.
args = []
derivative = dydt1(t, y, *args)
```

We can therefore generalise all of our calls to ODE derivative functions from within our RK functions to be of the form:

```
derivative = dydt(t, y, *args)
```

where `args` is a list of length equal to the number of additional parameters (i.e. beyond just the independent and dependent variable) that must be passed to the derivative function. The `args` can itself simply be passed into our RK functions as a list argument.

Exercises

Complete the following exercises for this task:

1. Perform one step of the Improved Euler and Classic RK4 methods by hand for the following initial value problem and a time step of $h = 2$:

   $$\frac{dy}{dt} = t - y \;, \qquad y\left(0\right) = 1$$

   Use this result to write two simple test cases `def test_step_ieuler` and `def test_step_rk4` in *test_itlab.py*. They should be designed to test a single step of your Improved Euler and RK4 step functions.

2. Complete `def step_ieuler` and `def step_rk4` in *functions_itlab.py*. Use your test cases from the previous step to check that both are working.

3. It may be somewhat time-consuming to construct a test case for the iterative solver. Another option to test our function is to compare our numerical results to the exact solution. For example, the initial value problem:

   $$\frac{dy}{dt} = \cos\left(t\right) \;, \qquad y\left(0\right) = 0$$

   has an exact solution of $y = \sin\left(t\right)$. We can use a statistic like the *mean absolute error* to measure the error in our numerical result:

   $$\frac{1}{n}\sum_{i=0}^{n-1} \left|y_i - \hat{y}_i\right| < \epsilon \tag{1}$$

   where y_i and $\hat{y}_i$ are the numerical and exact solutions at each independent variable point evaluated, and n is the number of points. This metric has the same units as the original data and can be compared against some desired error tolerance, ϵ (not machine epsilon, that would be too strict for an answer we expect to include a reasonable amount of truncation error).

   Write a test case `def test_solve_explicit_rk` that uses mean absolute error to test the iterative solver. You will need to choose a sensible value for ϵ.

4. Complete `def solve_explicit_rk` and use your test case from the previous step to check that it is working.

Task 2: Fisheries management

Background

In New Zealand, the Ministry for Primary Industries is responsible for managing our fisheries. At present, they operate a **Quota** system, where a fixed number of fish may be taken from a population each year. This can be expressed as:

$$\frac{dn}{dt} = rn\left(1 - \frac{n}{k}\right) - f_0$$

where n is the fish population, t is the time (in years), r is a constant representing the birth rate, k is a constant representing the carrying capacity (i.e. maximum sustainable population), and f_0 is the annual take of fish (i.e. the quota). A criticism of this approach is that the annual take is *insensitive* to the health of the fishery - even if the population is in decline, the quota remains the same.

The iwi authority of Ngāi Tahu, whose tribal boundaries cover much of the South Island, have proposed an alternative management strategy, based on the principle of **Kaitiakitanga** (duty of care, guardianship). A fixed annual *quota* is permitted, providing it *does not exceed* a fixed fraction of the population, otherwise it is reduced to that *fixed fraction*. For example, if the nominal quota is 10,000 fish, but the annual take cannot exceed 13% of the population then, for a population of 50,000 fish, the actual quota is capped at 0.13*50,000 = 6,500. However, if the population recovers to 100,000, then 0.13*100,000 = 13,000. This exceeds the nominal quota of 10,000 so the annual take for that year will be 10,000.

Ngāti Whātua, whose tribal lands extend throughout Auckland and Northland, have proposed yet another strategy. The kaitiakitanga approach to fishery management relies on there being a good estimate of the current fish population, something that can be difficult to measure. Instead, the Ngāti Whātua authority have proposed a system of **Periodic Rāhui** (a tapu that restricts access), in which fishing is permitted in an area for X years, but then all fishing operations must move out of the area for X years. Afterward, fishing may resume for another X years, before a second rāhui is imposed, etc.

Your objective in this task is to **solve** for the fish population over time under these three different management strategies: Quota, Kaitiakitanga, and Rāhui. The main focus is on the *ecological health of the fishery*.

Exercises

Complete the following exercises in this task:

1. Write a derivative function for each of the three management scenarios in *functions_itlab.py*. These are `def dndt_quota`, `def dndt_kaitiakitanga` and `def dndt_rahui`.

   Note that `def dndt_kaitiakitanga` will require an additional parameter that controls what maximum fraction of the total population can be taken, and `def dndt_rahui` will require an additional parameter that controls the duration of the rāhui.

   You may **optionally** want to write test cases in *test_itlab.py* to check that these functions are working as expected.

2. Let us assume that for a given fisheries region, the carrying capacity is 1 million fish, the initial population is 750,000 fish, the nominal quota is 130,000 fish, and the birth rate is 0.5. A healthy fishery is one that has a long-term (e.g. over 50 years) population that is at or above half the carrying capacity.

   In *fisheries_itlab.py*, investigate if the quota system will lead to a healthy fish population over a 50 year period. In addition, find both a kaitiakitanga and rāhui management strategy that will result in a healthy fishery population, while still maintaining a good level of fish caught over this time period.

   It is advised that you solve the system using the Classic RK4 method with a step size of around $h = 1$.

3. In *fisheries_itlab.py*, plot all three scenarios (i.e. the quota scenario, your choice of kaitiakitanga, and your choice of rāhui) from the previous exercise on a single plot for comparison over a 50 year timespan. Ensure that the plot is suitably labelled and includes a legend to distinguish the three scenarios.

4. Answer the following questions relating to this task:

   - What impact does each strategy have on the long-term fishery health?
   - How does changing the duration of the rāhui affect both the short-term and long-term fish population?
   - How sensitive are your final solutions to the step size chosen? You may want to solve one of the management strategies with a few different step

sizes and compare the results, similar to a convergence analysis.

Submission Instructions

For this lab, you should submit the following:

- *functions_itlab.py*
- *test_itlab.py*
- *fisheries_itlab.py*
- A PDF or Word file containing your answers to the Task 2 questions
- Your Task 2 plot named *task2.png*

You do **not** need to submit any other files. Do **not** modify or vary the names of the functions file (but don't worry if Canvas appends a `-1` or `-2` to the end).

Remember, all submissions are compared against each other and those from previous years. Copying someone else's code and changing the variable names constitutes academic misconduct by *both* parties.