Special Functions and Solving the Heat Equation for a Cold Cylinder

Natalie Price-Jones, 999091021

natalie.price.jones@mail.utoronto.ca

4 December 2014

1 Physics Background

In this lab, we will solve the heat equation ($\frac{\partial u}{\partial t} = c \triangle u$ where $\triangle$ is the Laplacian operator) for a solid cylinder. We start with a cylinder at some uniform temperature T_0. Then we hold the environment around the cylinder at a constant temperature $T = T_1 = 0K$, where $T_0 > T_1 > 0K$. Common sense tells us that the cylinder will equilibrate with its surroundings, its temperature decreasing until it reaches uniform $T = 0K$. The temperature change over time and position is described by the heat equation. For simplicity, we will assume our cylinder has infinite length and radius R. This reduces the problem to a single dimension, the radial coordinate s (since our initial temperature profile now has only radial dependence). So we wish to solve the following PDE:

$$\begin{cases} \frac{\partial u}{\partial t} = \frac{c}{s}\frac{\partial}{\partial s}\left(s\frac{\partial u}{\partial s}\right) \, : \, s \leq R, \, t > 0 \\ u(t=0,s) = T_0 \\ u(t,s=R) = 0 \end{cases} \tag{1}$$

where $u(t,s)$ is the temperature as a function of time t and distance from the centre of the cylinder s and c in our case is thermal diffusivity. We have learned a lot of techniques to numerically solve PDEs, but it turns out that Equation 1 can be solved analytically (sort of). If we employ separation of variables by assuming $u(t,s) = T(t)S(s)$, we can reduce it to two ODEs,

$$
\begin{aligned}
\frac{1}{cT(t)}\frac{dT(t)}{dt} &= -\lambda^2 \\
\frac{1}{S(s)}\frac{d^2S(s)}{ds^2} + \frac{1}{s\,S(s)}\frac{dS(s)}{ds} &= -\lambda^2
\end{aligned} \tag{2}
$$

Here, T is a function of only time whose product with S constructs u, not the temperature itself. $-\lambda^2$ is an eigenvalue. The solution for $T(t)$ is simply an exponential, but the solution to Equation 2 is not immediately obvious if you haven't seen it before. If we make a substitution $z = \lambda s$ and rearrange slightly, we see that Equation 2 is a form of Bessel's differential equation of order zero:

$$
0 = \frac{d^2S(z)}{dz^2} + \frac{1}{z}\frac{dS(z)}{dz} + S(z)
$$

The solutions to this equation are order zero Bessel functions of the first ($J_0(z)$) and second ($Y_0(z)$) kind (see Figure 1). So our final solution $S(z)$ should be a linear combination of these two functions.

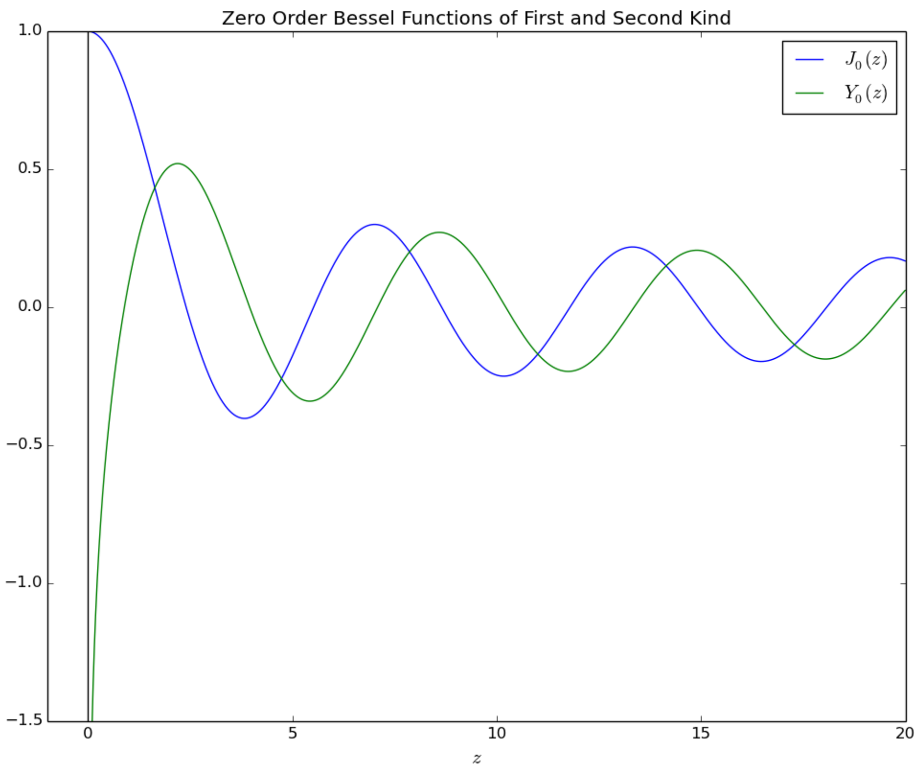

Figure 1: Zero order Bessel functions, created using scipy.special module.

We can see in Figure 1 that $Y_0(z) \to -\infty$ as $z \to 0$. But both the initial and boundary conditions imply $S(s)$ must be finite as $s \to 0$, and since $z \propto s$, we need $S(z)$ finite as $z \to 0$. For the purposes of solving Equation **??**, we discard the $Y_0(z)$ solution. The solution for $S(s)$ will be a linear combination of $J_0(\lambda s)$ for different λ eigenvalues. Using the boundary condition from Equation 1 and the orthogonality properties of the zero order Bessel functions, it is possible to determine a closed form solution to the PDE we have set out to solve. This derivation is shown in Appendix A, but the result is given below:

$$u(t, s) = 2T_0 \sum_{m=1}^{\infty} \frac{e^{-\lambda_m^2 c t}}{\lambda_m R} \frac{J_0(\lambda_m s)}{J_1(\lambda_m R)}, \tag{3}$$

where λ_m is chosen so $J_0(\lambda_m R) = 0$.

Equation 3 isn't in a form we really want when we try to code the result. For one thing, its an infinite series - we may be able to approximate it with sufficient numbers of terms but it would be impossible to find an exact solution. Even worse, the result depends on these Bessel functions, but I've not yet said what these are. In fact these function are very important to many other kinds of calculations, so much so that they are included in the scipy.special module as callable functions.

The Bessel functions have multiple definitions. Since we are concerned with functions of integer order, we will consider the following two:

$$\begin{aligned} J_n(x) &= \sum_{k=0}^{\infty} \frac{(-1)^k}{k!(k+n)!} \left(\frac{x}{2}\right)^{2k+n} && (4) \\ \text{and } J_n(x) &= \frac{1}{\pi} \int_0^{\pi} \cos(n\tau - x\sin(\tau))d\tau && (5) \end{aligned}$$

2 Computational Background

2.a Memoization

2.b Interpolation

3 Lab Instructions

Question 1: One of the biggest challenges in computing Bessel functions using Equation 4 lies in finding factorials effeciently. Python's built in math module provides a function to do this (math.factorial).

a) Write a recursive factorial function and compute factorials for 10,100,500. What happens when you try to find the factorial of 1000? How does the speed of the computation compare with using math.factorial?
b) A common way to approximate large factorials is to use Stirling's approximation, $\ln(n!) \approx n\ln(n) - n$. Can we compute factorial of 1000 with this? How does ln(1000!) compare with math.log(math.factorial(1000))? What about $\ln(1\times10^5!)$ compared with math.log(math.factorial(1×10^5))? Which computation takes longer?
c) Another technique that can be used to improve factorial computation is memoization, described in the Computational Background. Using memoization write a new recursive function that computes the a factorial. Use this new function, find the factorial (in order) of 10, 100, 500. Then try to use it to find 1000!. What happens if you restart your Python interpreter and use the function to find 1000! ? With this in mind, explain one advantage and one disadvantage of this method of computing factorials.

Question 2:
a) Simplify Equations 4 and 5 for the case where $n = 0$.
b) Find $J_n(x)$ from $x = 0$ to $x = 20$ with step size 0.01 using Equation 4 with 10 terms. Plot this on the same axes as scipy.special.jn (with labels). Do the same for 50 and 100 terms.

Question 3: A necessary component of Equation 3 is knowing for which λ_m $J_0(\lambda_m R) = 0$. Solving for eigenvalues λ_m is equivalent to finding the zeros of the Bessel function and dividing them by R.
a) Use any of the zero-finding methods of Lab 4 to find the first five zeros of J_0 (keep in mind J_0 has multiple zeros). Use the J_0 you found in question 2. Translate these to eigenvalues λ_m. Compare your results with the output of scipy.special.jn_zeros and adjust your accuracy until you get a match to ***********X************ significant figures. What accuracy was needed to reproduce scipy's result?
b) Although Bessel functions are not periodic, the location of their nth zeros can be approximated for sufficiently high n, using $\pi(n - 1/4)$. Use this to approximate the location of the 200th, 300th and 400th zeros. Compare your result with scipy.special.jn_zeros. Is this a good approximation?

A Deriving a Closed Form Solution to the Heat Equation

We have $S(z) = S(\lambda s) \sim J_0(\lambda s)$. The boundary condition in Equation 1 requires that our choice of λ must satisfy $J_0(\lambda R) = 0$. However we saw in Figure 1 that Bessel functions are oscillatory, and so there are infinite choices for λ that satisfy this requirement. If we denote them λ_m then we know $u(t, s)$ is a linear combination over all possible eigenvalues.

$$
\begin{aligned}
S_m(s) &= c_m J_0(\lambda_m s) \\
\text{and } T_m(t) &= T_m(0) e^{-\lambda_m^2 c t} \\
\implies u(t, s) &= \sum_{m=1}^{\infty} a_m e^{-\lambda_m^2 c t} J_0(\lambda_m s) \\
\text{since } u_m(t, s) &= S_m(s) T_m(t) \text{ and } u(t, s) = \sum_{m=1}^{\infty} u_m(t, s)
\end{aligned}
$$

Our λ_m are implicitly defined by $J_0(\lambda_m R) = 0$, so we are left with determining the coefficients a_m.

With this in mind, note the orthogonality property of zero order Bessel functions of the first kind:

$$
\int_0^R s\, J_0(\lambda_m s) J_0(\lambda_n s) ds = \delta_{mn} \frac{R^2}{2} J_1^2(\lambda_m R) \tag{6}
$$

where we have introduced the first order Bessel function of the first kind J_1. We will also need another property of Bessel functions:

$$
\int_0^x x' J_0(x') dx' = x J_1(x). \tag{7}
$$

This is easily verified by substituting Equation 4 in for J_0 and J_1.

We know our solution must satisfy $u(t = 0, s) = T_0$, so we have:

$$
T_0 = \sum_{m=1}^{\infty} a_m J_0(\lambda_m s)
$$

If we multiply this expression on each side by $s\, J_0(\lambda_n s)$ and integrate, we can exploit Equation 6 to isolate for a_m.

$$
\begin{aligned}
a_m &= \frac{2}{R^2 J_1^2(\lambda_m R)} \int_0^R T_0 s\, J_0(\lambda_m s) ds \\
a_m &= \frac{2}{R^2 J_1^2(\lambda_m R)} \frac{T_0 R\, J_1(\lambda_m R)}{\lambda_m} \\
a_m &= \frac{2T_0}{\lambda_m R\, J_1(\lambda_m R)}
\end{aligned}
$$

where we have used Equation 7 to go from the first to the second step.

Thus we find that:

$$
u(t, s) = \sum_{m=1}^{\infty} \frac{2T_0}{\lambda_m R\, J_1(\lambda_m R)} e^{-\lambda_m^2 c\, t} J_0(\lambda_m s),
$$

which matches Equation 3 above.