CS 130

Problem Set 2

Due: September 25, 2013

General Instructions

- You may opt to accomplish this problem set individually, or with a partner.
- Answer the items completely. Show your solutions for items requiring manual solution (i.e. no programming required).
- If you have consulted references (books, journal articles, online materials, other people), cite them as footnotes to the specific item where you used the resource/s as reference.
- Save your answer sheet as a PDF file. The answer sheet can be made using any word processing program (although preferably, one uses LaTeXto make it).
- Submission of the problem set answers should be done via e-mail. Attach the PDF file, and write as the subject header of the e-mail: [CS 130] $<$ *Student Number* $>-<$ *Last Name, First Name* $>-$ Problem Set 2 if done individually, or [CS 130] $<$ *Comma Separated Student Numbers* $>-<$ *Comma Separated Last Names* $>-$ Problem Set 2 if done as a duo. Send your answers to janmichaelyap@gmail.com.
- If you have any questions regarding an item (EXCEPT the answer and solution) in the problem set, do not hesitate to e-mail me to ask them.

Questions

1. Solve for the respective particular solutions of each of the following ODEs (given the boundary conditions):
   (a) $3x - 2x' - x^3e^{4t} = 0$ where $x(0) = 1$
   (b) $x' - \cos t + x \cot t = 0$ where $x(\frac{\pi}{2}) = \frac{5}{2}$
   (c) $x'' + 4x' + 4x - 12t = 16$ where $x(0) = 0$ and $x'(0) = 4$
   (d) $4x'' + 8x' + 3x - 9t = 6\cos t - 19\sin t$ where $x(0) = -2$ and $x'(0) = 0$

   HINT: When trying to solve for the particular solution of a second order differential equation (i.e. the value of the arbitrary constants in the general solution), solve first for the general solution, then get the first derivative of the resulting equation with respect to the independent variable (which in the cases of the items above, the variable x). Afterwards, plug in the boundary conditions, then you will now have 2 equations (the general solution and the first derivative of the general solution) with 2 unknowns (the two arbitrary constants), which you can now solve using standard algebraic methods.

2. We can actually "solve" for the particular solutions of ODEs using Scilab. For this, we use the *ode* function that is natively available in Scilab. To use this, we first create a Scilab function representing the ODE we are trying to solve. For example, let us solve the first order ODE $x(4-x)\frac{dy}{dx} - y = 0$, subject to the boundary condition that $y = 7$ when $x = 2$. The particular solution to this is $y = 7\left(\frac{x}{4-x}\right)^{\frac{1}{4}}$. To "solve" the first order ODE in Scilab, we first re-express the equation such that the $\frac{dy}{dx}$ term is solely on one side, and some other function is at the other side. Hence, our original ODE when re-expressed is $\frac{dy}{dx} = \frac{y}{x(4-x)}$. We then translate the re-expressed ODE to code as a Scilab function, which we will name as *example*, as such:

```
function dy = example(x,y)
   dy = y / (x * (4 - x))
endfunction
```

An important note about the *ode* function is that it will solve the ODE numerically. Hence, we need to define the range of values the independent variable will take that the *ode* function will use in "solving" the ODE numerically:

```
x = 2:0.01:3
```

The statement above creates a vector containing sequence of (rational) numbers from 2 to 3 inclusive, at an interval (called the step size) of 0.01 in between numbers. We are now ready to "solve" the ODE. The statement to do so is as follows:

```
y = ode(7,2,x,example)
```

The function has four input parameters: the first one is the boundary condition on the dependent variable ($y = 7$); the second one is the boundary condition on the independent variable ($x = 2$); the third parameter is the range of values upon which the function will base its numeric solution; and the fourth is the name of the function representing the ODE. What *ode* returns is a vector of values of the particular solution of the ODE evaluated on the range of values of the independent variable (i.e. x).

To check if the output is "correct" (remember that *ode* solves the ODE numerically rather than analytically/exactly), we then implement the analytic/exact solution as a function and check its value on the range of values defined in x:

```
for i = 1:101
   y_exact(i) = 7 * (x(i) / (4 - x(i)))^(0.25)
end
```

We then make a side by side plot of *ode*'s output and those obtained from the statement above:

```
plot2d(x, [y' y_exact])
```

You should see a plot which is almost linear in shape, and notice that there is only one line, instead of two. This is because the graphs of both the output of the *ode* function and the exact function are almost coincident, and only differ by a small value. This proves that the *ode* does "solve" an ODE, albeit numerically, and the results though not exact are at a certain precision. You could actually plot the outputs one by one separately by merely putting the output of one function instead of augmenting the other one. For example:

```
plot2d(x, y)
plot2d(x, y_exact)
```

To solve second order linear ODEs (with constant coefficients), a change in how we implement the function in Scilab is done. This is due to the fact that the *ode* function can only handle first order ODEs. Consider the ODE $2\frac{d^2y}{dx^2} - 7\frac{dy}{dx} - 4y = 100$, where $y = -26$ and $\frac{dy}{dx} = 5$ when $x = 0$. The particular solution would be $y = e^{4x} - 2e^{\frac{1}{2}x} - 25$. Similar to first order ODEs, we re-express the equation such that the $\frac{d^2y}{dx^2}$ term is solely on one side, and some other function is at the other side. The re-expressed version of our ODE is thus $\frac{d^2y}{dx^2} = \frac{7}{2}\frac{dy}{dx} + 2y + 50$. Next, what we need to do is to re-express (again) the second order ODE as a system of

first order ODEs. To do this, we "split" y into variables y_1 and y_2 such that we have the system of first order ODEs:

$$\frac{dy_1}{dx} = y_2$$
$$\frac{dy_2}{dx} = \frac{7}{2}y_2 + 2y_1 + 50$$

Note that $\frac{dy_1}{dx} = y_2$, and because of this the second equation

$$\frac{dy_2}{dx} = \frac{7}{2}y_2 + 2y_1 + 50 \Rightarrow \frac{d\left(\frac{dy_1}{dx}\right)}{dx} = \frac{7}{2}\frac{dy_1}{dx} + 2y_1 + 50 \Rightarrow \frac{d^2y_1}{dx^2} = \frac{7}{2}\frac{dy_1}{dx} + 2y_1 + 50$$

which is pretty much the same as our original re-expressed second order ODE. Implementing the system of first order ODEs in Scilab, we have:

```
function dy = example2(x,y)
   dy(1) = y(2)
   dy(2) = 3.5 * y(2) + 2 * y(1) + 50
endfunction
```

Using the same x as a while ago, the call to the *ode* function is as such:

```
y2 = ode([-26;5],0,x,example2)
```

Note now that the first input parameter representing the boundary condition for the dependent variable is now a (column) vector. The first entry represents the boundary condition on the dependent variable itself ($y = -26$), while the other is for the derivative of the dependent variable with respect to the independent variable ($\frac{dy}{dx} = 5$). The resulting y now actually has 2 rows: the first row is the particular solution for y_1 evaluated at the values in x, and the second row is the particular solution for y_1 evaluated at the values in x. Note again that $\frac{dy_1}{dx} = y_2$, so the first row is the actual "solution" to our second order ODE (which is what we need), and the second one is the values of first derivative of the second order ODE "solution" evaluated at the values in x. Again, implementing the exact solution and doing a side by side plot:

```
for i = 1:101
   y2_exact(i) = y2_exact(i) = exp(4*x(i)) - 2*exp(0.5*x(i)) - 25
end

plot2d(x, [y2(1,:)' y2_exact])
```

Part of your task is to implement (using Scilab) the functions corresponding to the ODEs solved in Item 1 that will be used as input to the *ode* function. Place in your answer sheet **only the codes representing the ODE functions** as part of the answer to Item 2. Name the functions *fa, fb, fc,* and *fd*, respectively. To test your ODE functions, you may use the template script below:

```
t_a = 0:0.01:1
t_b = %pi/2:0.01:3*%pi/4
t_c = 0:0.01:1
t_d = 0:0.01:%pi/2

x_a = ode(1, 0, t_a, fa)
x_b = ode(2.5, %pi/2, t_b, fb)
x_c = ode([0;4],0,t_c,fc)
x_d = ode([-2;0],0,t_d,fd)
```

Another part of the task is to take a screenshot of the plot of the output of the *ode* function applied to *fa, fb, fc,* and *fd*. Use t_a, t_b, t_c, and t_d as the x-axis in plotting the respective output.