1. Review of Numerical Methods

We have developed multiple numeric methods for approximately solving ODEs. So far, they include

- Forward Euler Method
- Trapezoidal Method
- Midpoint Method

These methods all are explicit (only information at time step \( n-1 \) is necessary for computing the solution at time step \( n \)). We have examined the methods in a few ways.

- Accuracy/Consistency: a method’s accuracy is a function of its Local Truncation Error. The local truncation error is computed by applying the method’s difference operator on the true solution. For Forward Euler, the local truncation error is \( O(h) \) (linear in the time step). The other two methods are \( O(h^2) \).
- Convergence: A method is convergent of order \( p \) if it is consistent of order \( p \) (i.e. has a local truncation error of order \( p \)) and is 0-stable. If a method is 0-stable, then we know that the order of the global error is the same as the order of the local truncation error.
- Region of absolute stability: This is the first analysis that involves the ODE itself. Each method has a region of absolute stability that depends on the step size and on the relative speed of the system. For systems with faster dynamics, we need smaller time steps. The region of absolute stability is the region in which the approximate solution is qualitatively correct. It says nothing about the size of the error.

In these examinations, we have encountered two types of errors: the local truncation error and the global error (which is a function of the local truncation error). But there is a third type of error: the local error. Here is a description of each type of error for a method of order \( p \) (for Forward Euler \( p = 1 \) and for trapezoidal and midpoint \( p = 2 \)).

- Local truncation error:
  - It’s size: \( O(h^p) \)
  - What it means: This is the error introduced at a time step. Sort of. You need to divide the introduced error by \( h \) to get the local truncation error.
– How to compute it: Apply the difference operator to the true solution.
– What it means for Forward Euler. We take the $O(h^2)$ terms we chopped off of the Taylor series and divide it by $h$.
– What Wikipedia calls it in the Forward Euler article: It isn’t there.

• Global error:
  – It’s size: $O(h^p)$
  – What it means: This is what you think of when you think of error. How well does the method do?
  – How to compute it: Compare the true solution at time step $n$ to the approximated time solution at time step $n$. From the discussion of 0-stability, we also know that it is a function of the local truncation error.
  – What it means for Forward Euler. Same thing. This one is self explanatory.
  – What Wikipedia calls it in the Forward Euler article: global truncation error.

• Local error:
  – It’s size: $O(h^{p+1})$
  – What it means: This is the error introduced at a time step. mostly.
  – How to compute it: Ok, this is “how to approximate it”. Use the method to approximate the solution at time step $n$. Compare it to the approximate solution of a higher order method.
  – What it means for Forward Euler. We take the $O(h^2)$ terms we chopped off of the Taylor series.
  – What Wikipedia calls it in the Forward Euler article: local truncation error
2. Error and step-size control

Let’s control the error by controlling the time step size \( h \).

For each time step, we compute a solution, then use a higher order method to estimate the local error. If the error is small enough, then we “accept” the solution at that value of \( h \). If it isn’t small enough, we compute a smaller \( h \) and try again. If the error was small enough, we compute a new, larger \( h \), so our next step can be further away.

Here is the algorithm:

- Initialize the arrays/matrices that will contain the output: \( t \) and \( y \) (and sizes, if you want an array that will keep track of the sizes)
- Initialize \( h \). I arbitrarily use \( h = (t_{\text{Final}} - t_0)/100 \);
- Loop until you reach \( t_{\text{Final}} \)
  - Compute \( \hat{y} \) using the forward Euler method.
  - Compute \( \hat{y}_H \) using the explicit trapezoidal method.
  - Estimate the error \( \text{err} = \text{abs}(\hat{y}_H - \hat{y}) \);
  - If the error is smaller than the given tolerance
    * Update the arrays to include this step.
    * If this time step puts you at or past \( t_{\text{Final}} \), then break out of the loop
  - Compute the new step size and update the step size variable \( h \).
    \[ h = \min(h \times (0.9 \times \text{EST/\text{err}})^{1/2}, h \times 2) \]