1 LTE of RK Methods

2 Absolute Stability of Runge-Kutta Methods

In order to plot the region of absolute stability for Runge-Kutta methods, we notice that for the method

$$y_{n+1} = p(h\lambda)y_n$$

to be stable, we require

$$|p(\kappa)| < 1$$

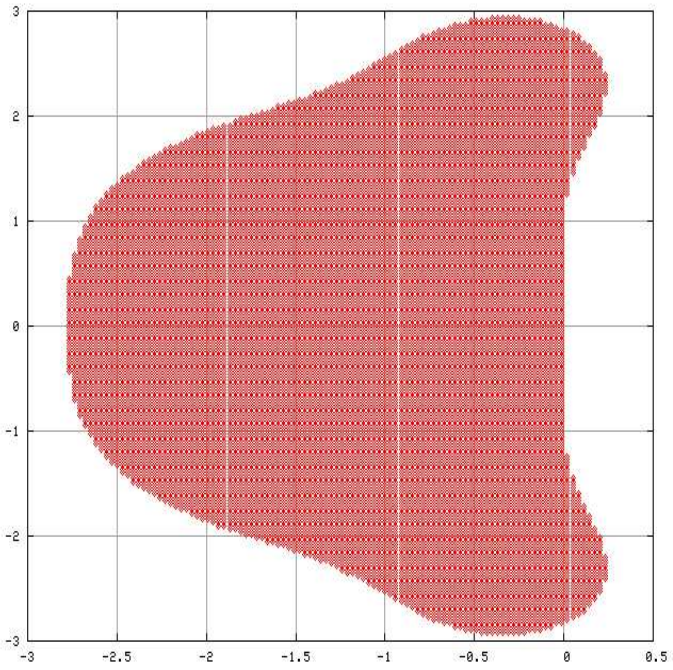
For some polynomial $p(h\lambda)$. For Runge-Kutta methods, the polynomial $p(\kappa)$ is defined as

$$p(\kappa) = \sum_{i=0}^{k} \frac{1}{i!}\kappa^i$$

Where $\kappa = h\lambda$ is complex and k is the stage of the method. For $k = 4$, we get

$$p(\kappa) = 1 + \kappa + \frac{1}{2}\kappa^2 + \frac{1}{6}\kappa^3 + \frac{1}{24}\kappa^4$$

We have written a routine in Python that scans through the complex plane and plots a point where $|p(\kappa)| < 1$. (Code available on request.) The routine produces this figure



3 Ritz Method

4 Finite Differences

5 Local Truncation Error of the Crank Nicolson Method

For the PDE $u_t = u_{xx}$, the Crank-Nicolson Method is defined as :

$$(1+r)U_m^{n+1} - \frac{r}{2}(U_{m+1}^{n+1} + U_{m-1}^{n+1}) = (1-r)U_m^n + \frac{r}{2}(U_{m+1}^n + U_{m-1}^n)$$

Where $r = k/h^2$. To calculate the local truncation error, we set this equal to zero and replace the approximate values with exact values.Then we take the Taylor Expansion, expanding about $u(x_m, t_n)$

$$(1+r)\left(u + ku_t + \frac{1}{2}k^2u_{tt} + \frac{1}{6}k^3u_{ttt}\right) \tag{1}$$

$$-\frac{r}{2}\left(u + hu_x + ku_t + \frac{1}{2}h^2u_{xx} + hku_{xt} + \frac{1}{2}k^2u_{tt} + \frac{1}{2}h^2ku_{xxt} + \frac{1}{2}hk^2u_{xtt}\right) \tag{2}$$

$$-\frac{r}{2}\left(u - hu_x + ku_t + \frac{1}{2}h^2u_{xx} - hku_{xt} + \frac{1}{2}k^2u_{tt} + \frac{1}{2}h^2ku_{xxt} - \frac{1}{2}hk^2u_{xtt}\right) \tag{3}$$

$$-(1-r)u \tag{4}$$

$$-\frac{r}{2}\left(u + hu_x + \frac{1}{2}h^2u_{xx} + \frac{1}{6}h^3u_{xxx}\right) \tag{5}$$

$$-\frac{r}{2}\left(u - hu_x + \frac{1}{2}h^2u_{xx} - \frac{1}{6}h^3u_{xxx}\right) \tag{6}$$

If we cancel, and collect terms we have

$$\begin{aligned}
& u_t\,(k + rk - rk) \\
+ \quad & u_{tt}\left(\frac{1}{2}k^2 + \frac{1}{2}rk^2 - \frac{1}{2}rk^2\right) \\
+ \quad & u_{xx}\left(-\frac{1}{2}rh^2 - \frac{1}{2}rh^2\right) \\
+ \quad & u_{xxt}\left(-\frac{1}{2}rh^2k\right) \\
+ \quad & u_{ttt}\left(\frac{1}{6}k^3 + \frac{1}{6}rk^3\right)
\end{aligned}$$

After applying the facts $u_t = u_{xx}$ and $r = k/h^2$, we are left with

$$u_{ttt}\left(\frac{1}{6}k^3(1 + \frac{k}{h^2})\right)$$

Notice that all the terms in (2),(3),(5),(6) canceled. We need to look at more terms to get a handle on the error. Since all terms in (2) and (3) with odd powers of h will cancel, we will only look at those with even powers. Then for the Taylor expansion we get

$$(1+r)\frac{1}{6}k^3u_{ttt} \tag{7}$$

$$- \quad r\left(\frac{1}{6}k^3u_{ttt} + \frac{1}{24}h^4u_{xxxx} + \frac{1}{4}k^2h^2u_{xxtt}\right) \tag{8}$$

$$- \quad r\frac{1}{24}h^4u_{xxxx} \tag{9}$$

Then we apply the identity $u_{ttt} = u_{xxtt}$ and $r = k/h^2$ to get

$$\frac{-1}{12}\left(k^3u_{ttt} + kh^2u_{xxxx}\right) + H.O.T.$$

Which gives us an LTE of $O(k^3 + kh^2)$.

6 Stab of Crank Nicolson

7 Stab of Douglas

8 More on the Douglas Formula

To show that the Douglas formula does not come from applying a LMM to the method of lines, we look at the one stage LMM

$$y_{n+1} - y_n = h(\beta_0 f_n + \beta_1 f_{n+1})$$

If we translate this to the method of lines where we advance in time, we get

$$U_m^{n+1} - U_m^n = \frac{k}{h^2}\left[\beta_0\left(U_{m-1}^n - 2U_m^n + U_{m+1}^n\right) + \beta_1\left(U_{m-1}^{n+1} - 2U_m^{n+1} + U_{m+1}^{n+1}\right)\right] \tag{10}$$

Keep in mind that the factor $1/h^2$ comes from the difference approximation. It is clear that if we pick $\beta_0 = \beta_1 = 1/2$, we get the Crank-Nicolson formula. What we have to show is that, for the Douglas formula, there are no β values that are independent of $r = k/h^2$. The Douglas formula is given as

$$U_m^{n+1} - \frac{1}{2}\left(r - \frac{1}{6}\right)\left(U_{m-1}^{n+1} - 2U_m^{n+1} + U_{m+1}^{n+1}\right) = U_m^n + \frac{1}{2}\left(r + \frac{1}{6}\right)\left(U_{m-1}^n - 2U_m^n + U_{m+1}^n\right) \tag{11}$$

If we rewrite this so that it looks more like the LMM version, we get

$$U_m^{n+1} - U_m^n = \frac{1}{2}\left(r + \frac{1}{6}\right)\left(U_{m-1}^n - 2U_m^n + U_{m+1}^n\right) + \frac{1}{2}\left(r - \frac{1}{6}\right)\left(U_{m-1}^{n+1} - 2U_m^{n+1} + U_{m+1}^{n+1}\right) \tag{12}$$

We could again rewrite this to

$$U_m^{n+1} - U_m^n = \frac{1}{2}\left[\left(r + \frac{1}{6}\right)\left(U_{m-1}^n - 2U_m^n + U_{m+1}^n\right) + \left(r - \frac{1}{6}\right)\left(U_{m-1}^{n+1} - 2U_m^{n+1} + U_{m+1}^{n+1}\right)\right] \tag{13}$$

This would mean that $\beta_0 = r + 1/6$ and $\beta_1 = r - 1/6$, since none of the β are independent of r, the Douglas formula cannot be thought of as a LMM applied to the method of lines.

9 Numerical Comparison

Here we are to solve

$$u_t = u_{xx}$$

subject to the initial value condition

$$u(x, 0) = \sin x \quad \text{for} \quad x \in [0, \pi]$$

and the boundary conditions

$$u(0, t) = u(\pi, t) = 0 \quad \text{for all} \quad t \geq 0$$

Using the Douglas formula and the Crank-Nicolson method. We accomplished this using Python along with the NumArray library to solve the linear systems (code available upon request). We have prepared a table that lists the number of time steps n, the theoretical solution (given by $u(x, t) = e^{-t} \sin x$), the value for each method followed by the error at the midpoint.

N	True Solution	Crank-Nicolson Douglas	Error
1	0.99449791563	0.994509174613 0.994497915643	1.12589828036e-05 1.36470834633e-11
2	0.989026104192	0.989048498389 0.989026104219	2.23941966252e-05 2.71438427291e-11
4	0.978172634773	0.978216932165 0.978172634827	4.42973915895e-05 5.369227285e-11
8	0.956821703419	0.956908366374 0.956821703524	8.6662954748e-05 1.05040198761e-10
16	0.915507772134	0.915673621636 0.915507772335	0.000165849502438 2.01009653367e-10
80	0.643146895793	0.643729655705 0.643146896499	0.000582759912498 7.06048997046e-10
160	0.413637929568	0.414387869634 0.413637930476	0.000749940066547 9.0818685905e-10
320	0.171096336777	0.1717173065 0.171096337528	0.000620969722957 7.51321088677e-10
640	0.0292739564585	0.0294868333516 0.0292739567156	0.000212876893091 2.57096528095e-10
800	0.0121088187398	0.0122189860548 0.0121088188727	0.000110167315077 1.32931096686e-10

We need to mention here that getting the values at the midpoint was a little tricky. When we discritize in x, we get a list of function values for each of these points. For this problem, this means a function value for $x_m = 0, 1, 2, \ldots, 20$. Since we are given the values along the boundary, we can throw out the conditions on the end, giving us values $x_m = 1, 2, \ldots, 19$. When we calculate the true solution in the middle, we set $x = 10h$, since 10 is the middle value. However, when we look at the middle value in our array, this corresponds to position 9. This gets a little tricky at times. It is good to bear in mind that we should really look for the maximum error on this interval and that looking just at the middle value is laziness on our part.

It is also interesting to note is how well the Douglas formula performs. By applying methods based solely on the PDE, we gain an increase in accuracy on the order of 10^{-7}. This is a significant increase above the Crank Nicolson method. Since both methods are equally difficult/easy (depending on your point of view) to implement, there is no reason to use the Crank Nicolson method.

10 Shooting

The main point of Shooting methods is to take a boundary value problem, which is typically difficult to solve, and replace it with an initial value problem, which is easier to solve. In our case, we will solve the boundary value problem

$$y'' = y \qquad y(-1) = y(1) = 1$$

Which we then turn into the initial value problem

$$y'' = y \quad y(-1) = 1 \quad y'(-1) = \eta$$

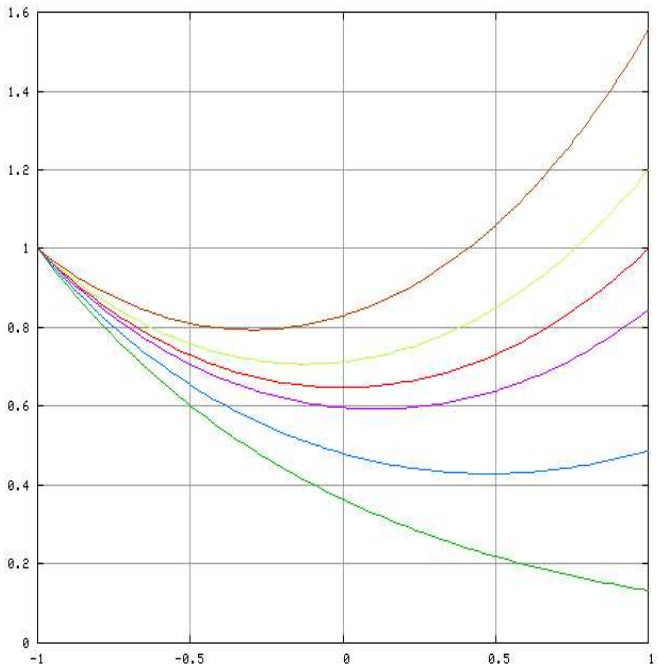
Where η is unknown. We have an implementation (in Python) that will compute the solution of this IVP using Euler's method (with $h = 1/50$), as a function of η. In order to find a good value for η, we define the function

$$F(\eta) = y(1, \eta) - 1$$

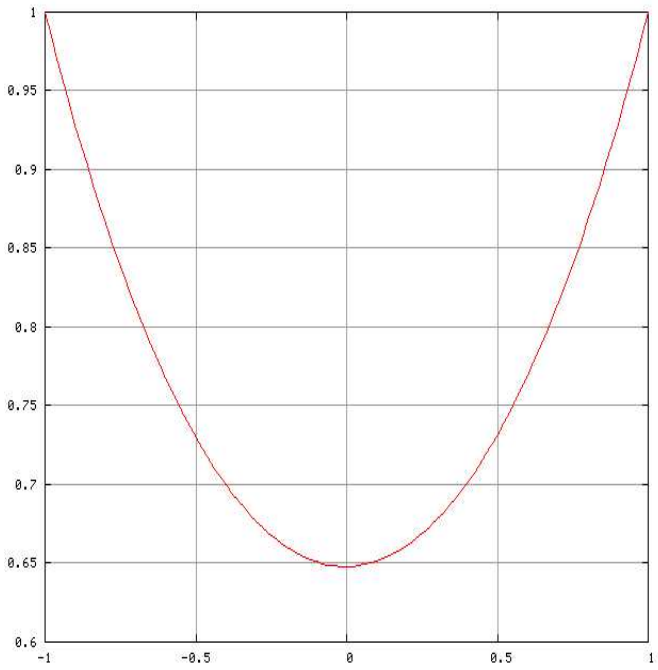
This equation is the difference between the solution we want and the solution we get using Euler's method. Then we want to solve for 0 using bisection method. (Remembering that $y(1, \eta)$ is found using our implementation of Euler's method.) For bisection, we need to find two values that bracket the root of our function $F(\eta)$. We do this, more or less, by hand. To begin with, we iterate over the interval $[-1, 1]$ by steps of 0.1 and get the values

η	$F(\eta)$
-1.0	-0.867380444105
-0.9	-0.511779115987
-0.8	-0.156177787869
-0.7	0.199423540248
-0.6	0.555024868366
-0.5	0.910626196484
-0.4	1.2662275246
-0.3	1.62182885272
-0.2	1.97743018084
-0.1	2.33303150896
0	2.68863283707
0.1	3.04423416519
0.2	3.39983549331
0.3	3.75543682143
0.4	4.11103814955
0.5	4.46663947766
0.6	4.82224080578
0.7	5.1778421339
0.8	5.53344346202
0.9	5.88904479013

To illustrate how shooting works, we take a set of solutions, say the ones for $\eta = -1, -0.9, \ldots, -0.5$ and plot them along with the numerical solution.



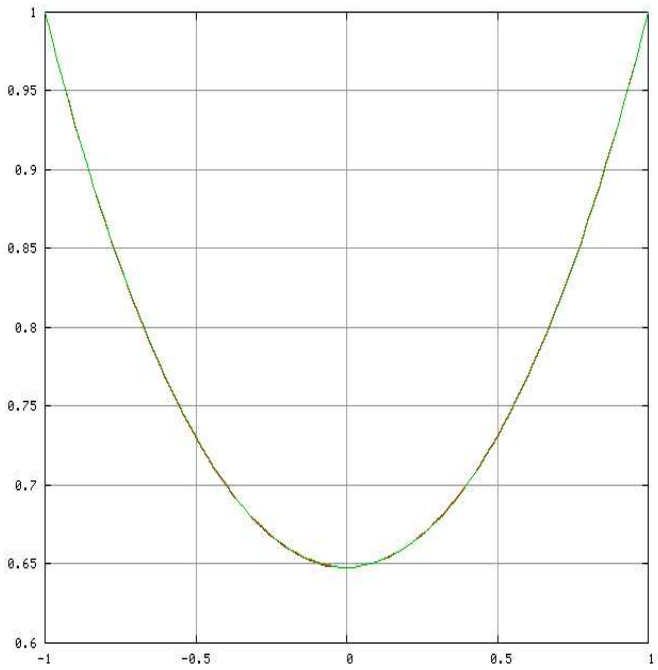
We can use the values in our table to determine a bracket for bisection. Using SciPy's bisection method on the bracket $-0.8, -0.7$ we, find the approximate value of $\eta = -0.756080651134$. If we plot the solution given by Euler's method by itself, using $h = 1/50$, we get this plot



The true solution is given by

$$y(x) = \frac{e^{x+1} + e^{1-x}}{e^2 + 1}$$

If we plot the true solution along with our approximated solution, we get



There is virtually no difference between the true solution and the approximated one. It is interesting to note that we are essentially relying on the accuracy of both Euler's method and the bisection method, therefore, we don't have to do any extra error analysis as our results are as good as the methods we choose. For example, we could use more accurate methods or different methods altogether. (Code available upon request.)