Math 151B HW 1 Project Report

Justin JR Wang

4/11/14

User Guide

odesolver.m : This function implements the solver functions in this program, Euler's method, Runge-Kutta method of order 4,the Adams fourth order predictor-corrector method, and finally, the implicit 3-step Adams-Moulton method. It takes in all the arguments that make up an initial value problem, then passes them to the actual solver functions. For the Adams-fourth order predictor-corrector method, and the implicit 3-step Adams-Moulton method, the for loops to update the solution vectors are in odesolver because we specify the endpoint b in odesolver, which lets us know n, which ultimately gives us the size of the vectors.

The calling syntax is: `function [ yt ] = odesolver( a, b, y0, h, f, option )`

Inputs:

```
a is the initial time
b is the final time
y0 is the initial condition
h is the step size by default, matlab array
indicies start from 1, be
aware when defining the size of yt using h
f is the function handle f(t,y) by default
option can be strings 'euler', 'rk4', 'adams4',
'am3'
```

Outputs:

- yt is the solution to the initial value problem. It is a function f with respect to t, y(t).

euler.m : This function implements the Euler's method for solving initial value problems. The algorithm is given on page 267 in Burden and Faires, Algorithm 5.1. Euler's method is an explicit one-step method.

Euler's method is the most elementary approximation technique for solving initial-value problems.

The calling syntax is : `function [ wc ] = euler( t, w, h, f )`

Inputs:

- t is the preceding time when we have computed the approximation of the solution
- w is the approximation of the solution at time
- h is the step size.

- f is the function that defines the IVP.

In all cases, f is the function handle which defines the f(t, y), in the IVP of the usual standard format:

$$\begin{cases} y'(t) = f(t, y(t)), & a \le t \le b \\ y(a) = \alpha \end{cases}$$

Output:
- wc is the approximation w to y at the (N+1) values of t.

rk4.m : This functions implements the Runge-Kutta method of order four, the procedure is explained in the textbook in Algorithm 5.2. Like Euler's method, Runge-Kutta methods are explicit one-step methods.

The calling syntax is : `function [ wc ] = rk4( ti, wi, h, f )`

Inputs:
- ti is the preceding time when we have computed the approximation of the solution
- wi is the approximation of the solution at time
- h is the step size.
- f is the function that defines the IVP.

Output:
- wc is the approximation w to y at the (N+1) values of t.

am3.m : This function implements the implicit 3-step Adams-Moulton method. The 3 step method uses information from the two previous

time steps to update the next iteration. The calling syntax is:

```
function [ wc ] = am3( t, w, h, f, a2)
```

Inputs:

- t is the preceding time when we have computed the approximation of the solution
- w is the approximation of the solution at time
- h is the step size.
- f is the function that defines the IVP.

Output:

- wc is the approximation w to y at the (N+1) values of t.

am3.m : This function implements the implicit 3-step Adams-Moulton method. The 3 step method uses information from the two previous time steps to update the next iteration. So am3 is an implicit, multistep method . The calling syntax is: `function [ wc ] = am3( t, w, h, f, a2)`

Inputs:

```
updates wi using implicit 3-step Adams-Moulton
method
t is a vector storing ti, ti-1, ti-2
w is a vector storing wi, wi-1, wi-2
a is the coefficient vector associated to wi+1,
wi, wi-1, wi-2
```

```
varargin is an optional input which can store
the derivative function
F'(w) if Newton's method is used
since implicit method, apply root-finding method
(secant) to find wi+1
```

Output:
- wc is the approximation w to y at the (N+1) values of t.

adams4.m : This function implements the adams fourth order predictor-corrector method. The method first uses the fourth-order Adams-Bashforth method as predictor and one iteration of the Adams-Moulton method as corrector, with the starting values obtained from the fourth-order Runge-Kutta method
Hence, adams4 is an explicit and implicit multistep method.
The calling syntax is: `function [ wc ] = adams4( t, w, h, f, a1, a2 )`

Inputs:

```
part of Algorithm 5.4, explicit & implicit
multistep
updates wi using the Adams fourth order
predictor-corrector method
t is a vector which stores the four preceding
times ti, ti-1, ti-2, ti-3
to generate wi+1
w is a vector storing wi, wi-1, wi-2, wi-3
a1 is the coefficient vector associated to wi,
wi-1, wi-2, wi-3 in the
```

```
prediction step, ie [55, -59, 37, -9]./24, which
can be defined in
odesolver.m
a2 is the coefficient vector associated to wi+1,
wi, wi-1, wi-2 in the
correction step, ie [9, 19, -5, 1]./24, which
can be defined in
odesolver.m
wc is the generated wi+1, which is a scalar in
first-order ODEs but a
vector in systems of ODEs, use vector case for
more general code
```

Output:
- wc is the approximation w to y at the (N+1) values of t.

secant.m: This function finds the root of the equation F(wi+1)=0 in the implicit multistep methods. This method is not as good (i.e. does not converge) as Newton's method for finding the roots of a root solving problem; however, we prefer to use this secant method because it does not require that we find the derivative function F'(w).
The secant method is described in Algorithm 2.4 in the textbook.
Note: A root solver such as secant is required to complete the solutions to the implicit methods because of the way they are designed. The w(i+1) iteration is the root that we are trying to solve for.

```
The calling syntax is :  function [ p ] =
secant( p0, p1, f, tol, itermax )
```

Input:

p0 and p1 are the initial approximations
tol is the error tolerance
itermax is the maximum number of iterations

Output:
p is the root of the root finding problem equation f.

OPTIONAL:

```
function [ p ] = newton( p0, f, df, tol, itermax)
If we are able to find the explicit expression of F', consider using
Newton's method, Algorithm 2.3, for faster convergence
This function also determines if a root is found, and the number of
iterations actually used.
```

Input:

p0 is the initial approximations
f is the root finding function
df is the derivative of the root finding function, in this case it's F'(w)
tol is the error tolerance
itermax is the maximum number of iterations

Output:
p is the root of the root finding problem equation f.

main.m solves the specified IVPS in solutions, exercise
ivp1.m
5.6.2d, and exercise

```
5.11.2b with h=.1, and h=2, ivp2.m  It returns
the graph of y(t) vs w(t), and the graph of the
global error.
```

Implementation:

The tests implemented to validate all this code are exercises 5.6.2d and exercise 5.11.2b with h=.1, .2 in the textbook. The functions are called ivp1.m, and ivp2.m in the zip file, respectively. The results are output as a total of 24 graphs in main, since the Runge-Kutta method of order 4 graphs are also included. While working on the code, I also used sample exercises in the textbook for Euler and RK4 and checked that the charts matched. Also, the same for the secant root solver.

At first, I had trouble knowing how to code adams4.m, since the way it is wanted specified in this project, is slightly different than the way it's described in Algorithm 5.4 of the textbook, because we do some parts in odesolver.m, and other parts in the adams4.m. This is because we don't specify the number of iterations, N. We are able to solve for N; however, in odesolver, because b is an input parameter. Actually, the algorithm outlined in the textbook had to be modified in various ways for matlab. For one thing, there are some steps in the end that have to do with updating things that are not necessary because we are not using a programming language like Maple.

One way to define stiffness is: A problem is stiff if Euler's method has to use a step h smaller than needed to resolve the solution accurately. If we look at the graphs, this would be the case for the second exercise 5.11.2b, ivp2.m, where the first set of graphs uses h=.1, and the second set of graphs uses h=.2. Therefore, if the first set of graphs is better, and the second one got worse, then the IVP is likely to be stiff. The conclusions suggest that some methods, such as Euler's method, is no good for solving stiff equations. In fact, Euler's method is not really good for anything, it's just a nice elementary technique that has a fairly intuitive derivation. Also,

the Runge-Kutta method of order 4, which is also an explicit method like Euler's method, is not good for stiff, however, it's still a good general method that can be adaptive. The implicit methods are better in general. For instance, the trapezoidal method (Adams-Moulton of order 2) is great for solving stiff equations! The Adams-Moulton implicit 3 step method, am3.m, has better stability (although not A-stable), and is better for stiff.

Solutions:

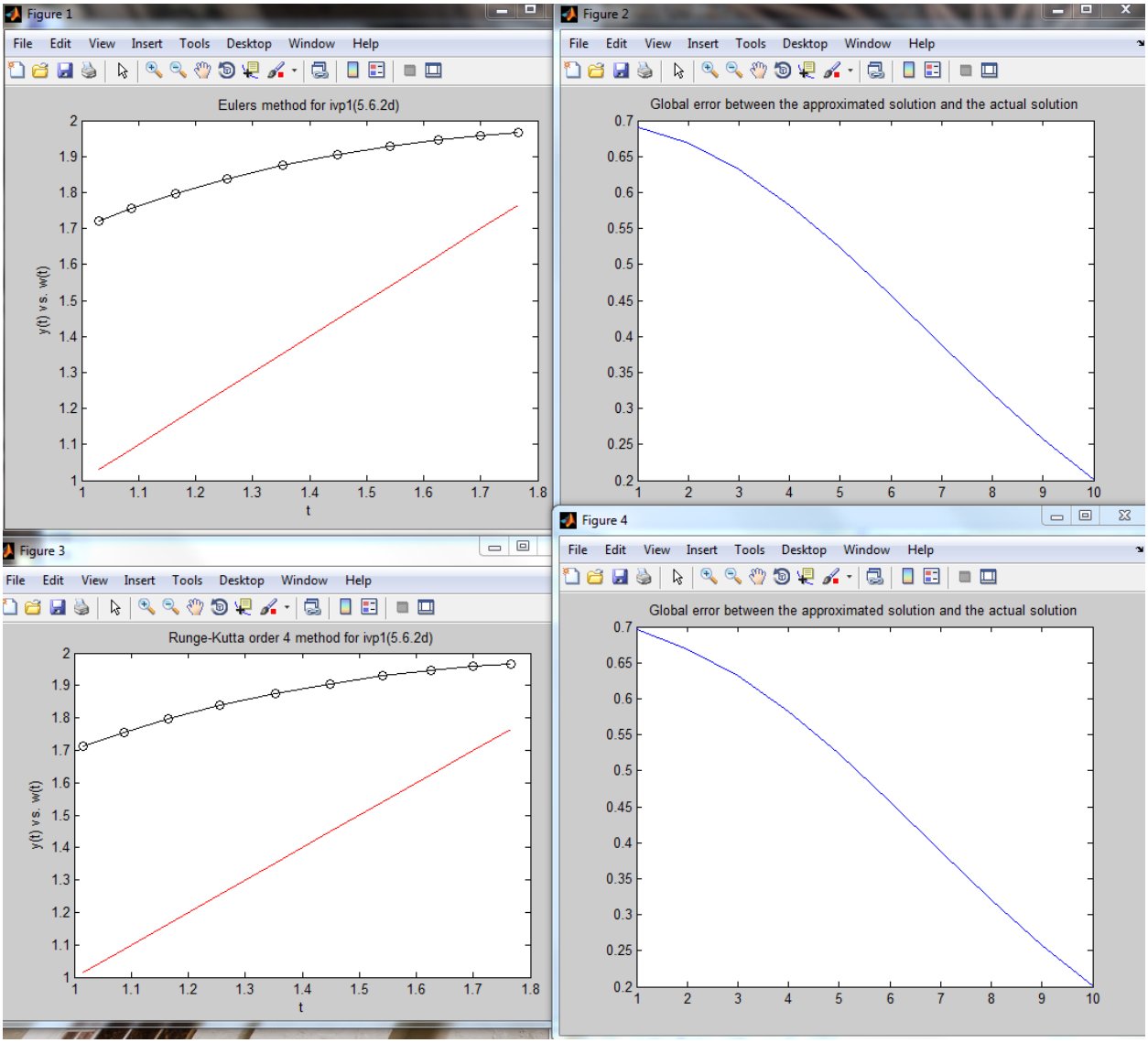
Running main.m will output 24 graphs, starting with exercise 5.6.2d, ivp1.m, which will output 8 graphs, the first 2 are Euler's method, second 2 are RK(4), then the implicit 3-step Adams-Moulton method, and finally adams4.m, which is the Adams fourth order predictor-corrector method, (ABM). ABM predictor-corrector is good for high accuracy.

First we'll discuss the solutions to the first exercise, 5.6.2d in terms of the global error between the approximated solution and the actual solution:

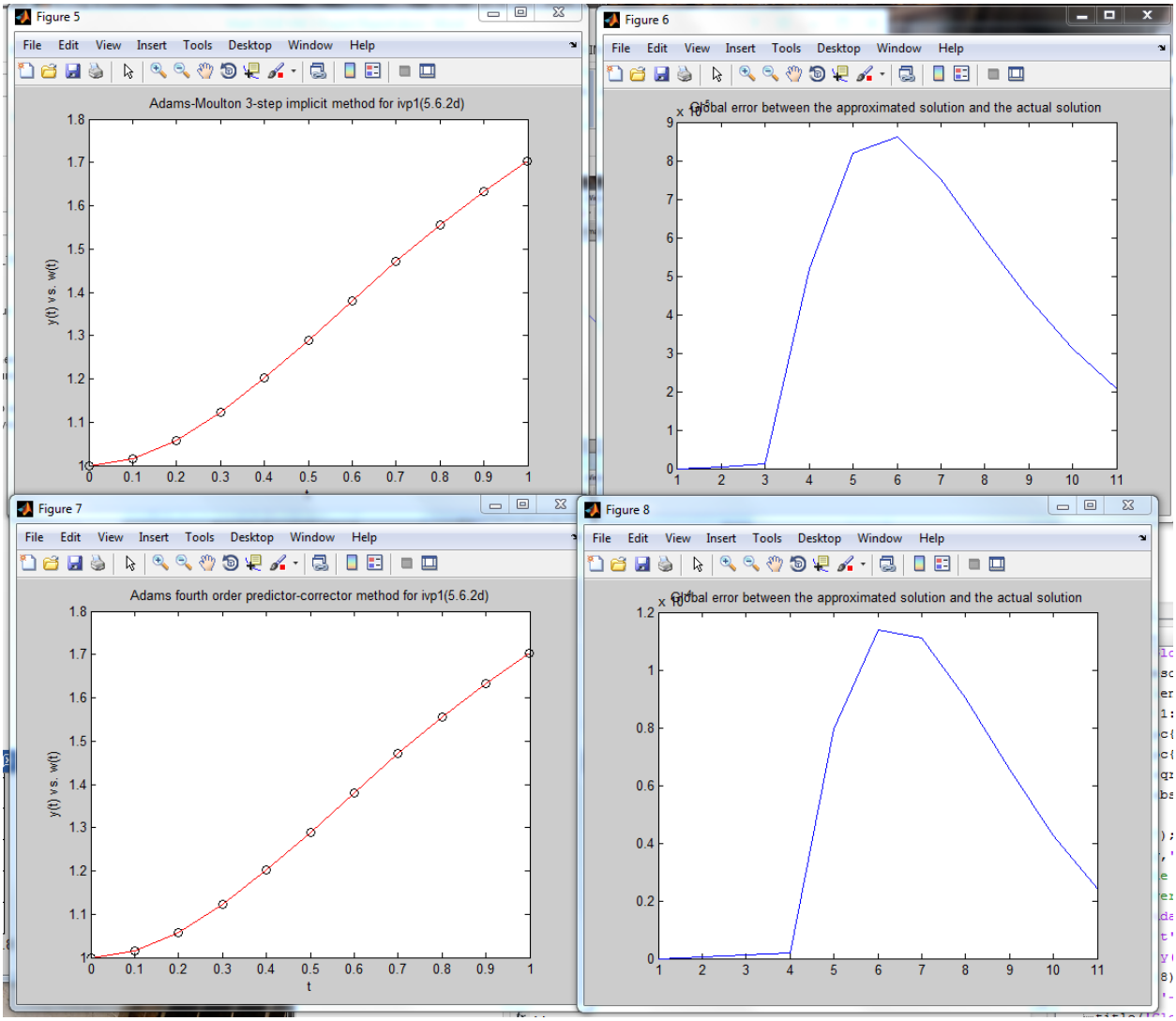
$$., \text{ i.e., } \max_{1 \le i \le N} |y(t_i) - w_i|.$$

We know that the first IVP is not stiff, so the behavior of the various methods will be different than in the second one, which is stiff.

Here are the graphs: Figures (1-8)



For this first method, it is hard to see the difference between the two explicit methods, Euler's method and Runge-Kutta method of order 4

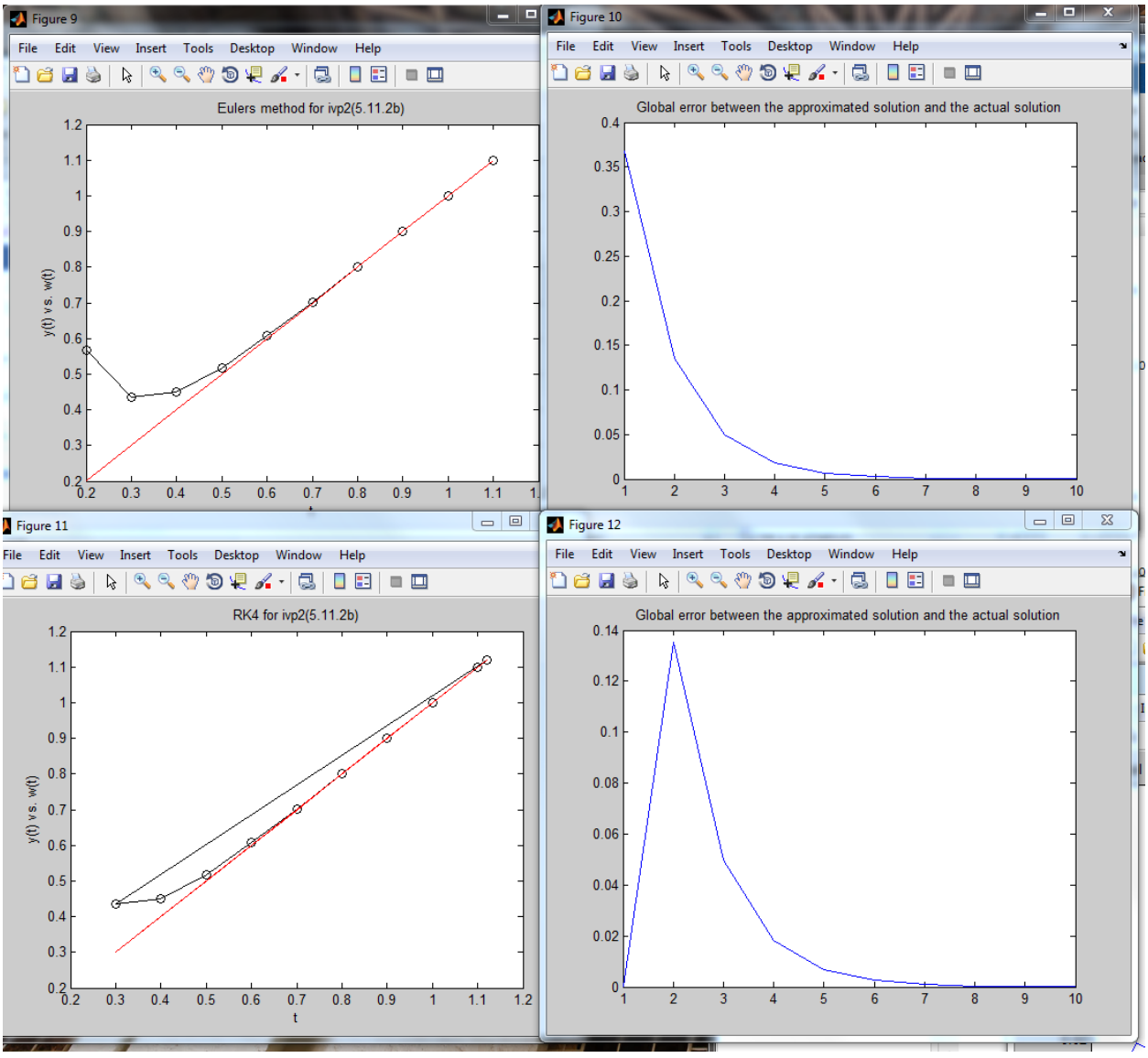


For the implicit methods, the results using am3 and adams4 are also pretty much the same. The global error for adams-moulton 3 step implicit starts to increase (spike) after 3 while for the adams 4th order predictor-corrector method the increase happens after 4, because. This is because adams4 uses 4 preceding times, ti, ti-1, ti-2, ti-3, to generate wi+1.

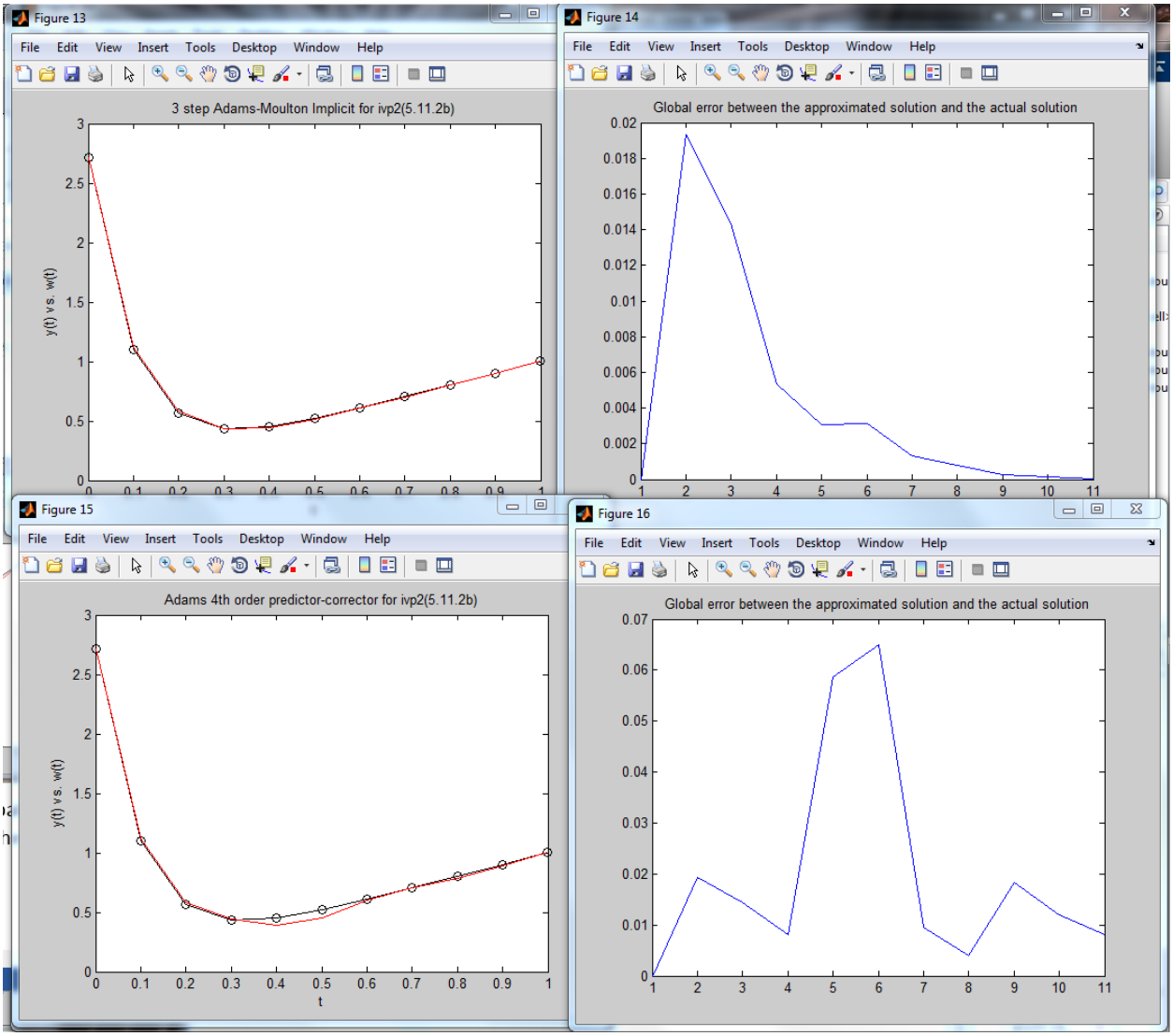
Now onto the second exercise 5.11.2b with step size h=.1:

```
%======Now for ivp2, exercise 5.11.2b with h=.1==========
```

the first 2 are Euler's method, second 2 are RK(4), then the implicit 3-step Adams-Moulton method, and finally adams4.m, which is the Adams fourth order predictor-corrector method, (ABM).



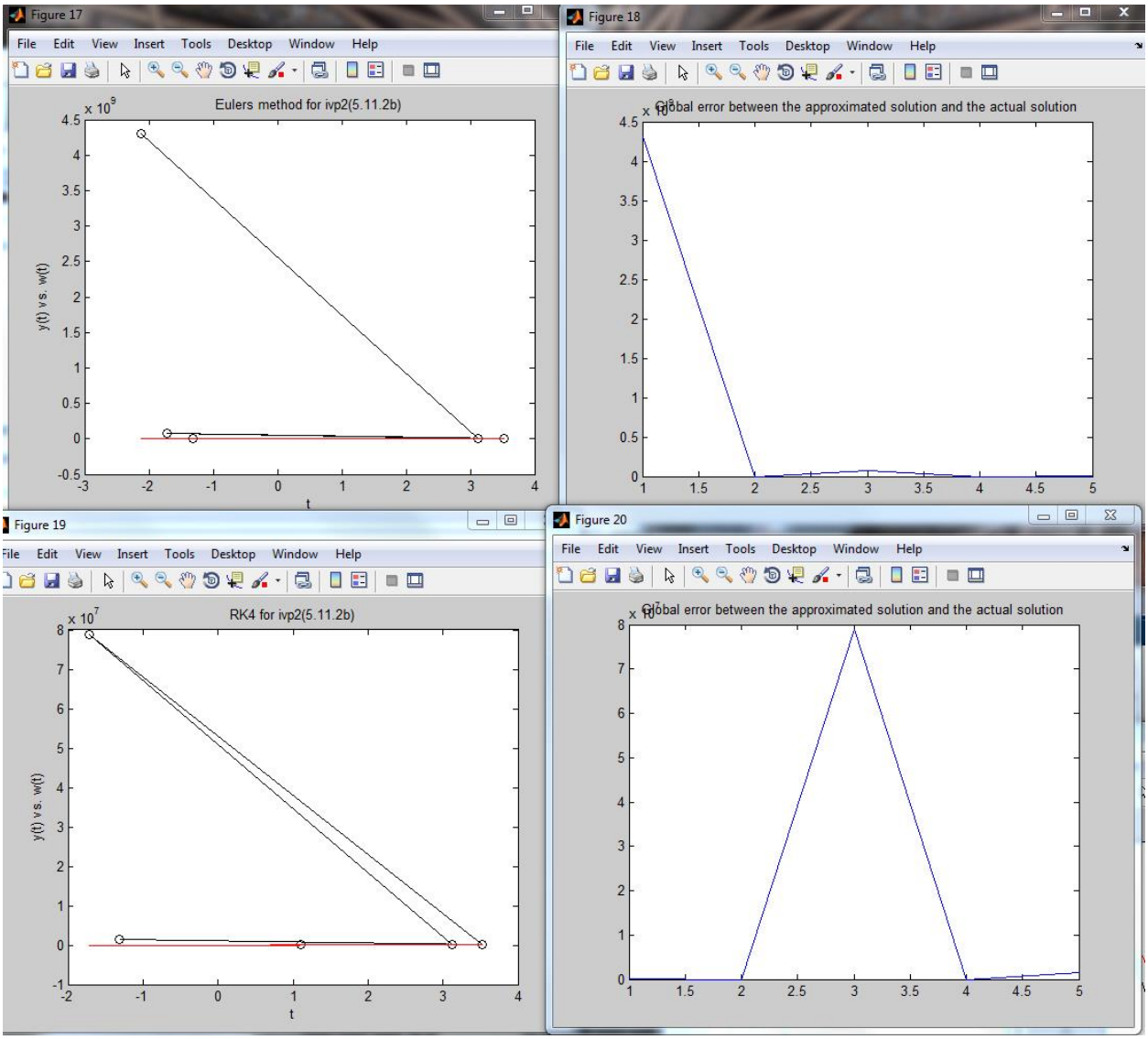
RK4 is much better, the global error does not go beyond .15, where as for Euler, you can see that the scale is in the .3 /.4 range.



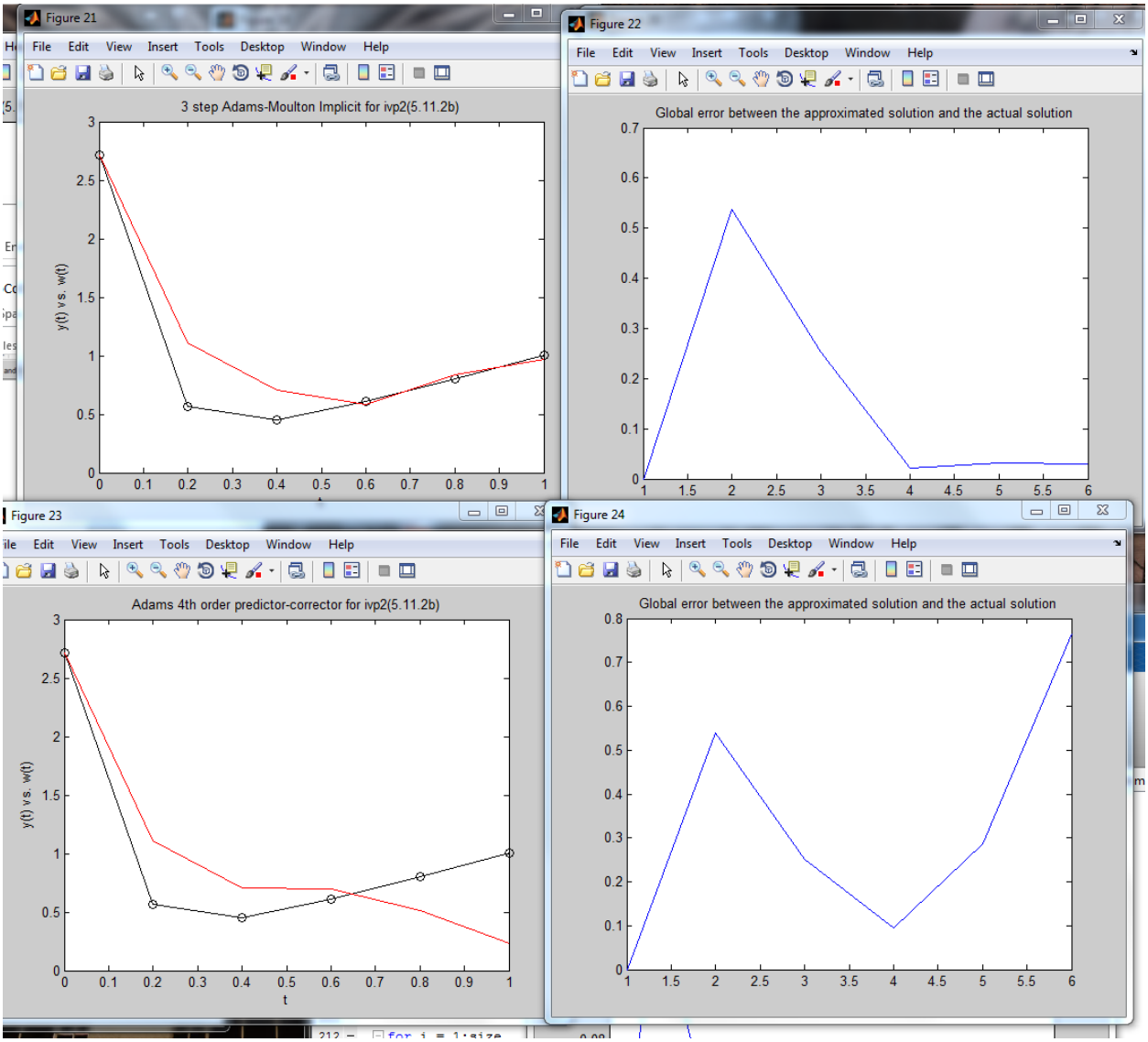
In this set of graphs, the top one, the am3.m, 3-step Adams-Moulton Implicit is a much better approximation.

Now with a large step size: h=.2, the approximations are not as good, compared to h=.1, smaller step size is more accurate.

```
%======Now for ivp2, exercise 5.11.2b with h=.2==========
```



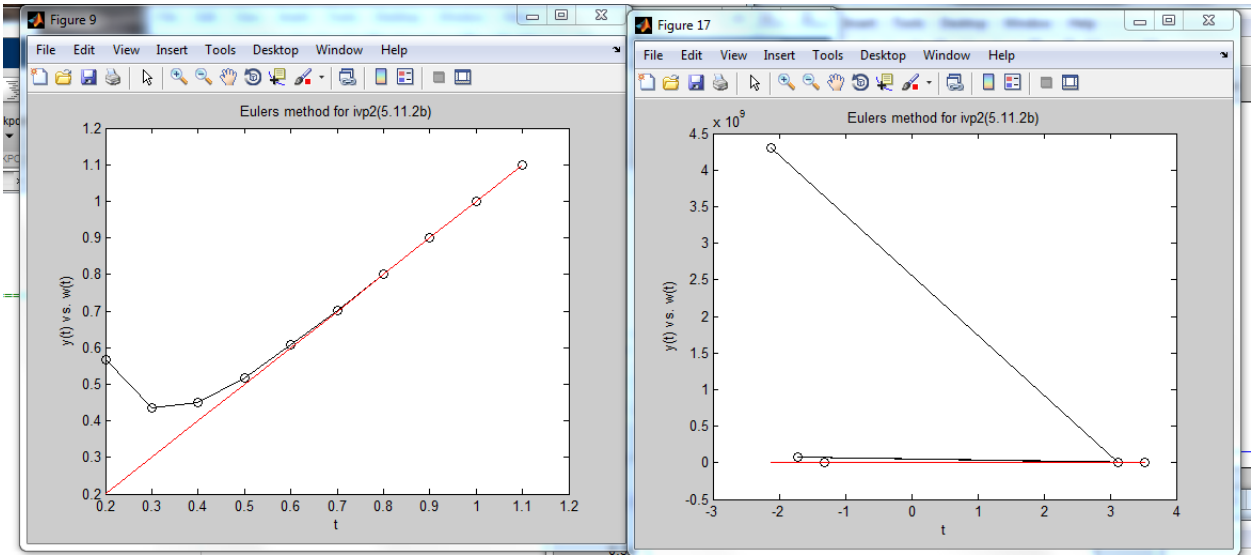
Runge-Kutta has greater global error for this step size. The Euler approximation is close to the actual solution.



The 3 step Adams-Moulton Implicit graph is a better approximation in this case than adams 4th order predictor-corrector.

This is a STIFF IVP, which is why we use two different step sizes to see how it affects Euler's method. Now let's compare them side by side:

That's figure 9 vs figure 17



It is confirmed! In the case of stiff IVPs such as this second one, a decrease in step size using Euler's method results in a better approximation!