Autumn Quarter 2005
Math. Methods Problem Set 4: ODE's in 1 dimension

October 30, 2005

1 Approach to radiative equilibrium

The temperature of a planet heated by constant Solar radiation S, and subject to cooling by infrared radiation to space, is governed by

$$M\frac{dT}{dt} = S - \sigma T^4 \tag{1}$$

where M is a constant "thermal inertia" factor and σ is the Stefan-Boltzmann constant. Answer the following:

- Find the equilibrium point(s) (steady states) of the system
- Determine the stability of the equilibria
- Find an approximate form of the solution if T is initially much smaller than the equilibrium temperature (but still positive), valid during the time over which the temperature remains small.
- Find an approximate form of the solution if T is initially much larger than the equilibrium temperature, valid during the time before which the temperature gets too close to the equilibrium
- Using a few qualitative sketches, explain what you think the general behavior of the system is like.

Extra Credit: Using partial fractions, find an exact general solution to this system, and discuss its behavior. Here's a hint that should help get you started: You can write

$$\frac{1}{z^4-1} = \frac{A_1}{z-e_1} + \frac{A_2}{z-e_2} + \frac{A_3}{z-e_3} + \frac{A_4}{z-e_4} \tag{2}$$

where the e_j are the four fourth roots of unity and the A_j are constants. The trick to solving this problem without a ridiculous amount of algebra is to multipy both sides by z^4-1 (which is equal to $(z-e_1)(z-e_2)(z-e_3)(z-e_4)$) and then determine the unknown coefficients by noting that the relation must be true *whatever* value of z you plug in. By being clever about what you plug in, you can get the coefficients with relatively little pain. (Hint: try plugging in $z = e_1$, and so forth). In the end, you won't get an expression allowing you to write down $T(t)$, but you will get an expression allowing you to write down $t(T)$, which you can then plot and turn sideways.

2 Condorcet's equation

Discuss the behavior of the solutions to the equation

$$\frac{dP}{dt} = g_o P(1+P) \tag{3}$$

assuming that P is initially positive. What is the approximate form of the solution when $P \ll 1$? How does the system behave when $g_o > 0$? When $g_o < 0$?

3 Logistic equation with time-varying growth rate

Find the solution to the equation

$$\frac{dP}{dt} = g_o(t) P(1-P) \tag{4}$$

where $g_o(t) = g_1 \cdot (1 + \sin \omega t)$

4 Programming Project: A polynomial object

In this exercise, you will develop a class for manipulating polynomials. Let the polynomial P be defined by

$$P(x) = a_o + a_1 x + a_2 x^2 + ... + a_n x^n \tag{5}$$

The polynomial can be defined by a list of coefficients, $[a_0, a_1, ..., a_n]$. Start by defining a class called `poly`, whose creator (the `__init__` method) takes the list of coefficients and stores it in a member called `coeffs`. Next, we add a few useful methods to the class. First, add a `__repr__` method which prints out a polynomial object in a nice way when you type its name. For example, if `myPoly = poly([1.,2.,3.])`, then typing `myPoly` should produce the output `1.  + 2.x + 3.x**2`.

Next, add a `__call__` method which returns the value of the polynomial at a given x. The straightforward way of computing the value by summing the values $a_j x^j$ is very inefficient and also lets a lot of roundoff error accumulate. The right way to evaluate a polynomial on a computer is to rewrite the computation as follows:

$$P(x) = a_0 + x * (a_1 + x * (a_2 + ...)...) \tag{6}$$

For example, we'd write $a + bx + cx^2$ as $a + x(b + cx)$. The way this would look as an algorithm is that we would first evaluate $b + cx$, then multiply by x then add a. Generalize this procedure, and code it up as a function. Make this function into a method of your polynomial object.

Finally, add an `__add__` method which allows you to add two polynomial objects. It should return a polynomial object whose coefficients are the sums of the coefficients of the two operands.

Extra Credit: Develop a `__mul__` method which allows you to multipy two polynomial objects, returning a polynomial object whose coefficients are the coefficients of the product. For example, multiplying `poly([a,b])` by `poly([c,d])` should return `poly([ac,bc+ad,bd])`. Defining object multiplication in this way is so powerful that,if you have done everything right, you should be able to use the `__call__` method you wrote previously, without modifications, to find the coefficients of the composition of two polynomials. In other words, `myPoly(.1)` should return the value of `myPoly` at $x = .1$, but

`myPoly(poly[1.,2.])` should return the polynomial whose coefficients are given by substituting $1 + 2x$ into the polynomial defined by `myPoly`.

Verify that your object is behaving properly, by checking it against some cases where you know what the answer should be. (This request applies to all parts of the problem, not just the extra credit. Doing this sort of check should be such a reflex that I won't always explicitly state it in the future).