Math 128a - Homework 8 - Due May 2
1) Problem 8.4.4 (Page 555)

Solution: As discussed in the text, the fourth-order Adams-Bashforth formula is a formula of the type

$$x_{n+1} = x_n + h[Af_n + Bf_{n-1} + Cf_{n-2} + Df_{n-3}], \tag{1}$$

where $x_i = x(t_i)$, $f_i = f(t_i, x(t_i))$, and the t_i's are equally spaced points a distance h from each other. These coefficients are determined by approximating the integral

$$\int_{t_n}^{t_{n+1}} f(t, x(t))\, dt \approx h[Af_n + Bf_{n-1} + Cf_{n-2} + Df_{n-3}].$$

As stated in the text, we are allowed to assume that $t_n = 0$ and $h = 1$, and then we require that the approximation is exact for all polynomials of degree at most 3.

We take the four polynomials

$$p_0(t) = 1, \quad p_1(t) = t, \quad p_2(t) = t(t+1), \quad p_3(t) = t(t+1)(t+2)$$

and substitute them into the equation

$$\int_0^1 p_n(t)\, dt = Ap_n(0) + Bp_n(-1) + Cp_n(-2) + Dp_n(-3).$$

This gives us the four equations

$$\begin{aligned} 1 &= A + B + C + D, \\ \frac{1}{2} &= -B - 2C - 3D, \\ \frac{5}{6} &= 2C + 6D, \\ \frac{9}{4} &= -6D. \end{aligned}$$

These equations are easily solved via back-substitution to get

$$D = \frac{-9}{24}, \quad C = \frac{37}{24}, \quad B = -\frac{59}{24}, \quad A = \frac{55}{24}.$$

Substituting these values into (1) gives the desired formula.

2) Problem 8.4.5 (Page 555)

Solution: The fourth-order Adams-Moulton formula that we seek is of the form

$$x_{n+1} = x_n + h[Af_{n+1} + Bf_n + Cf_{n-1} + Df_{n-2}], \tag{2}$$

notation as in the previous problem. As with the Adams-Bashforth formula, we determine the unknown coefficients in (2) by approximating the integral $\int_0^1 f(t, x(t))\, dt$ and requiring this approximation to be exact on polynomials of degree at most 3. To this end, we substitute the polynomials

$$p_0(t) = 1, \quad p_1(t) = (t-1), \quad p_2(t) = (t-1)t, \quad p_3(t) = (t-1)t(t+1)$$

into the equation

$$\int_0^1 p_n(t)\, dt = Ap_n(1) + Bp_n(0) + Cp_n(-1) + Dp_n(-2),$$

getting the equations

$$\begin{aligned} 1 &= A + B + C + D, \\ -\frac{1}{2} &= -B - 2C - 3D, \\ -\frac{1}{6} &= 2C + 6D, \\ -\frac{1}{4} &= -6D. \end{aligned}$$

Solving these equations by back-substitution yields

$$D = \frac{1}{24}, \quad C = -\frac{5}{24}, \quad B = \frac{19}{24}, \quad A = \frac{9}{24},$$

and substituting these coeffiecients into (2) gives us the formula in the text.

3) The purpose of this question is to familiarize you with the properties of the ODE solvers in Matlab and their numerical properties. Consider the 0DE

$$x'(t) = c(t) \cdot (x(t) - \sin(t)) + cos(t) = f(x(t), t)$$

with initial conditions $x(0) = 0$. The solution is clearly $x(t) = \sin(t)$ for any continuous $c(t)$. However, we will see that the numerical solvers do not all behave the same way.

The command "options = odeset('AbsTol',1e-4,'RelTol',1e-4)" and then passing options as an argument to the ODE solver means that the estimated LTE at step i should be at most

$$\max(\text{RelTol} \cdot |x_i|, \text{AbsTol}) = 10^{-4} * \max(|x_i|, 1)$$

where $x(i)$ is the i-th computed solution. (The default values of AbsTol and RelTol are 1e-3.)

Solve the above ODE on the interval $[0, 10]$ for each combination of solver, value of AbsTol=RelTol, and $c(t)$ given below ($3 \times 2 \times 6 = 36$ combinations in all):

- ode solver (ode45, ode23, or ode15s)
- RelTol=AbsTol (10^{-4} or 10^{-8}) and
- $c(t)$ (see list below)

For each combination, report

- number of time steps take
- number of seconds taken
- maximum absolute error $\max_i |x_i - sin(t_i)|$, where t_i and x_i are the output arrays of times and solution values at those times.
- maximum relative error $\max_i |x_i - sin(t_i)|/|sin(t_i)|$.

To measure the time taken, measure only the time to solve, not plot or do anything else. Use the command sequence "secs = cputime; [t,x]=ode45(...); secs = cputime - secs" to get the running time in seconds.

Here are the values of $c(t)$ to try for each ODE solver (recall the exact solution of the ODE does not depend on $c(t)$):

- $c(t) = 2$

- $c(t) = 1$
- $c(t) = -1$
- $c(t) = -10$
- $c(t) = -100$
- $c(t) = -10 * \sin(t)$

Comment on your output, saying which method is most accurate, which method is most efficient (measured by the number of steps taken and the time taken and how this depends on $c(t)$. You should plot the solutions "plot(t,y)" to see what is going on but you do not have to turn any plots in. Comment on how the error depends on t for $c(t) = -10 * \sin(t)$.

Solution: Following is a table of all the requested data for this problem.

Tol	$c(t)$	Solver	Steps	Secs	Max Abs Error	Max Rel Error
10^{-4}	2	ode45	109	.53	2.4174e+03	3.4192e+04
		ode23	118	.34	4.1398e+04	6.8977e+05
		ode15s	75	.65	1.5133e+05	3.4357e+06
	1	ode45	69	.06	1.2966e-01	3.9250e+01
		ode23	67	.11	1.8297e+00	2.3562e+01
		ode15s	56	.15	8.3521e+00	1.7015e+04
	-1	ode45	73	.06	8.0843e-05	5.6942e-04
		ode23	66	.10	2.8823e-04	3.5550e-03
		ode15s	64	.19	6.7010e-04	2.4235e-02
	-10	ode45	297	.22	9.9147e-05	3.0718e-03
		ode23	116	.17	2.7042e-04	4.0541e-02
		ode15s	53	.15	2.3912e-04	2.8136e-03
	-100	ode45	1417	.98	6.2119e-05	8.3079e-02
		ode23	429	.75	1.8758e-04	5.6284e-02
		ode15s	55	.15	3.0228e-05	2.7062e-04
	$-10\sin(t)$	ode45	305	.24	6.8475e+03	3.2893e+05
		ode23	242	.34	6.3404e+04	4.5166e+06
		ode15s	170	.53	3.9349e+04	2.9213e+07
10^{-8}	2	ode45	633	.44	5.8566e-01	5.7602e+01
		ode23	1621	2.15	1.1485e+01	8.6072e+04
		ode15s	239	.59	1.2417e+01	4.0044e+02
	1	ode45	397	.29	2.8574e-05	2.2521e-03
		ode23	1333	1.78	4.0307e-04	1.2434e-01
		ode15s	201	.52	1.2843e-03	3.7418e-01
	-1	ode45	405	.30	1.1897e-08	2.8612e-06
		ode23	1333	1.77	5.8660e-08	1.1364e-05
		ode15s	200	.48	1.2113e-07	2.2496e-05
	-10	ode45	1665	1.14	5.6769e-09	1.2871e-05
		ode23	1363	1.81	1.2880e-06	5.4528e-05
		ode15s	200	.49	3.5093e-08	2.4872e-06
	-100	ode45	6825	4.64	7.7461e-09	2.7744e-04
		ode23	3338	4.42	8.9634e-08	2.1924e-04
		ode15s	209	.51	3.6833e-08	2.5635e-06
	$-10\sin(t)$	ode45	1277	.91	2.6726e-01	4.6314e+01
		ode23	2777	3.73	3.3587e+01	1.8103e+04
		ode15s	359	1.00	8.6419e-01	5.6370e+04

From this table, we see that almost without exception, `ode45` gives the lowest maximum absolute error. This command also gives the lowest maximum relative error, except when $c(t) = -10$ and $c(t) = -100$, when `ode15s` does a better job. This should come as little surprise, as those c values correspond to a very stiff ODE, and `ode15s` is designed to solve stiff ODEs.

The solver `ode15s` is the clear winner in terms of space efficiency (number of time steps, which is proportional to the amount of space or memory needed to store the solution): in every case, `ode15s` uses the fewest number of time steps. The question of time efficiency (number of seconds of computer time needed to compute the solution) is more complicated. For the stiff ODEs, `ode15s` is the fastest, sometimes by more than a factor of eight. For other choices of $c(t)$, `ode45` is almost always the fastest. Note also that `ode23` is especially inefficient (in terms of both time *and* space) when AbsTol and RelTol

are 10^{-8}; this is probably explained by the fact that the 2nd and 3rd order Runge-Kutta method is only $O(h^4)$, whereas the other methods are $O(h^6)$; thus, `ode23` needs much smaller step sizes (and thus many more steps and more time) than the other two solvers to achieve a small local truncation error.

Finally, observe that all the solvers are quite inaccurate when $c(t) = 2$, $c(t) = 1$, or $c(t) = -10\sin(t)$. All of these choices of $c(t)$ have regions of t-values in which $c(t) > 0$. For these values of t, the solution $x(t) = \sin(t)$ is exponentially repelling, so once a solver gets off the true solution, it tends to move farther and farther away from that solution. That is, small errors turn very quickly into large errors. This sort of behavior can be seen in the plots for these cases. When we require the LTE to be small (when AbsTol=RelTol=10^{-8}), the solvers do a better job in these cases, but they are still nowhere near as accurate as we requested.

When $c(t) = -10\sin(t)$, we see in the plot that the computed solution veers away from the true solution, but then comes back to the true solution; this is because there are regions of t-values for which $-10\sin(t)$ is negative, and for these t's, the true solution is exponentially *attracting*. Thus, the compute solution is alternately pushed away from and pulled back to the true solution.

The code used to produce this output is given below.

```
res = []; for tol = [1e-4 1e-8],
   options = odeset('AbsTol',tol,'RelTol',tol);
   for c = [2 1 -1 -10 -100],
      secs = cputime;
      [t,y]=ode45(@tst1,[0 10],0,options,c);
      secs = cputime - secs;
      aerr = max(abs(y-sin(t)));
      rerr = max(abs(y-sin(t))./abs(sin(t)));
      res1 = [tol, c, 45, aerr, rerr, length(t), secs]
      res = [res;res1];
      plot(t,y,'b',t,sin(t),'r'), pause
%
      secs = cputime;
      [t,y]=ode23(@tst1,[0 10],0,options,c);
      secs = cputime - secs;
      aerr = max(abs(y-sin(t)));
      rerr = max(abs(y-sin(t))./abs(sin(t)));
      res1 = [tol, c, 23, aerr, rerr, length(t), secs]
      res = [res;res1];
      plot(t,y,'b',t,sin(t),'r'), pause
%
      secs = cputime;
      [t,y]=ode15s(@tst1,[0 10],0,options,c);
      secs = cputime - secs;
      aerr = max(abs(y-sin(t)));
      rerr = max(abs(y-sin(t))./abs(sin(t)));
      res1 = [tol, c, 15, aerr, rerr, length(t), secs]
      res = [res;res1];
      plot(t,y,'b',t,sin(t),'r'), pause
   end
%
   c = -10;
   secs = cputime;
   [t,y]=ode45(@tst2,[0 10],0,options,c);
   secs = cputime - secs;
```

```
   aerr = max(abs(y-sin(t)));
   rerr = max(abs(y-sin(t))./abs(sin(t)));
   res1 = [tol, c, 45, aerr, rerr, length(t), secs]
   res = [res;res1];
   plot(t,y,'b',t,sin(t),'r'), pause
%
   secs = cputime;
   [t,y]=ode23(@tst2,[0 10],0,options,c);
   secs = cputime - secs;
   aerr = max(abs(y-sin(t)));
   rerr = max(abs(y-sin(t))./abs(sin(t)));
   res1 = [tol, c, 23, aerr, rerr, length(t), secs]
   res = [res;res1];
   plot(t,y,'b',t,sin(t),'r'), pause
%
   secs = cputime;
   [t,y]=ode15s(@tst2,[0 10],0,options,c);
   secs = cputime - secs;
   aerr = max(abs(y-sin(t)));
   rerr = max(abs(y-sin(t))./abs(sin(t)));
   res1 = [tol, c, 15, aerr, rerr, length(t), secs]
   res = [res;res1];
   plot(t,y,'b',t,sin(t),'r'), pause
end
```

The code above calls the following functions:

```
function yp = tst1(t,y,c)
yp = c*(y-sin(t)) + cos(t);

function yp = tst2(t,y,c)
yp = c*sin(t)*(y-sin(t)) + cos(t);
```