Solving A Partial Differential Equation

Dan Wilson

23 December 2016

1 Introduction

The aim of this program is to reproduce Dr Olsen-Kettle's graph for the temperature distribution around a nuclear waste rod, as showcased in her lectures [1]. The problem first involves simplifying a 4 dimensional heat equation into a 1 dimensional matrix equation to be solved using linear algebra solvers given in the GNU Scientific Libraries.

2 Theory

We begin with the 4 dimensional heat equation

$$\frac{1}{\kappa}\frac{\partial T}{\partial t}(r,t) - \bigtriangledown^2 T(r,t) = S(r,t) \tag{1}$$

which describes the temperature increase due to storage of nuclear rods which release heat due to radioactive decay, where the source term is

$$S(r,t) = T_{rod}\text{exp}(-t/\tau_0)/a^2 \tag{2}$$

for $r \leq a$, and zero elsewhere, where a is the radius of the rod.

This problem can be transformed into a 1 dimensional problem given the circular symmetry of the system, and by assuming that the rod is very long with respect to the distance from the rod that we are investigating.

Using the implicit backwards euler method [1], the 1D PDE can be solved, producing the matrix equation

$$A\vec{T}^{k+1} = \vec{T}^{k} + \kappa\Delta t\vec{S}^{k} + \vec{b} \tag{3}$$

where k is an integer representing a time step of Δt, A is a tridiagonal $N \times N$ matrix where N is the number of discrete spaces, and b is a constant. For $N = 3$,

$$A = \begin{bmatrix} 1+s+s/2j & -s-s/2j & 0 \\ -s+s/2j & 1+2s & -s-s/2j \\ 0 & -s+s/2j & 1+2s \end{bmatrix}$$

where j is the row number and s is the gain parameter $s = \kappa\Delta t/\Delta r^2$.

The (1,1) matrix term $1+s+s/2j$ comes from the discrete Neumann boundary conditions, where $T_0^k \approx T_1^k$. The discrete Dirichlet boundary conditions are contained within $\vec{b}$, where $T \to 300K$ as $r \to \infty$

3 Method

The code uses the GNU Scientific Library function `gsl_linalg_solve_tridiag()` to extract $\vec{T}^{k+1}$ from $A\vec{T}^{k+1}$ in (3), allowing the newly found temperature to be substituted back into the right hand side of equation (3), and then increment forwards in time again.

The solving function requires 5 vector arguments, all of which must be defined as `gsl_vector`, allocated with `gsl_vector_alloc()`, and set with `gsl_vector_set()`. These arguments are:

- diag: an N element vector containing the diagonal elements of the matrix A
- superdiag: an N-1 element vector containing the super-diagonal elements of the matrix A
- subdiag: an N-1 element vector containing the sub -diagonal elements of the matrix A
- b : an N element vector containing all the elements on the right hand side of (3)
- temp: an N element vector containing the temperature at each discretised node, to be updated

`gsl_vector_view` can be used to define the non-temperature arrays, however this creates a temporary array, and so would have to be re-allocated at every time step. Thus the permanent `gsl_vector` is used instead, and must be freed at the end of the program. These are defined within the struct 'Matrix'.

The time is checked every time step, and when the time becomes 1/100, 1/10, 1/2 or 1 times the total time (as defined by T1, T2, T3 and T4), the temperatures for all N locations are stored one of the columns of the array 'storedTemp[4][N]'. This program is highly optimised for plotting only desired time periods, and so only stores 4 time periods. This is to prioritise efficiency over the flexibility of output data. This is then plotted over the distances $0 + dr \geq r \leq R - dr$, where R is the maximum distance from the rod.

4 Results and Discussion

Figure 1 shows the temperature decay with distance from the rod for a total time of 100 years, over a distance of 1m. The values used to plot Figure 1 are in very strong agreement to those of Dr. Olsen-Kettle's matlab solution. The difference between the values calculated for each method is 1×10^{-11}, and thus agree to 14 significant figues, as seen in Table 1.

One cause for the difference between the two results is the numerical method used to solve the '$Ax = b$' matrix equation. Dr. Olsen-Kettle's Matlab solution uses a general in-built matrix division with the use of the backslash operator, whereas the C solution used the GNU Scientific Library linear algebra solver for tridiagonal matrices.

For a square $N \times N$ matrix, which we have, the backslash operator computes the equation by Gaussian elimination with partial pivoting. It has been shown that it is advantageous not to perform pivoting on certain structures of matrices, including some class of tridiagonal matrices, given that the process produces larger errors than more optimised methods [3]. Matlab does in fact contain a function for solving tridiagonal matrix equations, and using this may have yielded more agreeable solutions to the C/GSL program.

Another reason for the difference between the two results is that double precision floating point numbers are typically given to 1517 significant decimal digits. Thus given that the accuracy of the result is approaching this boundary, it is likely that rounding errors, on either the Matlab or C/GSL solution are responsible.

Table 1: A table showing the difference in temperature at $r = dr$ after 1 year for the matlab solution [1] and the C/GSL solution.

	Temperature at r = dr (K)	
No. of years	**Matlab**	**C/GSL**
1	300.905257376016	300.905257376006

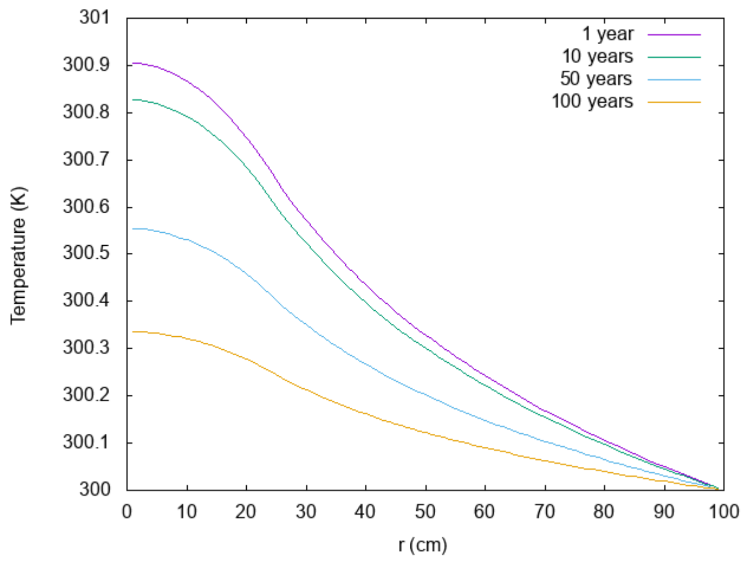

Figure 1: Graph showing the temperature decay near the nuclear rod using Backward Euler method. The dashed line represents the Matlab solution, and the solid line represents the C/GSL solution.

5 Conclusion

By condensing the 4-dimensional heat equation into a simple 1-dimension NxN matrix equation, the problem has been solved numerically, as done by Dr Olsen-Kettle [1]. The solutions from Matlab and C/GSL agree to 14 significant figures, where the difference could arise given that they are solved numerically using different algorithms, or due to rounding errors of double precision rounding errors. The Gaussian elimination with partial pivoting algorithm used in the Matlab solution has been shown to produce larger errors for tridiagonal matrices [3] than with optimised algorithms, as is used in the C/GSL program [2].

References

[1] `http://espace.library.uq.edu.au/view/UQ:239427/Lectures_Book.pdf`, page 42, (accessed 23/12/16).

[2] `https://www.gnu.org/software/gsl/manual/html_node/Tridiagonal-Systems.html#Tridiagonal-Systems` (accessed 23/12/16).

[3] `http://www.maths.manchester.ac.uk/~higham/narep/narep173.pdf`, page 11, (accessed 23/12/16).