Math 561 FINAL HOMEWORK due Thursday, Dec 18, 2014, at noon
Fritz Keinert

For each problem that uses Matlab or some other tool, you should hand in a printout of the relevant script or function file(s), or a transcript of your interactive session, plus whatever outputs or plots are requested. Put the problems in the proper order, and label all printouts clearly. The final output should have full accuracy (`format long`); intermediate results can be shorter, if you want.

1. (This is a modification of Gautschi, Chapter 6, Machine Assignment 4, page 460)
(a) Write a Matlab routine for plotting the region of absolute stability for the 4-step Adams-Bashforth method.

Hints: The region of stability of some methods extends into the right half plane, but you can ignore that here. All regions are symmetric about the real axis, so it suffices to do the computations for the second quadrant (but plot the whole region).

Let θ go from $\pi/2$ to π, in reasonably small steps. For each θ, find the value of r where the largest root of the characteristic polynomial for $h\lambda = re^{i\theta}$ becomes larger than 1. This will give you the curve in polar coordinates. Use the `fzero` routine to find the radius numerically, for each angle.

The point $\theta = \pi/2$ may require special treatment. Often (but not always), $r = 0$ there, but Matlab goes off searching forever for another root. I found it easier to start at π and work backwards. You can use the result from the previous step as the initial guess for the next step.

It is enough to turn in a plot of the region. Use `axis equal`.

(b) Do the same for the 3-step Adams-Moulton method.

(c) Do the same for the 4th-order ABM PECE predictor/corrector method (4-step AB, 3-step AM).

Notes: This one took me a while. To see the difficulty, consider the PEC case first. You start with the correct values for $y_n, \ldots, y_{n+3}$ and $f_n = \lambda y_n, \ldots, f_{n+3} = \lambda y_{n+3}$. After one step, you get a new y_{n+4} and f_{n+4}, and find $y_{n+4} = \phi(h\lambda)y_n$.

However, $f_{n+4} \neq \lambda y_{n+4}$ any more. At the next step, you get a different ϕ, and at the next step yet another one. All of them have different regions of stability. I could not figure out the pattern. I think the multivalue formulation is the way to go here.

However, for PECE it is much easier. Here, $f_{n+4} = \lambda y_{n+4}$, and things are the same at every step.

What you should observe is that the region for AB is the smallest, the region for PECE ABM is bigger, and the region for AM alone is the biggest. See also problem 2(c). (15)

SOLUTION: **(a)** Here is my code. I used Matlab to compute the relevant growth factor, since I was playing around multiple steps, but that is not necessary.

```
syms yn yn1 yn2 yn3 yn4 r z
n = 50;
theta = pi:-pi/(2*n):pi/2;
theta(end+1) = 0;

% AB4
P = yn3 + z/24*(55*yn3-59*yn2+37*yn1-9*yn);
C = yn3 + z/24*(9*yn4+19*yn3-5*yn2+yn1);
```

```
% predictor only
disp(' '); disp('Predictor'); disp('=========');
E = expand(subs(yn4-P,{yn4,yn3,yn2,yn1,yn},{r^4,r^3,r^2,r,1}));
[charpoly, powers] = coeffs(E,r)

rho_p = zeros(size(theta));
rho0 = 0.5;
for j = 1:n+1
    c = @(t) max(abs(roots(double(subs(charpoly,z,exp(i*theta(j))*t)))))-1;
    rho_p(j) = fzero(c,rho0,optimset('MaxFunEvals',100));
    rho0 = rho_p(j);
end
figure(1);
plot(rho_p.*cos(theta),rho_p.*sin(theta),'g',...
     rho_p.*cos(theta),-rho_p.*sin(theta),'g')
title('Predictor Only','FontSize',20)
axis equal
```

The output is

```
Predictor
=========
charpoly =
[ 1, - (55*z)/24 - 1, (59*z)/24, -(37*z)/24, (3*z)/8]
powers =
[ r^4, r^3, r^2, r, 1]
```

which means that the characteristic polynomial is

$$r^4 - \left(\frac{55z}{24} + 1\right) r^3 + \frac{59z}{24} r^2 - \frac{37z}{24} r + \frac{3z}{8}.$$

(b) Here is the relevant part of the code. The rest of the code is basically identical to before.

```
% corrector only
disp(' '); disp('Corrector'); disp('=========');
E = expand(subs(yn4-C,{yn4,yn3,yn2,yn1,yn},{r^4,r^3,r^2,r,1}));
[charpoly, powers] = coeffs(E,r)
```

The output is

```
Corrector
=========
charpoly =
[ 1 - (3*z)/8, - (19*z)/24 - 1, (5*z)/24, -z/24]
powers =
[ r^4, r^3, r^2, r]
```

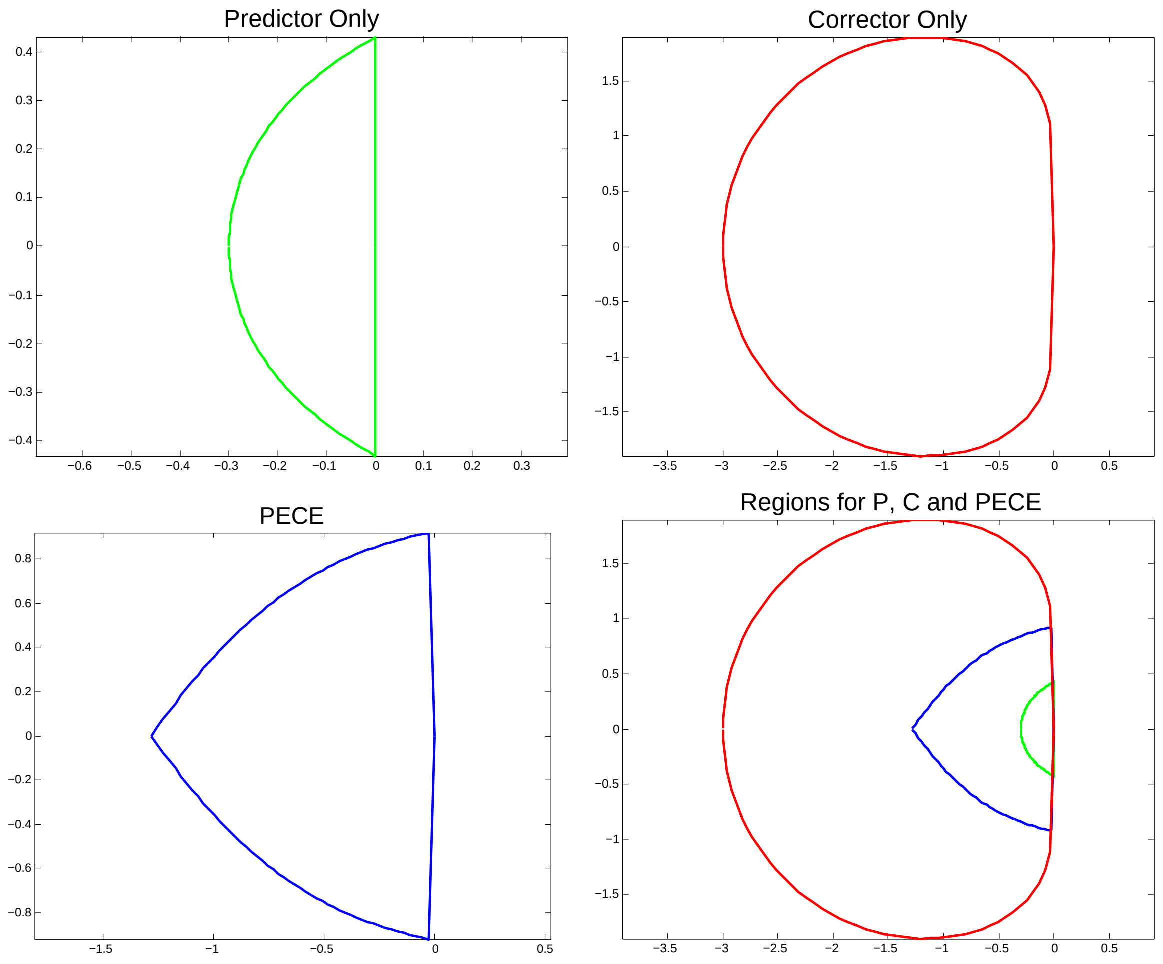
(c) Here is the relevant part of the code. Most of it is identical to before.

```
% predictor + corrector
disp(' '); disp('Predictor+Corrector'); disp('===================');
PC = subs(C,yn4,P);
E = expand(subs(yn4-PC,{yn4,yn3,yn2,yn1,yn},{r^4,r^3,r^2,r,1}));
[charpoly, powers] = coeffs(E,r)
```

The output is

```
charpoly =
[ 1, - (7*z)/6 - (55*z^2)/64 - 1, (59*z^2)/64 + (5*z)/24, ...
  - z/24 - (37*z^2)/64, (9*z^2)/64]
powers =
[ r^4, r^3, r^2, r, 1]
```

All the graphs are here:



2. Consider the predictor-corrector method given by a 1-step Adams-Bashforth predictor and 1-step Adams-Moulton corrector:

$$\begin{aligned} y_n &= y_{n-1} + hf_{n-1} \\ y_n &= y_{n-1} + \frac{h}{2}\left[f_n + f_{n-1}\right] \end{aligned}$$

(a) Convert this to multivalue form, that is,

$$\begin{aligned} \mathbf{y}_n^{(0)} &= B\mathbf{y}_{n-1} \\ \mathbf{y}_n^{(k+1)} &= \mathbf{y}_n^{(k)} + G(\mathbf{y}_n^{(k)})\mathbf{c} \end{aligned}$$

Here

$$\mathbf{y} = \begin{pmatrix} y \\ hy' \end{pmatrix}$$

so B is just a 2 by 2 matrix, and $\mathbf{c}$ is a 2-vector.

(b) For the test problem $y' = \lambda y$, calculate the matrix S which governs stability, for the schemes PEC and $PECE$. For both, verify that the methods are zero stable, and determine how far along the negative real axis the region of stability extends.

(c) (5 points extra credit) We determined in class that the region of stability of the 1-step AM corrector includes the entire left half plane. However, the regions of stability for the multivalue PEC and PECE are quite small, and even with more iterations do not get very large. Explain the apparent contradiction. (15)

Solution: **(a)** The formulas are

$$\begin{aligned} y_n^{(0)} &= y_{n-1} + hf_{n-1} \\ hf_n^{(0)} &= hf_{n-1} \\ y_n^{(1)} &= \left[y_n^{(0)} - hf_{n-1}\right] + \frac{h}{2}\left[f_n + f_{n-1}\right] \\ &= y_n^{(0)} + \frac{h}{2}\left[f_n^{(1)} - f_n^{(0)}\right], \end{aligned}$$

so

$$B = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \qquad \mathbf{c} = \begin{pmatrix} 1/2 \\ 1 \end{pmatrix}.$$

(b) For PEC, we have

$$S = (I + \mathbf{c} \cdot DG)\, B = \begin{pmatrix} 1 + \frac{h\lambda}{2} & \frac{1}{2} + \frac{h\lambda}{2} \\ h\lambda & h\lambda \end{pmatrix}.$$

The eigenvalues are

$$\frac{2 + 3h\lambda \pm \sqrt{9(h\lambda)^2 + 4h\lambda + 4}}{4}.$$

For $h\lambda = 0$, the eigenvalues are 1, 0, which implies zero stability. You can check (analytically or numerically) that the eigenvalues are ≤ 1 on $[-1, 0]$.

For PECE, we have

$$S = (I + \mathbf{c}_2 \cdot DG)\,(I + \mathbf{c} \cdot DG)\, B = \begin{pmatrix} 1 + \frac{h\lambda}{2} & \frac{1}{2} + \frac{h\lambda}{2} \\ h\lambda + \frac{1}{2}(h\lambda)^2 & \frac{1}{2}h\lambda + \frac{1}{2}(h\lambda)^2 \end{pmatrix}.$$

The eigenvalues are 0 and $1 + h\lambda + (h\lambda)^2/2$. This is zero stable, and stable on $[-2, 0]$.

(c) The region of the AM-1 method was derived under the assumption that we solve the implicit equation for y_{n+1} exactly. For stiff equations, this requires a nonlinear equation solver, since fixed point iteration does not work any more. The multivalue formulation is inherently based on fixed point iteration, and the region becomes smaller.

3. Consider the ODE

$$\begin{aligned} y' &= -10y + 9.9e^{-t/10} \\ y(0) &= 2. \end{aligned}$$

The true solution is $y = e^{-10t} + e^{-t/10}$, so this is mildly stiff. Solve the equation on the interval $[0, 2]$ with step size $h = 0.1$, using Gear's third order method in the multi-value formulation, that is

$$\begin{aligned} \mathbf{y}_n^{(0)} &= B\mathbf{y}_{n-1} \\ \mathbf{y}_n^{(k+1)} &= \mathbf{y}_n^{(k)} + G(\mathbf{y}_n^{(k)})\mathbf{c} \end{aligned}$$

where

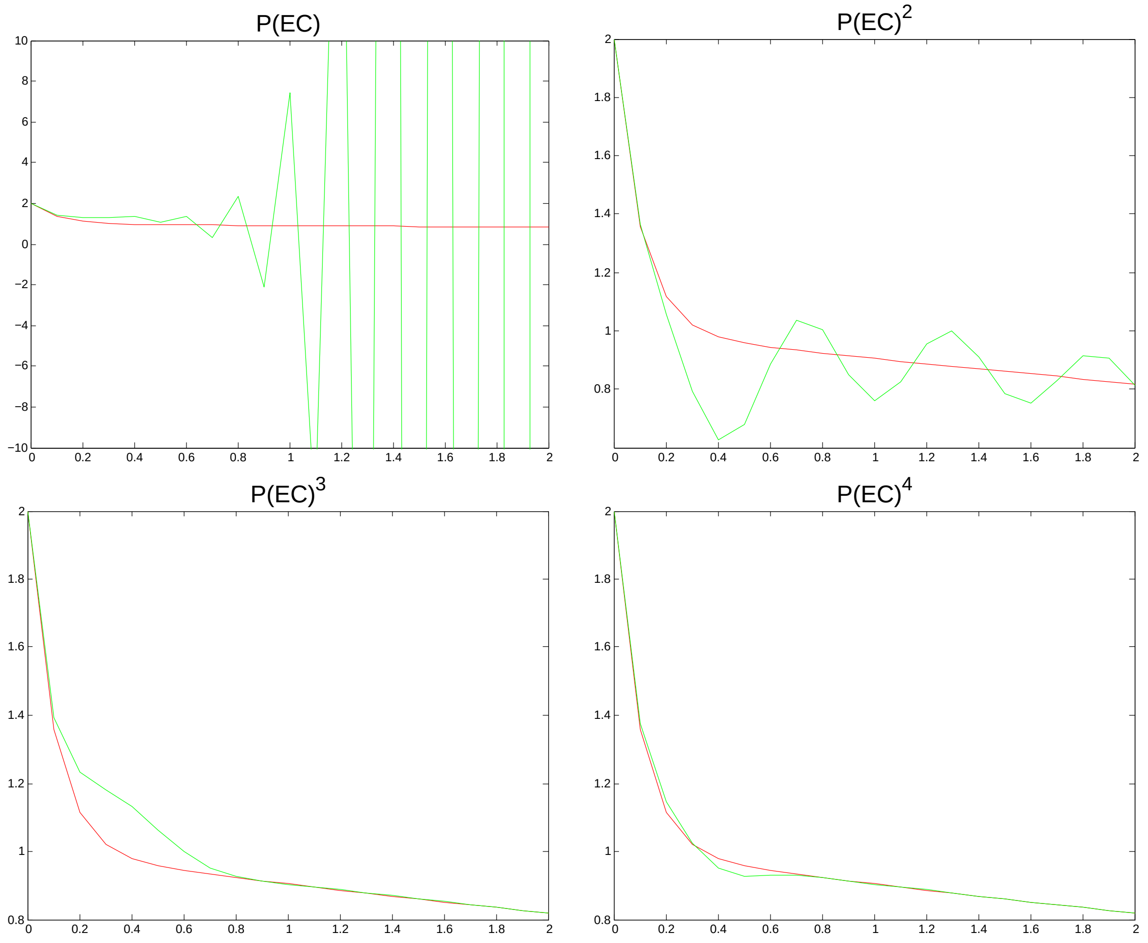
$$B = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad \mathbf{c} = \begin{pmatrix} 6/11 \\ 1 \\ 6/11 \\ 1/11 \end{pmatrix}, \quad \mathbf{y} = \begin{pmatrix} y \\ hy' \\ (h^2/2)y'' \\ (h^3/6)y''' \end{pmatrix}.$$

(a) Start with the correct $\mathbf{y}$ for $t = 0$. For each of the schemes PEC, $P(EC)^2$, $P(EC)^3$, $P(EC)^4$, plot the correct solution and the numerical solution.

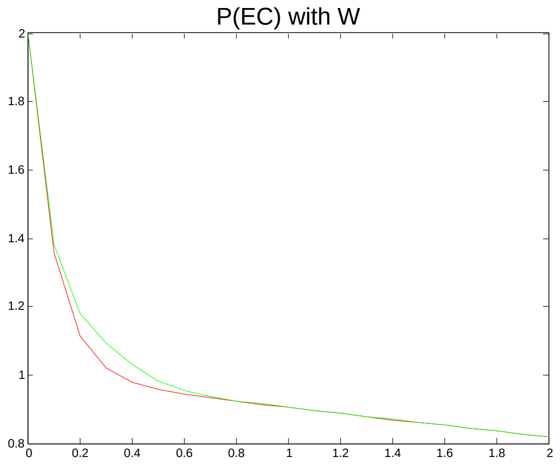
Note: Observe that the scheme is unstable in the first case. The region of stability of PEC in this case is only $[-0.45, 0]$, and in this assignment $h\lambda = -1$. The other solutions are not very good, either, but at least they don't blow up. The solutions get much better with a smaller h.

(b) Put in the factor W, with scheme PEC. Observe that it is now stable, with results similar to the unmodified $P(EC)^3$ or $P(EC)^4$. (15)

Solution: Here are the solution curves:



The factor W is $1/(1 - (6/11)h\lambda)$, which in this case is $11/17$.



4. Solve the boundary value problem

$$y''(t) = \frac{3}{2}y^2(t)$$
$$y(0) = 4$$
$$y(1) = 1$$

by a shooting method. The required steps are as follows:

(a) Write a function `g(s)` which calculates the amount by which the solution with initial conditions

$$y(0) = 4$$
$$y'(0) = s$$

misses the value $y(1) = 1$. In other words: Use `ode45` to solve the problem with initial condition $y'(0) = s$ numerically, and return $y(1) - 1$.

(b) Plot `g(s)` for $s \in [-40, -5]$ and read off initial guesses for the correct s. There are two solutions.

(c) Use `fzero` to find the exact values of s, to default accuracy.

(d) Plot the two solution curves.

(e) (5 points extra credit) One of the two solutions has a very simple closed form. Find it, and generalize it to a one-parameter family of solutions. (The full ODE has a two-parameter family of solutions, but I don't know how to find it). (15)

Solution: **(a)** Here is my code for all the parts. The function `f` can be written as an inline function, but that does not work for `g`.

```
% main program
global f
f = @(t,y) [y(2); 3/2 * y(1)^2];

% part (b)
s = -40:-5;
gs = zeros(size(s));
for i = 1:length(s)
    gs(i) = g(s(i));
end
figure(1);
```

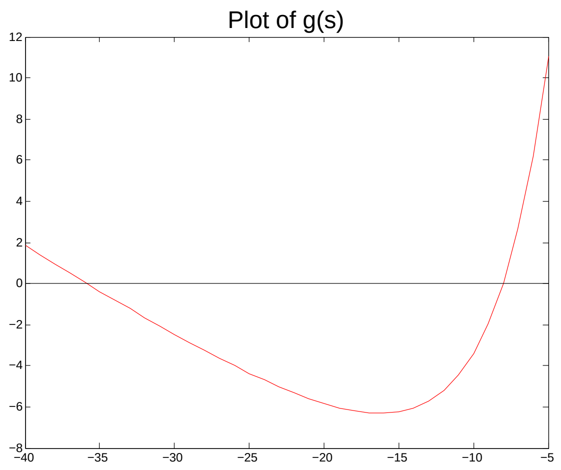
```
plot(s,gs,'r',[-40,-5],[0,0],'k');
title('Plot of g(s)','FontSize',20);

% part (c)
s1 = fzero(@g,-35)
s2 = fzero(@g,-8)
[t1,y1] = ode45(f,[0,1],[4;s1]);
[t2,y2] = ode45(f,[0,1],[4;s2]);

% part (d)
figure(2);
plot(t1,y1(:,1),'r',t2,y2(:,1),'g');
title('Two Solutions of BVP','FontSize',20);

% file g.m
function miss = g(s)
global f
[t,y] = ode45(f,[0,1],[4;s]);
miss = y(end,1) - 1;
```

(b) The graph looks like this:

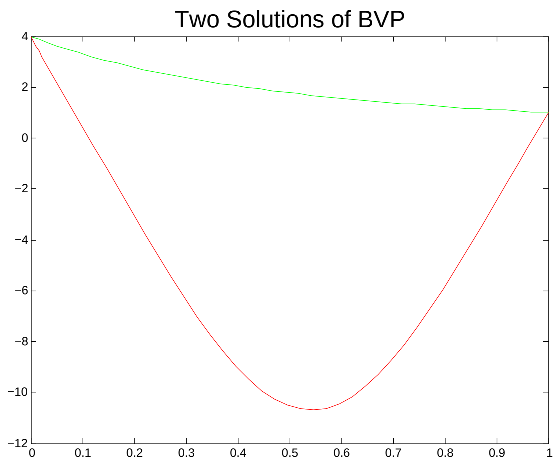


The zeros seem to be near -35 and -8.

(c) The output is

```
s1 =
 -35.858569016562328
s2 =
  -7.999994873574373
```

(d) The solution curves look like this:



(e) The second solution is actually exactly -8, with $y(t) = 4(1+t)^{-2}$. All functions of the form

$$y(t) = \frac{4}{(c+t)^2}$$

are solutions.

5. This problem will use most of what you have learned this term, all in the same problem. Make sure you get an early start.

The xz coordinate plane represents a slice through the ocean: x is horizontal distance, z is depth below the ocean surface (so the z axis points downward).

We assume the speed of sound $c(z)$ in ocean water depends only on depth z. Various values of z and $c(z)$ (in ft/sec) are given in a table in file `sound.dat`. Values at other points need to be calculated by spline interpolation.

A sound source is located at position $(0, z_0)$ and emits sound rays in all directions. Because of the varying sound speed, sound travels along curved rays instead of straight lines. Each sound ray is given by a function $z(x)$. At any point on a ray, $\theta = \theta(x)$ is the angle between the horizontal and the tangent (see picture)

$$\tan\theta = \frac{dz}{dx}.$$

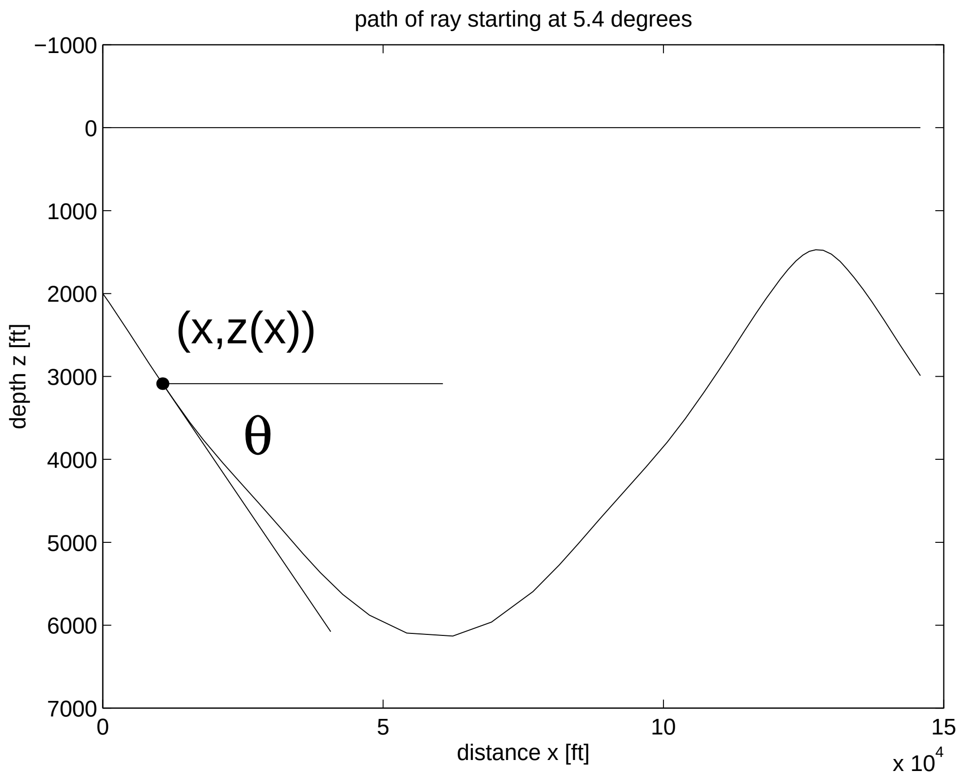
Snell's law states that

$$\frac{\cos\theta}{c(z)} = \frac{\cos\theta_0}{c(z_0)} = \text{constant} = A.$$

From this, we can derive the ODE

$$\begin{aligned}\frac{d^2z}{dx^2} &= -\frac{c'(z)}{A^2 c(z)^3},\\ z(0) &= z_0,\\ \frac{dz}{dx}(0) &= \tan\theta_0.\end{aligned}$$

We want to figure out which of the rays reach a given receiver. We will break the work into several stages. This problem is very similar to problem 4, except that the details are much more messy.



Here is what you need to do:

(a) First make sure you can trace one ray at a time. "Trace a ray" just means "solve the ODE, and plot the resulting curve".

Trace the ray beginning at $z_0 = 2000$ feet and $\theta_0 = 5.4°$ from $x = 0$ to $x = 24$ nautical miles. A nautical mile is 6,076 feet. Plot the resulting ray. It should look a lot like my picture. Use `axis ij` to turn the z axis upside down.

(b) Suppose the receiver is located at $x = 24$ nautical miles, $z = 3000$ ft.

Write a function $g(\theta_0)$ which measures the vertical distance by which the ray starting with angle θ_0 misses the receiver.

Print out a table of values of g for θ_0 going from $-10°$ to $10°$ in steps of $1°$.

(c) Use `fzero` to locate the zeros of g near $-8°$, $4°$ and $7°$ (there are more zeros than that).

Print out these values of θ_0, and plot the three corresponding rays in one picture.

(d) For the ray from part (c) with initial angle near 4°, figure out how long it takes the sound to travel from source to receiver.

From *speed = distance/time* we get *time = distance/speed*, and the travel time is found by integrating that along the curve with respect to arclength:

$$T = \int \frac{\sqrt{1 + (z'(x))^2}}{c(z(x))}\, dx.$$

Now for some major programming hints:

General Hints:

1. The numbers are very sensitive to minor variations in programming. Don't expect your results and someone else's to match to more than 2 or 3 decimals. If the graphs look right, you are doing fine.

In fact, I don't even want to see any printouts of results except what I explicitly request, especially not the numbers that make up the sound rays. Just plot them. Print out your subroutines or diary files as usual, though.

2. Watch out for the units. Print all results in feet and degrees, but of course you need to convert the angles back and forth to radians to evaluate any trig functions.

Specific Hints:

Part (a): In principle, you know how to do this, since you used `ode45` before. The problem is that the function `f` that describes the ODE requires values of $c(z)$, $c'(z)$. It would be very inefficient to compute the splines for $c(z)$ and $c'(x)$ from scratch every single time, so we compute the spline coefficients once and for all, and then just evaluate them every time. Use global variables to pass the spline coefficients, as well as the constant A.

Part (b): This is comparatively easy, after you do part (a). Your function `g` needs to compute `A`, solve the ODE just like in (a), and subtract 3000 from the final value of z.

Part (c): Also easy. Just wrap `fzero` around your function `g`.

Part (d): Make a function `h` which represents the integrand, and call a quadrature routine on it. Make sure your function expects to calculate a whole vector of values at once. (40)

SOLUTION: **(a)** The function `f` could be written as an inline function, but it would have to be redefined for every θ_0, since the value of A changes. I wrote it as a function file, with global variables. The function `g` cannot be written as an inline function.

```
% main program
% preparations
global c c_prime A
load sound.dat
z_data = sound(:,1);
c_data = sound(:,2);
c = spline(z_data,c_data);
c_prime = fnder(c);

% part (a)
theta_0 = 5.4;
A = cos(theta_0/180*pi)/ppval(c,2000);
[x,z] = ode45(@f,[0,24*6076],[2000;tan(theta_0/180*pi)]);
figure(1);
plot(x,z(:,1)); axis ij;
xlabel('Distance x [ft]','FontSize',20);
ylabel('Depth z [ft]','FontSize',20)
title('Path of Ray Starting at 5.4 Degrees','FontSize',20);

% file f.m
function zprime = f(t,z)
global c c_prime A
zprime = [z(2);- ppval(c_prime,z(1)) / (A^2 * ppval(c,z(1))^3)];
```

The graph looks like the one in the problem statement.

(b) The code is

```
% main program
% part (b)
disp('  theta        g(theta)');
disp('=========================');
for theta = -10:10
```

```
    disp(sprintf('%6d%20.10f',theta,g(theta)));
end

% file g.m
function miss = g(theta_0)
global c c_prime A
A = cos(theta_0/180*pi)/ppval(c,2000);
[x,z] = ode45(@f,[0,24*6076],[2000;tan(theta_0/180*pi)]);
miss = z(end,1) - 3000;
```

The outut is

```
  theta         g(theta)
==========================
   -10     1856.2686164531
    -9      364.7825901716
    -8     -184.5795439291
    -7    -1593.8812418864
    -6    -1312.5696718546
    -5    -1375.4818142731
    -4      514.8950437771
    -3      118.7026361605
    -2      177.2050217320
    -1       90.0176965017
     0      257.9647939904
     1      410.0754596685
     2      653.4141713457
     3      899.9102878074
     4     -919.8538576012
     5     -501.5848728404
     6      491.4011361283
     7      498.3470462548
     8    -1605.0225377687
     9    -1847.7103292791
    10    -1239.2387389827
```

The exact values are highly sensitive to programming details.

(c) After you find the initial values, you have to call `ode45` again to get the actual curves.

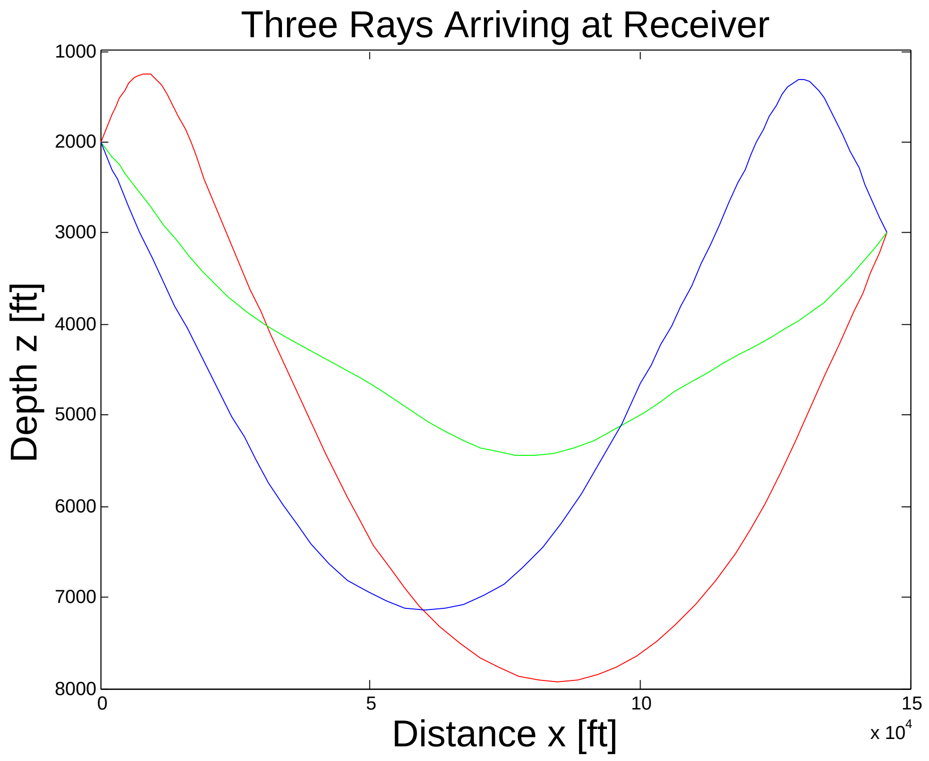
```
% main program
% part (c)
theta_1 = fzero(@g,-8)
theta_2 = fzero(@g,4)
theta_3 = fzero(@g,7)
[x_1,z_1] = ode45(@f,[0,24*6076],[2000;tan(theta_1/180*pi)]);
[x_2,z_2] = ode45(@f,[0,24*6076],[2000;tan(theta_2/180*pi)]);
[x_3,z_3] = ode45(@f,[0,24*6076],[2000;tan(theta_3/180*pi)]);
figure(2);
plot(x_1,z_1(:,1),'r',x_2,z_2(:,1),'g',x_3,z_3(:,1)','b'); axis ij;
xlabel('Distance x [ft]','FontSize',20);
ylabel('Depth z [ft]','FontSize',20);
```

```
    title('Three Rays Arriving at Receiver','FontSize',20);
```

The resulting values are

```
theta_1 =
  -8.269327315709313
theta_2 =
   3.770190205459487
theta_3 =
   7.274330732181878
T =
  29.950496509256187
```

The resulting rays look like this:



(d) The coordinates of the curve are still in x2, z2 from part (c).

```
% part (d)
z = spline(x_2,z_2(:,1));
z_prime = spline(x_2,z_2(:,2));
h = @(x) sqrt(1 + ppval(z_prime,x).^2) ./ ppval(c,ppval(z,x));
T = integral(h,0,24*6076)
```

The result is

```
T =
  29.950496509256187
```

Check: the sound speed varies, but is around 5000 ft/sec. The travel time along the straight line between source and receiver at 5000 ft/sec is about 29.165 seconds. Close enough.

Actually, the travel times along all the rays are within a fraction of a second of one another. That seems surprising at first, but you have to realize that all paths are virtually straight lines, despite what the graphs look like. Look at the scale.