

For each problem that uses Matlab, you should hand in a printout of the relevant script or function file(s), or a transcript of your interactive session (use the `diary` feature), 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.

**Note:** The problems are in increasing order of complexity. It is highly recommended that you do them in order. The earlier problems will prepare you for the later ones.

1. Solve the ODE

$$y'(t) = \frac{y(t)}{t} - \frac{t^2}{y^2(t)},$$
$$y(1) = 1$$

on the interval  $[1, 1.7]$ , using the following methods:

(a) Euler's method with stepsize  $h = 0.05$ .

(b) classical 4th order Runge-Kutta with stepsize  $h = 0.05$ .

(c) 4th order Adams-Bashforth-Moulton (PECE) with Runge-Kutta startup, stepsize  $h = 0.05$ .

(d) the Matlab built-in `ode45`. `ode45` picks its own steps; take the output and use spline interpolation to produce values of the solution at the same points as in (a) through (c). How many steps does `ode45` take?

(e) (optional; 2 points extra credit) Solve the ODE by hand and find the exact solution. Evaluate the exact solution at the same points as in (a) through (d).

You can either print out the results, or plot them (or both).

**Note:** the solution has a mild singularity between 1.35 and 1.4. You will observe that all methods work pretty well up to that point, but then Euler, Runge-Kutta and Adams-Bashforth-Moulton produce garbage. `ode45` will continue to work, at the cost of taking many, many tiny steps near the singularity. This illustrates the point that professionally written subroutines are usually better than what you can write yourself.

(15)

2. Repeat problem 1 (including the extra credit) for the ODE

$$t^2 y''(t) + t y'(t) + (t^2 - 1)y(t) = 0$$
$$y(1) = 1$$
$$y'(1) = 0$$

on the interval  $[1, 2]$ , using a step size of  $h = 0.1$ . You need to convert the equation to a  $2 \times 2$  system of first-order equations first. Print out or plot both  $y$  and  $y'$ .

**Notes:** This solution function does not have a singularity. All methods should work pretty well.

Since you can use virtually the same code for problems 1 and 2, make sure it is correct. Otherwise you might lose credit twice.

**Hint for extra credit:** The solution is a kind of Bessel function. The extra credit consist in finding out exactly which one it is. You can use the relationships

$$\begin{aligned} J_1'(t) &= [J_0(t) - J_2(t)] / 2 \\ Y_1'(t) &= [Y_0(t) - Y_2(t)] / 2 \end{aligned} \tag{15}$$

3. Find the two solutions of the equation

$$e^x = x + 2,$$

using Matlab routine `fzero`.

This is a very short and easy problem, just a preparation for problem 4. (5)

4. Solve the boundary value problem

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

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

$$\begin{aligned} y(0) &= 4 \\ y'(0) &= s \end{aligned}$$

misses the value  $y(1) = 1$ . In other words: Use `ode45` to solve the problem with initial condition  $y'(0) = s$  numerically, and return  $y(1) - 1$ . Use accuracy  $10^{-6}$  for the ODE solver.

(b) Plot `miss(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. (10)

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)

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

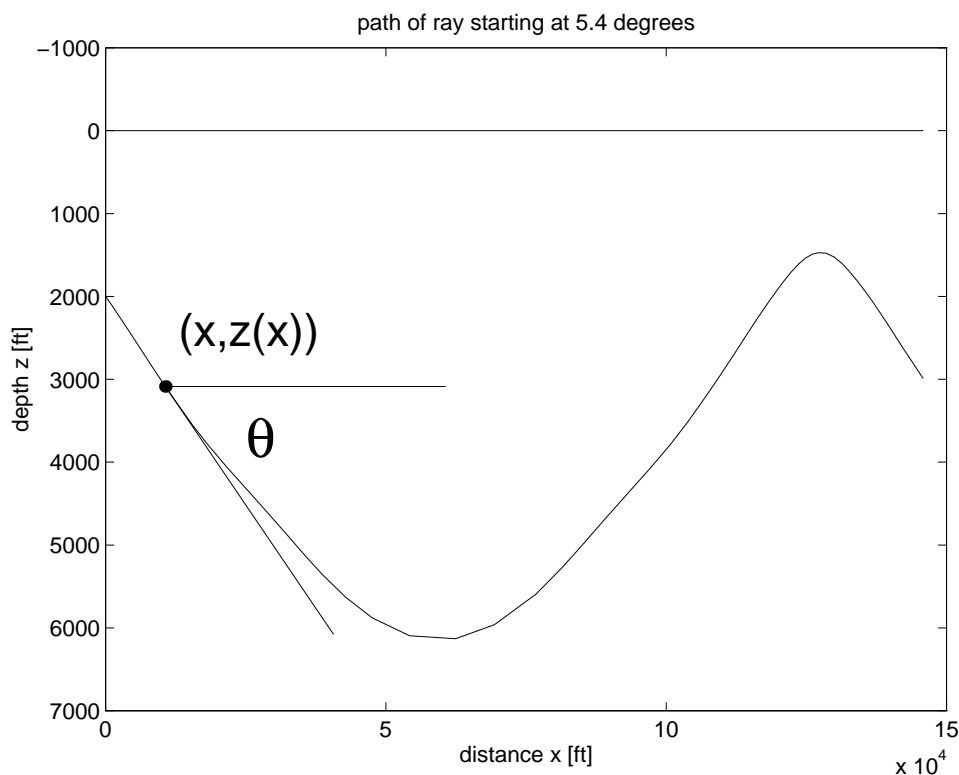
Snell's law states that

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

From this, we can derive the ODE

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

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  $\theta_0 = 5.4^\circ$  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  $x = 24$  miles,  $z = 3000$  ft.

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

Print out a table of values of  $g$  for  $\theta_0$  going from  $-10^\circ$  to  $10^\circ$  in steps of  $1^\circ$ .

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

Print out these values of  $\theta_0$ , and plot the three corresponding rays in one picture.

(d) (5 points extra credit) For the ray from part (c) with initial angle near  $4^\circ$ , 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. I think you will need the spline toolbox. If your Matlab does not have that, you need to use a lab machine somewhere.

2. 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.

3. 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 subroutine `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  $c'(z)$  from scratch every single time, so we compute the spline coefficients once and for all, and then just evaluate them every time. Here is how to do that:

At the main Matlab level, you declare the coefficients as global, load in the data, and compute the spline coefficients:

```
global c_coef cprime_coef
load sound.dat;
c_coef = spline(sound(:,1),sound(:,2));
cp_prime_coef = fnder(c_coef);
```

Routine `fnder` from the spline toolbox does the differentiating. Then, in your subroutine, you evaluate the splines with `ppval`.

```
function z_prime = f(x,z)
global c_coef cprime_coef

c = ppval(c_coef,z(1));
cp_prime = ppval(cprime_coef,z(1));
...
```

The same trick is necessary for the values of  $z'(x)$  in part (d).

Other variables, like the constant `A`, are also computed once and passed as global variables. Every subroutine that expects to access global variables has to have the `global` declaration in it.

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 `quad` or `quadl` on it. Make sure your function expects to calculate a whole vector of values at once. This function will be similar in complexity and in some programming details to the `zdot` from part (a). (40)