Numerical ODES approximation

March 2022

1 Introduction

In this document we reflect the development process of the ODE class done in python in which, methods for each of the following numerical schemes are implemented:

1. Forward Euler
2. Backward Euler
3. Trapezoidal
4. Runge-Kutta of second order
5. Runge-Kutta of fourth order
6. Adams-Bashforth of second order
7. Adams-Bashforth of fourth order
8. Leapfrog method
9. Adams-Moulton of second order
10. Adams-Moulton of fourth order
11. Backwards-differences of second order
12. Backwards-differences of fourth order

Where the items from 1 to 5 are one step methods and 6 to 12 are linear multi-step methods. From the linear multi-step methods, 6 to 8 are explicit methods, and 9 to 12 are implicit methods.

2 Class attributes

First we must recall that we are considering initial value problems (IVP) of the form u'= f(t, u(t)) In the constructor method, the given fields are required as keyword arguments:

- The function *self.__f* received as a string and then later converted into a lambda function through the lambdify method in the sympy library.
- The initial condition self.__icon, passed as a two-element tuple.
- The final time *self.__tf*, the approximations will be computed from the first element of the *icon* tuple to *self.__tf*.
- The separation in between points *self.__h*
- The exact solution *self.__e*, which is passed as a string and then converted into a lambda function with the sympy package.

Also in the constructor we define an attribute for each of the methods referred to in the previous section, which consist of an instance of the class T. This class has two different attributes one called vals, and another one called todate. This is done in order to handle changes in the separation between points or in the initial value which would entail that the previously computed values are invalid for all the methods and the exact array of values needs to be recomputed as well since either the first element is different or the array length must change.

3 Class methods

As stated in the introduction, there is a method for all the 12 numerical schemes mentioned. There is also a method that given a method, with the condition the instance has will return the global error.

In order to generalize this more, we could use the function *odeint* from scipy, but since it solves the IVP by numerical approximations, it is more interesting to manually set in the exact solution.