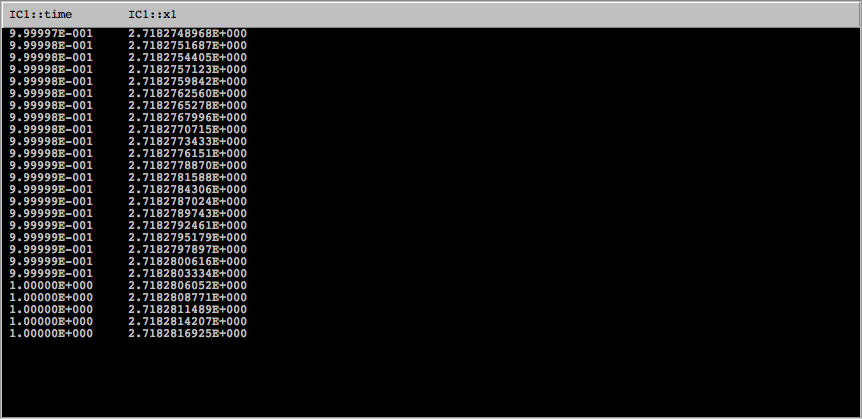
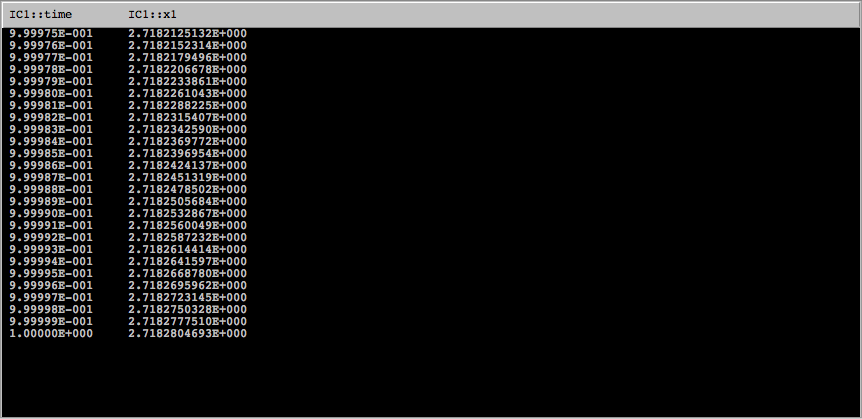
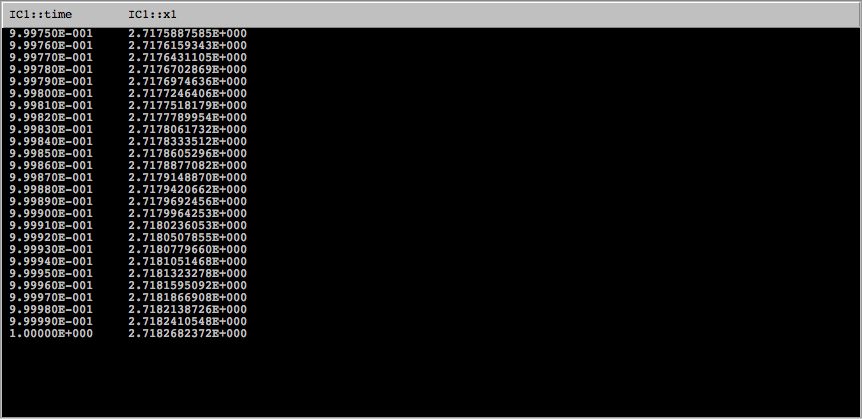
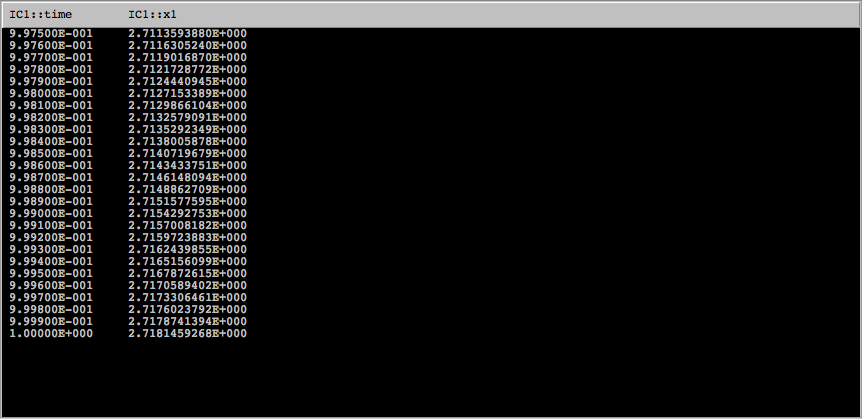
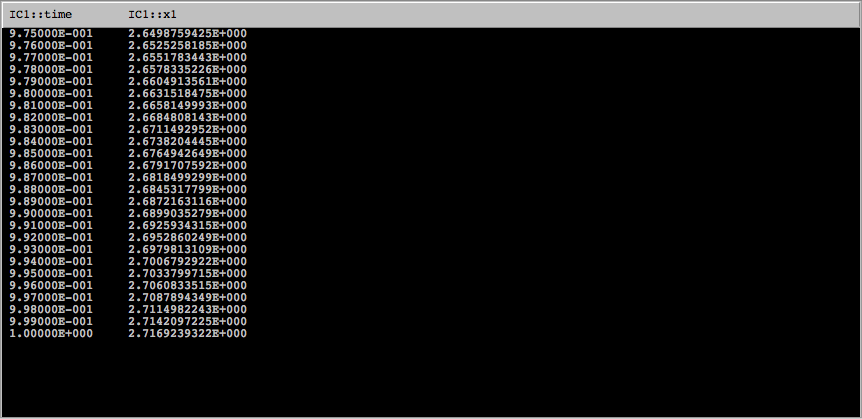
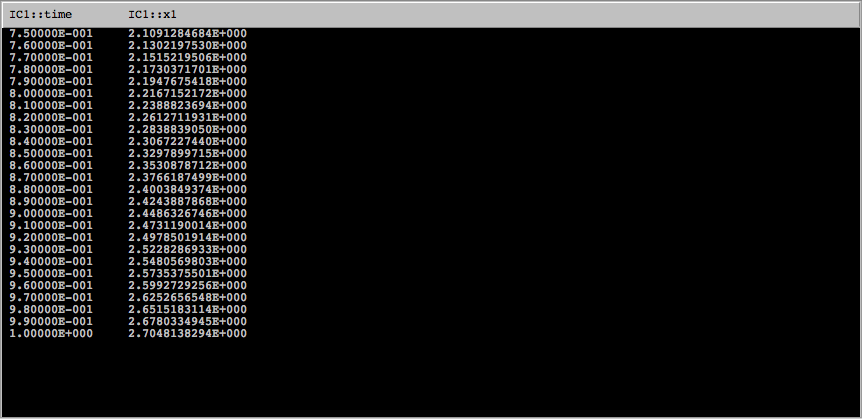
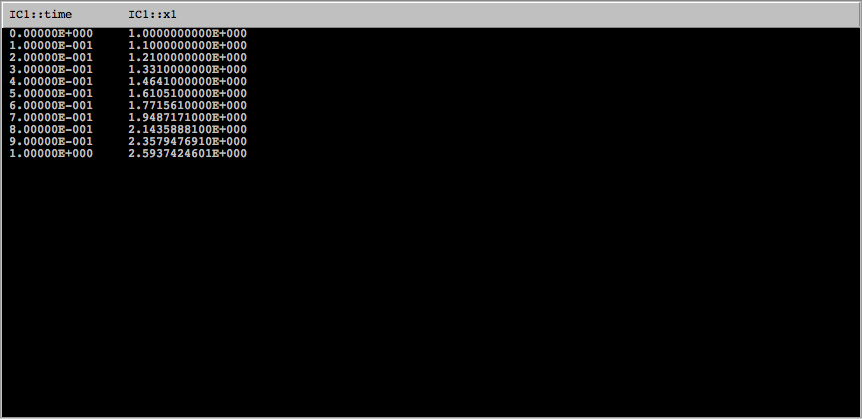
Hannah Machado

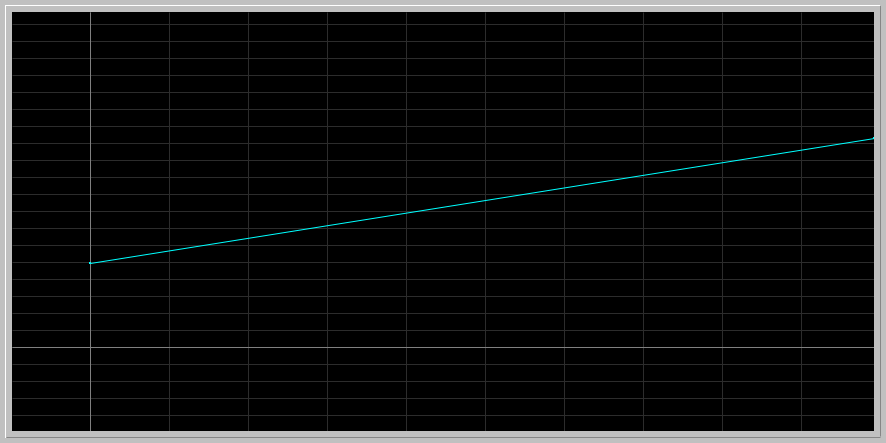
CSC 210

March 10th, 2017

*Homework 6*

1.**Error vs. step size in Euler:** We showed in class that the global error bound in Euler's algorithm is proportional to the step size. Now, in PHASER solve the initial-value problem x' = x, x(0) = 1 to compute x(1) = e = 2.7182818284590452354, using Euler's algorithm with seven different step sizes. Using your favorite plotting program, plot the errors against the step sizes. Do you get a linear relationship? Be careful of the scales on your graph.

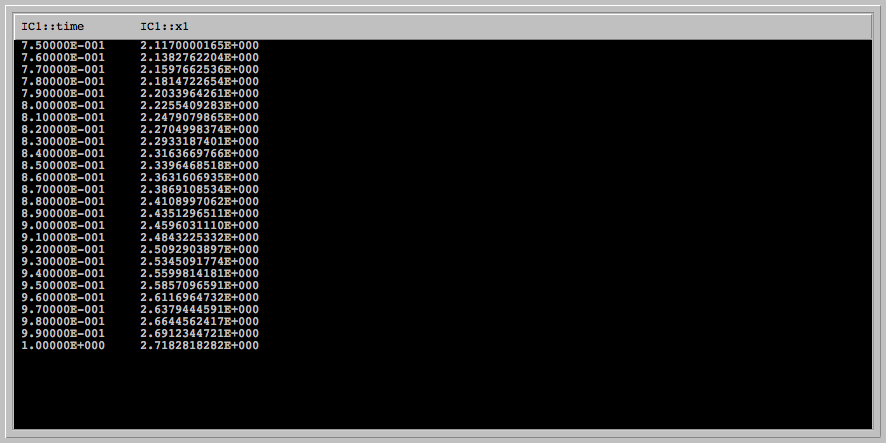
The graphs of h=0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001, 0.0000001



Yes, the plot of the errors against the step sizes is linear and the slope is positive, meaning that as the step size increases, the error increases, and vice versa.

2.**Orders of algorithms:** Use PHASER for solving the initial-value problem x' = x, x(0) = 1 at t=1, that is x(1)= e = 2.7182818284590452354 with several algorithms as follows. You should open the Console in the Numerics Editor to get the stats for your computations:

* First use Runge-Kutta with step h = 0.01. Note that you get about 9 digits correct to the right of the decimal. How long does your computation take?

Xi Values -> Run # 1

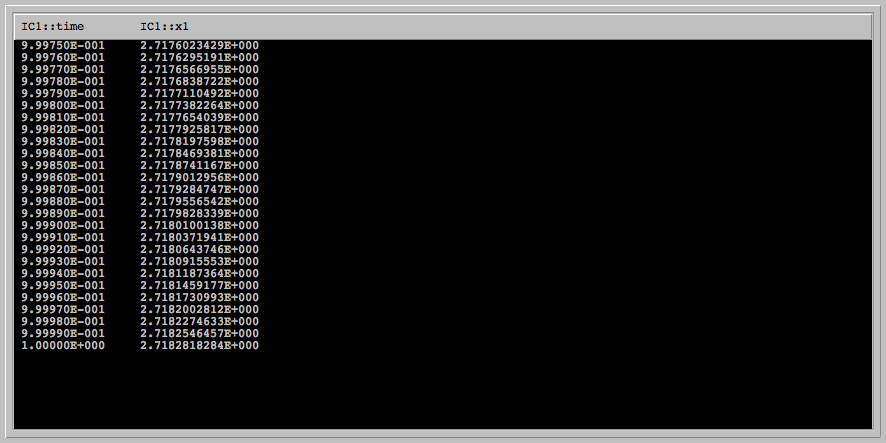
Algorithm [Runge-Kutta (4)] Stats:

# Function Evaluations... = 400

# Computed Steps......... = 100

View Runtime: 0.0050 seconds

* Now determine the "largest" step size in the Improved Euler algorithm to get the same number of correct digits. How long does it take to compute?

The “largest” step size to obtain the same amount of correct digits was h=0.00001

Xi Values -> Run # 1

Algorithm [Improved Euler (2)] Stats:

# Function Evaluations... = 200000

# Computed Steps......... = 100000

View Runtime: 0.015 seconds

* Explain your results in terms of the orders of the algorithms you use.

Using the Runge-Kutta algorithm, the global error bound is proportional to h^4 because it is a fourth order algorithm. Because of that, it achieves a bigger precision with a smaller number of steps. The time taken to get the 9 digits correct to the right of the decimal was 0.005 seconds. Using Improved Euler algorithm, however, the step size had to be decreased to 0.00001 in order to achieve the same precision. The global error bound in improved Euler is proportional to h^2, it is a second order algorithm, and therefore takes more computations in order to get a more precise value.

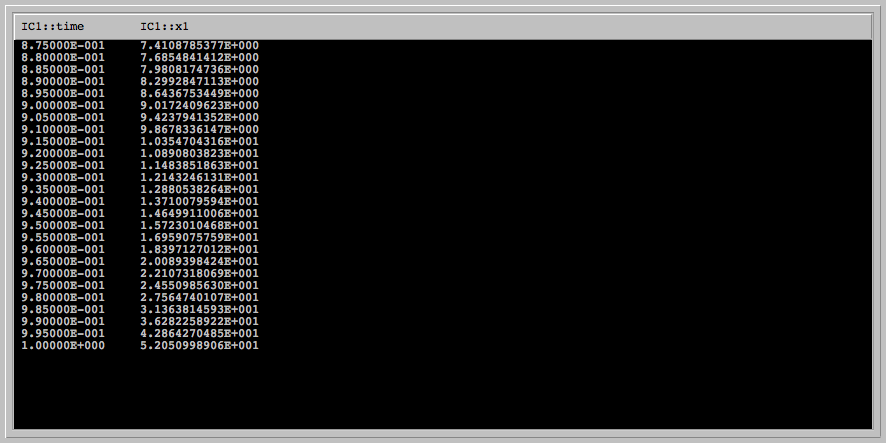
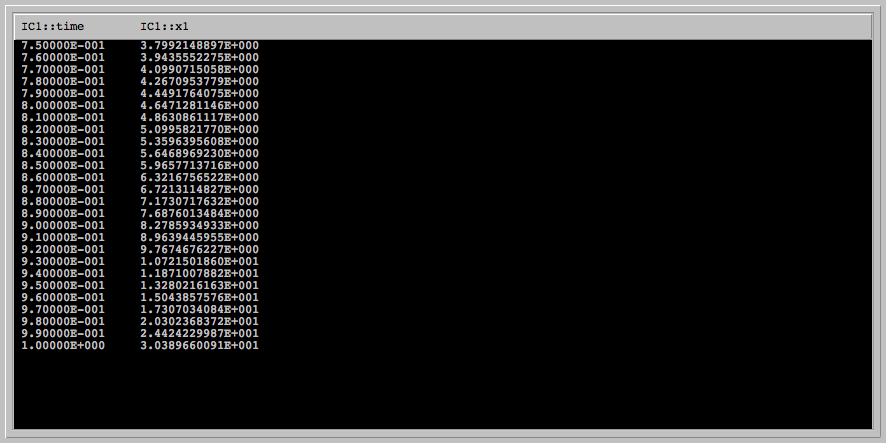
**3.Impossible computation:** Consider the "explosion" problem x'= x^2, x(0)=1, and try to compute x(1) using several different algorithms with various step sizes.

* Verify by differentiation that the unique solution of this initial value problem is x(t) = 1/(1-t). Conclude that x(1) is infinite.

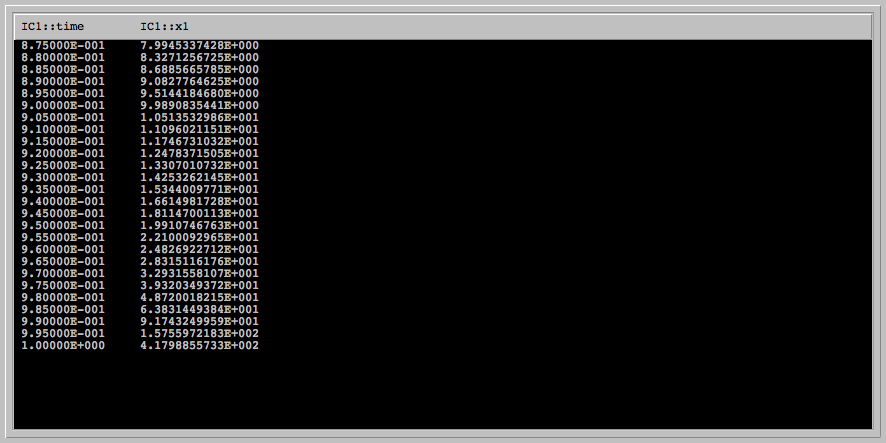
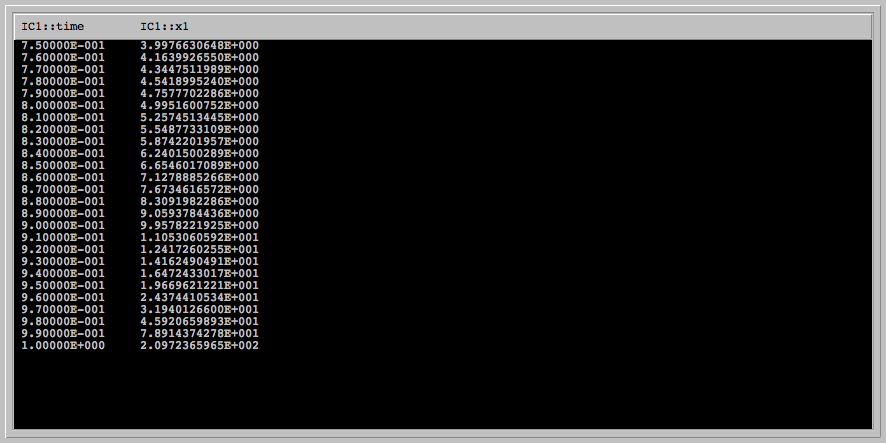
x(1) is infinite since the value x(1)= 1/1-1= 1/0= infinity

* Now in Phaser, compute this number, x(1), using steps h = 0.01 and h = 0.005 and algorithms Euler, Improved Euler, Runge-Kutta4, Dormand-Prince5(4), and Dormand-Prince8(5,3).

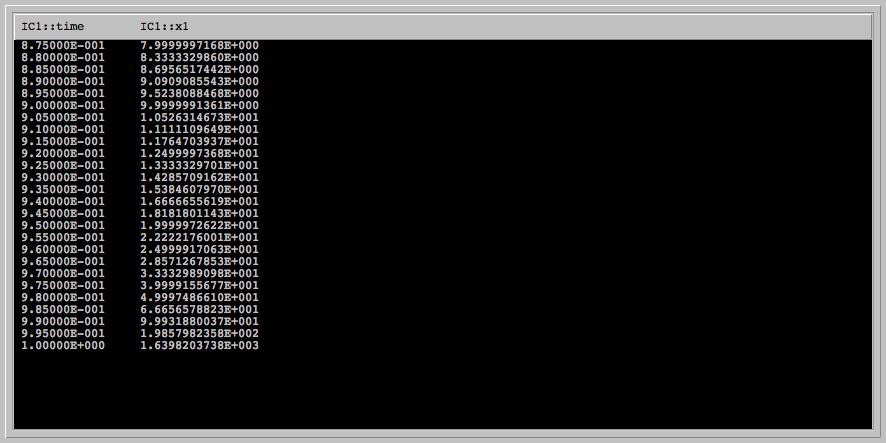
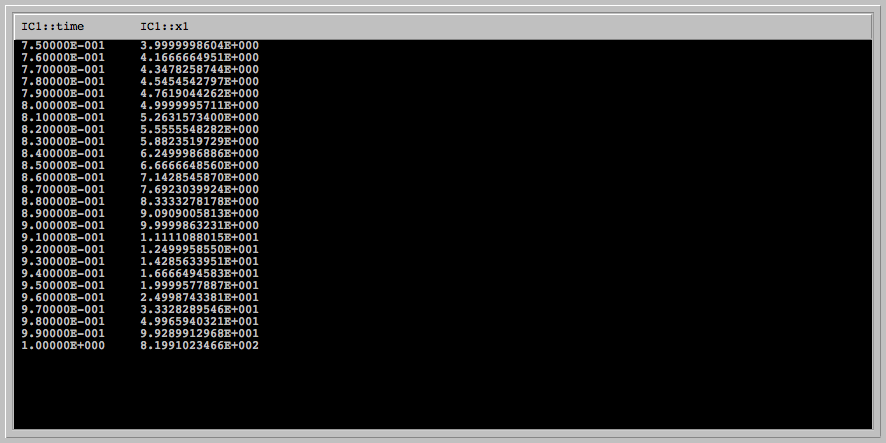
Euler using h=0.01 and h=0.005



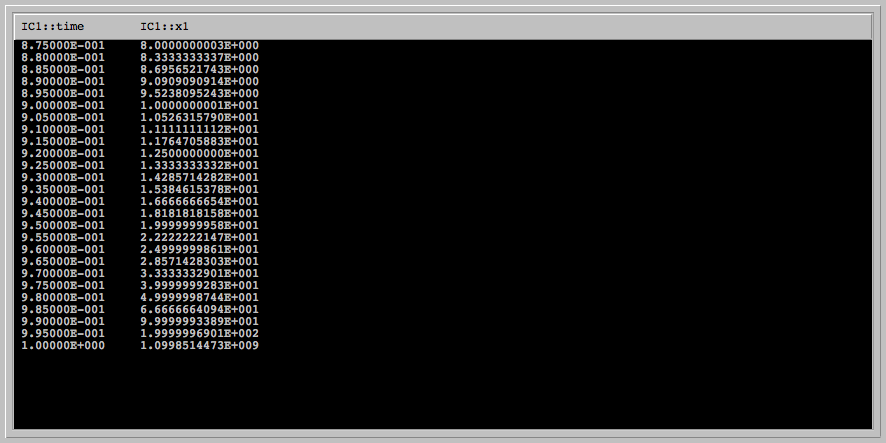
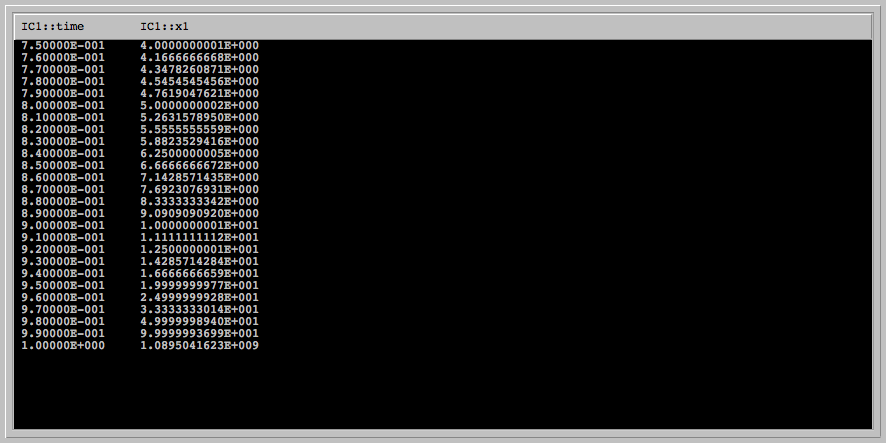
Improved Euler using h=0.01 and h=0.005



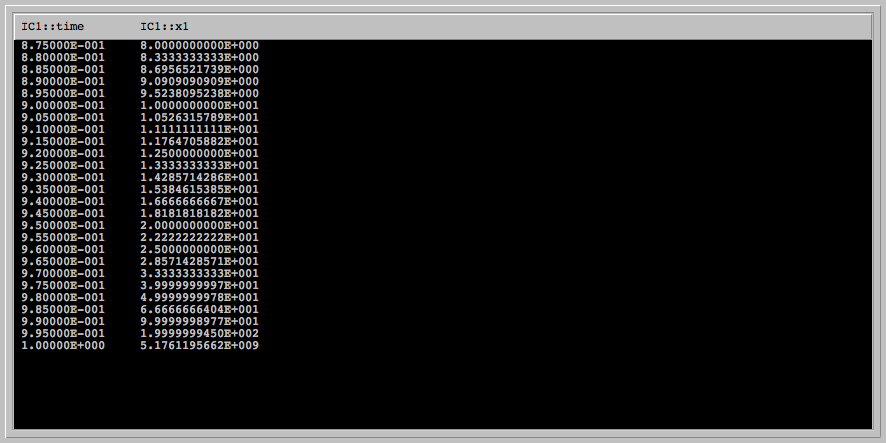
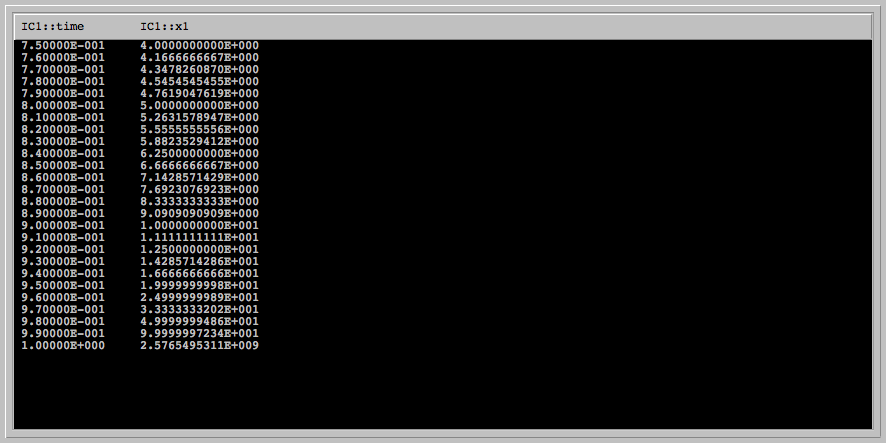
Runge-kutta(4) using h=0.01 and h=0.005



Dormand-Prince5(4) using h=0.01 and h=0.005



Dormand-Prince8(5,3) using h=0.01 and h=0.005



* Following "What to do in practice" rule above, conclude that this problem is indeed dangerous to compute.

This problem is indeed dangerous to compute because all of the algorithms give very different results, indicating that there is an abnormal behavior at x(1). Therefore, there is no guarantee that the chosen algorithm will be the most accurate one. Since we don’t know the function itself, only the derivative, it is hard to know which algorithm is the most reliable. In this case, the solution would end up being infinity, since 1/1-1= 1/0= infinity. There are numbers ranging from 52.05 to 5.17\*10^9 in the answers depending on which algorithm is used, and that is why a lot of algorithms should be tested and varied before coming to a certain conclusion. If they all give an answer with an intolerable difference, then that may give you a clue that the problem is too difficult to compute, and in the real world some degree of certainty is required for success.

* Next, In the Dormand-Prince 5(4), set the step h = 0.01 and decrease the Absolute error and Relative error from 1.0E-7 until Phaser refuses to compute the value x(1). What is the smallest error tolerance for which PHASER refuses to compute infinity? Monitoring the Console in the Numerics Editor, report at which t value PHASER stops computations for your set tolerances.

When the Absolute error and Relative error are decreased to 1.0E-9, Phaser refuses to compute infinity and stops the computation at t= 9.90000E-001

Xi Values -> IC Set 1: Maximum number of steps ( 500 ) exceeded at time 0.9999999998753417!

Xi Values -> Run # 33

Algorithm [Dormand-Prince 5(4)] Stats:

# Function Evaluations... = 4874

# Computed Steps......... = 779

# Accepted Steps......... = 775

# Rejected Steps......... = 0

View Runtime: 0.0050 seconds

**4.One step of Runge-Kutta (4):** Consider our canonical example of x' = x, x(0) = 1. By hand, compute one step of Runge-Kutta (4) algorithm with step size of h = 1. Compare your answer to the one you obtain from Phaser. Note: The formulas for Runge-Kutta (4) are available in PhaserHelp, or the link above.

yn+1 = yn + ( K1 + 2K2 + 2K3 + K4) / 6 ,

where

K1 = h f( tn, yn ),

K2 = h f( tn + (1/2) h, yn + (1/2) K1 ),

K3 = h f( tn + (1/2) h, yn + (1/2) K2 ),

K4 = h f( tn + h, yn + K3 ).

K1= 1\*f(xn)= 1\*1=1

K2= 1\*f(xn +K1)= 1\*2=2

K3= 1\*f(xn +K2)= 1\*3=3

K4= 1\*f(xn +K2)= 1\*4=3

yn+1= 1 + (1+2+3+4)/6= 2.66666667

Value computed in Phaser:

0.00000E+000 1.0000000000E+000

1.00000E+000 2.7083333333E+000

5.**Reading Euler:** Try to read the original paper of Euler listed above. Write a short commentary on this paper. Is his algorithm the same as the one we derived in class? What does he have to say about errors?

The algorithm in the paper is slightly different from the one we derived in class, but more the format. In the paper, the Euler is stated as b’ = b + A(a’ - a), while the one derived in class is Xn+1= xn +hf(x). They are structurally different but essentially the same. The errors, however, work in the same way. The paper explains that when smaller intervals are taken, more accurate values are obtained, but the errors accumulate because of the multitude, so the more steps you take, the more accumulated error you are going to have, and that needs to be taken into account.