How hot is my hot chocolate? Partial differential equations for an ideal thermos

Henrik Åhl
in collaboration with Denhanh Huynh

November 9, 2015

Department of Astronomy and Theoretical Physics, Lund University

Project supervised by Tobias Ambjörnsson

LUND UNIVERSITY

1 Introduction

Partial differential equations link the simultaneous changes in several variables to the ultimate change in the corresponding function itself. The amount of applications are of course numerous, but in particular partial differential equations, or henceforth *PDE's*, can be used to describe the change in heat throughout liquids, gases and solids. The so called *heat equation* links all these together through the identity

$$\frac{\partial T}{\partial t} - \nabla^2 u = 0$$

where $T = T(\mathbf{r}, t)$. In this report we will limit ourselves to the one-dimensional case, so that $\mathbf{r} = x\hat{x}$, and focus on the numerical calculation of how the heat spreads in an open, ideal thermos.

2 Solving the heat equation numerically

2.1 Description of the problem

Jonathan is out on a warm winter day skiing, having brought his new futuristic super-thermos *ThermaHold*^{TM} [1] with him. Imsy-wimsy Jonathan however, off onto new adventures, manages to forget his thermos left open in the snow. How long will it take before Jonathan's thermos full of hot chocolate has turned cold? What if Jonathan has added his daily nutrient pill to the hot beverage, effectively raising the diffusivity? What if the whole pill does not dissolve completely?

The explicit euler method is used for calculating the increment due to steps in time. This method is chosen because of the simplicity and the merits with respect to the computation time. No matrix inversions are for example needed, as tends to be the case in the Crank-Nicholson method. The explicit euler method is however of second order in space and first order in time, whereas the Crank-Nicholson algorithm renders a result that is on par in space and in second order in time (due to using the second derivative to approximate the function). However, the lack of matrix operations required for the explicit euler case outweighs the benefits of a more accurate algorithm. The same argument applies for the implicit euler method, as a matrix inversion is required in that case. In short: a more complicated model is simply not needed for our problem. The greatest limitation of the explicit method is simply mainly the restriction in the relation between the time and space step – a hindrance the implicit methods both avoid.

[1] Patent pending.

2.2 A numerical approximation

Knowing from pre-university mathematics, the first derivative of a function can be roughly approximated by the slope between two points along the curve. For a function T that is, it means that the first derivate at a point j would approximately equal, through the forward difference method,

$$\frac{\partial T_j}{\partial x} \approx \frac{T_{j+1} - T_j}{a} \tag{2.1}$$

for some step size a, when the dimension $\hat{x}$ is discretized so that $x_j = ja$. This can be done recursively, since the same reasoning applies to the second derivative, so that it resultingly can be written as

$$\frac{\partial^2 T_j}{\partial x^2} \approx \frac{T_{j+1} - 2T_j + T_{j-1}}{a^2}, \tag{2.2}$$

here instead using the backward difference.

This all implies that the diffusion equation can be written as a system of first-order ordinary differential equations, such that

$$\mathbf{A} = \frac{D}{a^2} \begin{pmatrix} \ddots & \ddots & \ddots & & 0 & \\ & 1 & -2 & 1 & & \\ & & 1 & -2 & 1 & \\ & 0 & & \ddots & \ddots & \ddots \\ & & & & & \end{pmatrix} \tag{2.3}$$

Discretizing with respect to time as well, i.e. stating that $t_n = nh$ for some constant step size in time h, one can solve the resulting equation using the *explicit euler method* through realizing that

$$\frac{\mathbf{T}^{n+1} - \mathbf{T}^n}{h} = \mathbf{A}\mathbf{T}^n \tag{2.4}$$

for all $\mathbf{T}^n$ such that $\mathbf{T}^n = \mathbf{T}(t_n)$. This method is stable for all solutions as long as $h \leq a^2/2$.

Because of the one-sided boundary conditions (see section 2.3), the last value was calculated via a backward step instead of a forward one, so as to limit the value at the bottom. Correspondingly, the first value of $\mathbf{u}$ was set to correspond to the direct outerward boundary condition. From a practical perspective, this allows for direct computation of the vector $\mathbf{u}^{n+1}$ via matrix multiplication.

The computation speed for the system ought furthermore scale quadratically with time due to the quadric nature of the matrix $\mathbf{A}$.

2.3 Simulation settings and boundary conditions

The simulations were performed assuming that the thermos is completely isolated due to the high tecnological capabilities of a *ThermaHold*^{TM}, aside from the open top. The simulations use the diffusivity constant for water at $T = 25°\text{C}$, $D = 0.14 \cdot 10^{-6}$ m^2/s [2], and boundary temperatures

$$T(x = 0, t) = 0 \ °\text{C}$$
$$T(x, t = 0) = 90 \ °\text{C}$$

as well as step sizes $a = 1.0 \cdot 10^{-3}$ m and $h \lesssim 5.0 \cdot 10^{-6}$ m. The size of the thermos is set to 0.30 m.

3 Results and conclusions

The results prove that indeed, the temperature spread throughout the depth of the thermos is apparent. Due to the extremely high functionality however, which is to say the very ideal conditions, the case based on the realistic heat diffusivity constant D is extremely slowly developing, resulting in a final mean temperature of roughly the initial one, as can be seen in fig. 4. The reader should here note that time is given in the numbers of hours passed since the start of the simulation, whereas the position value corresponds to the relevant cell (due to the discretization).

Figure 1 shows, however, the intended dynamics of the model, which is that the surface-aligned cell will decrease in temperature the fastest, bringing on a slower, but steady development for the after-coming cells.

Notably, the spread over time towards the cells lying deeper inside the thermos is visible in fig. 1, as the slope in the x–y plane is evident. The effect is even clearer when the diffusion constant is changed somewhat, to a value on the order 10^-1, as is visualized in fig. 2.

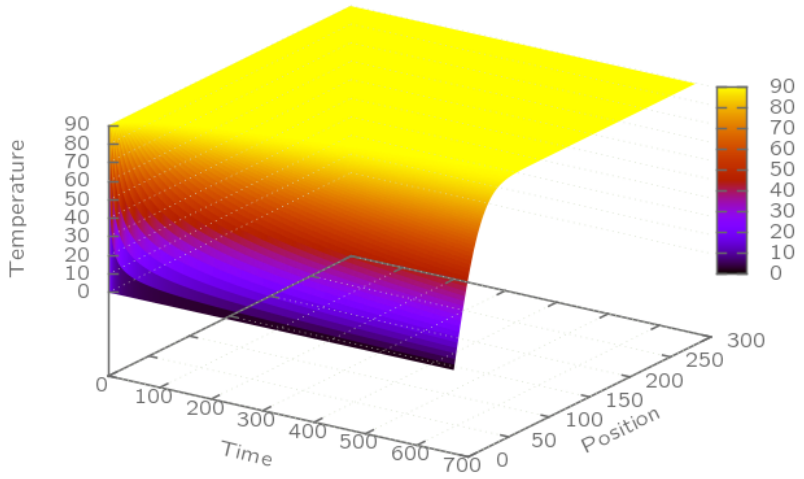

Figure 1: Temperature development with respect to space and time. Positions are given in the cell number inherent to the discretization, i.e. in this case the step parameter j.

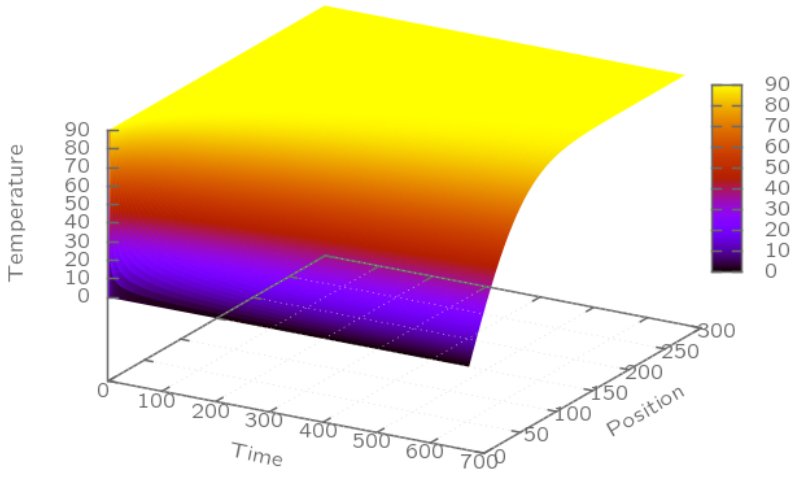

Figure 2: Temperature development with respect to space and time. Positions are given in the cell number inherent to the discretization, i.e. in this case the step parameter j.

When the model is adjusted so that the diffusitivity constant is increased by a value on the order of 10^6 times the original value, the mean temperature in the thermos decreases significantly more rapidly, although it still takes about four full days for the mean temperature to rougly reach the outside temperature. Although this implies issues with our model, it signifies the affectance the ideal conditions have on the outcome. The temperature development on the other hand, as can be seen in fig. 3, shows a much stronger bias towards time than towards position over the course of the first days, which does not conflict with the expectations for the outcome markedly.

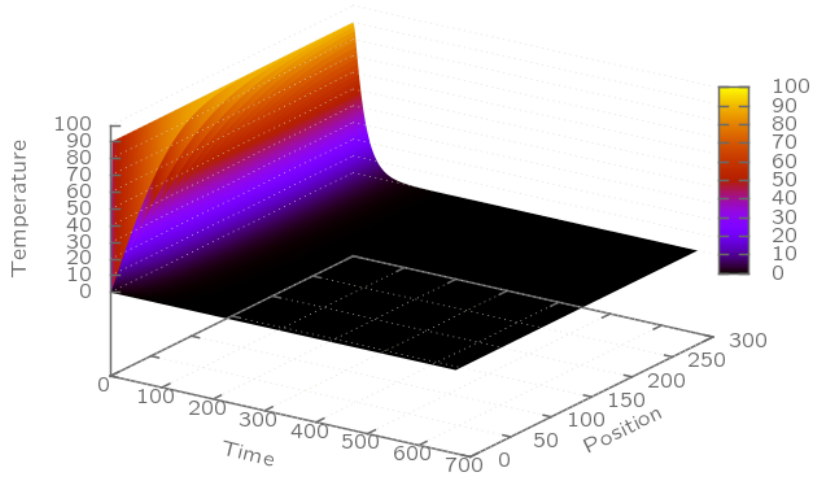

Figure 3: Temperature development with the heat diffusivity raising pill added to the mixture. Compare with the mean temperate over time in fig. 4.

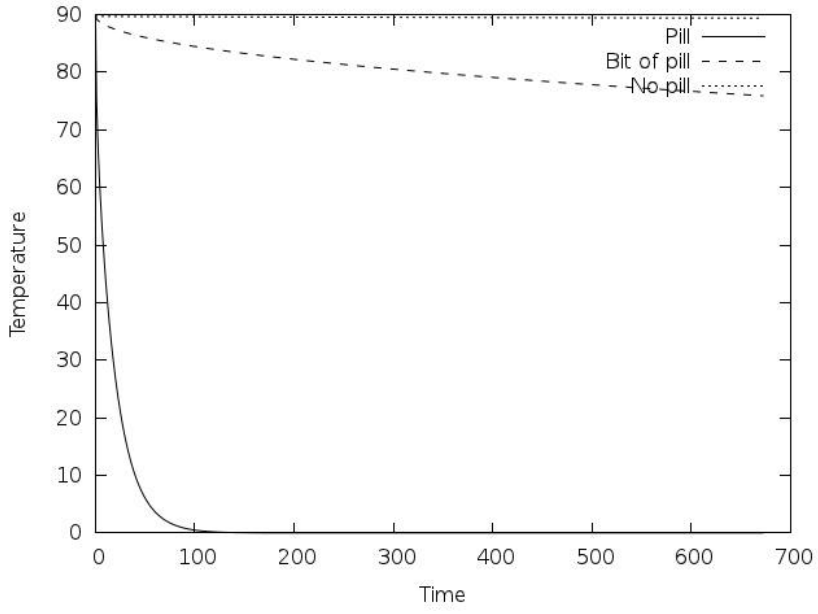

Figure 4: Mean temperature over the course of a total of four weeks. Note that the "Pill" labels signify different diffusion constants from the original one.

In conclusion, our model has showed the significant impacts idealization can have on the outcome of simulations. The results do not provide realistical data, but can be of interest to understand the dynamics of heat diffusion in a near-ideal case.

References

[1] Tobias Ambjörnsson, *Lecture notes, Computational Physics*, Department of Astronomy and Theoretical Physics, Lund University, 2015.

[2] J. Blumm, A. Lindemann, *Characterization of the thermophysical properties of molten polymers and liquids using the flash technique*, High Temperatures – High Pressures 35/36 (6) 2007.

A Code

```
package pde;

public class Thermos
{
   private static final int ITERATIONS = 3600 * 24 * 7 * 4; //
      Iterate over four weeks
   private static final int OUTSIDE = -0; // Temperature outside
      of thermos
   private static final int INSIDE = 90; // Initial chocolate
      temperature
   private static final int N = 300; // Number of cells
   private static final double H = 1 / 2000001.; // Step length in
      time
   private static final double D = 0.14e-2; // Heat constant
   private static final double LENGTH = 0.30; // Length of thermos
      in m
   private static final double A = LENGTH / N; // Resulting step
      length in space
   private static double[] T; // Temperature vector
   private static double[][] matrix; // Computing matrix

   public static void main(String[] args)
   {

      matrix = new double[N][N];
      double matrixconst = H * D / (A * A);

      // Set first row
      matrix[0][0] = 1;
      for (int i = 1; i < matrix[0].length; i++)
         matrix[0][i] = 0;

      // Set rest of matrix (but last row)
      for (int i = 1; i < matrix.length - 1; i++)
         for (int j = 0; j < matrix[i].length; j++)
            if (j == i - 1 || j == i + 1)
               matrix[i][j] = matrixconst;
            else if (i == j)
               matrix[i][j] = 1 - 2 * matrixconst;
            else
               matrix[i][j] = 0;
```

```
      // Set last row
      for (int i = 0; i < matrix.length - 3; i++)
         matrix[N - 1][i] = 0;
      matrix[N - 1][N - 3] = matrixconst;
      matrix[N - 1][N - 2] = -2 * matrixconst;
      matrix[N - 1][N - 1] = matrixconst + 1;

      // Set boundary conditions
      T = new double[N];
      T[0] = OUTSIDE;
      for (int i = 1; i < T.length; i++)
         T[i] = INSIDE;

      int count = 0;
      while (ITERATIONS - count >= 0)
      {
         timeStep();
         if (count++ % 3600 == 0)
         {
            System.out.print(count / 3600 + "\t"); // Only print
               hours
            // Print data
            for (int j = 0; j < T.length; j++)
               System.out.print(T[j] + "\t");
            System.out.println();
         }
      }
   }

   // Calculates the change in temperature over a step in time
   public static void timeStep()
   {
      double[] copy = new double[N];
      for (int i = 0; i < matrix.length; i++)
         for (int j = 0; j < matrix[i].length; j++)
            copy[i] += matrix[i][j] * T[j];
      T = copy;
   }

}
```