Investigating and Computing Heat Diffusion

A MATH235 University of Wollongong Project

Andrew Morris

Autumn 2011

Contents

Chapter 1

Introduction

Heat diffusion is a broad and heavily researched phenomenon. The aim of this project has not been to tread new ground by exploring some obscure fringe of the subject. Rather, it has been to experiment, gain personal understanding, and create an accessible overview of the area through curiosity driven investigation.

This project is at least as much about the computer science aspects of implementation and efficiency as it is about mathematics. As such, this report shall contain a considerable discussion on how the programming language C++ was used to simulate heat diffusion and produce images like figure 1.1:

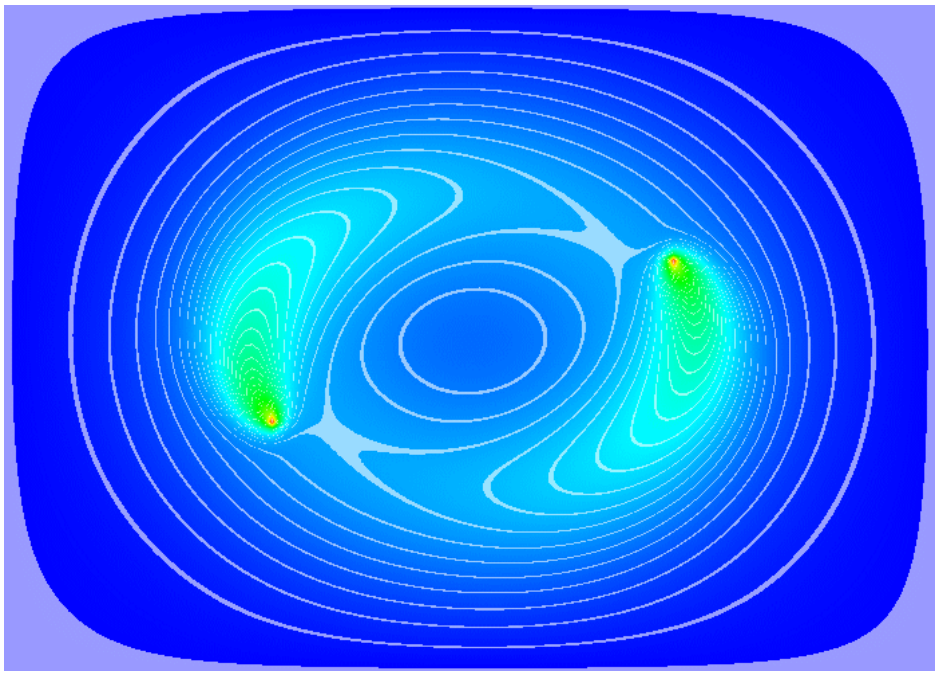


Fig 1.1: an example of a simulated temperature profile.

Chapter 2

Background

The central equation of heat diffusion that this project revolves around is:

$$\frac{dT}{dt} = \kappa \frac{d^2T}{dx^2} \qquad \text{- Eq 2.0.1}$$

This is the heat diffusion equation, also more simply called the "heat equation". But first, why does this describe heat diffusion? Why is it the way it is? To gain a conceptual understanding, we first go back to basics.

2.1 | Newtonian Cooling

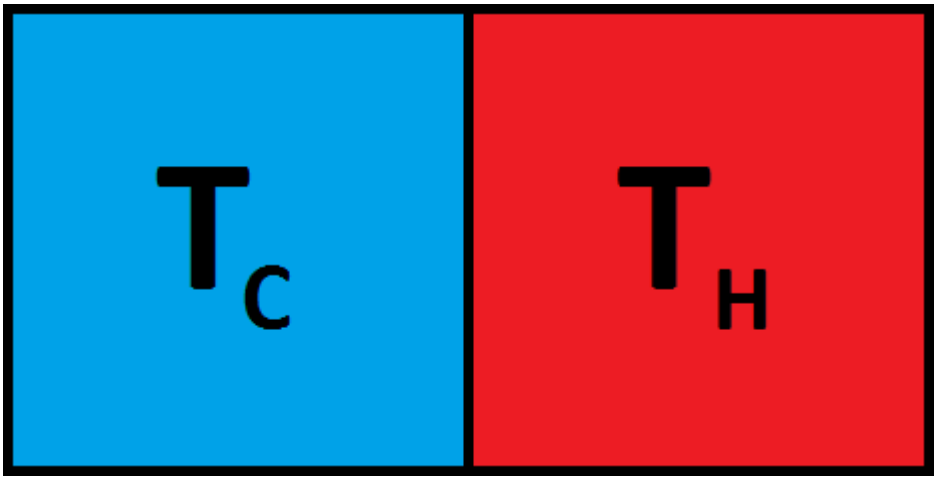


Fig 2.1.1: A Newtonian cooling system with one hot and one cold object.

We all know that heat flows from hot to cold. Figure 2.1.1 is a diagram of an abstract situation where two blocks are in thermal contact. Heat can flow between the blocks, but, for simplicity, it cannot flow *within* the blocks. Both blocks have homogenous temperatures. If desired, one could imagine that inside each block is a fluid which is efficiently mixed and agitated so that any internal temperature fluctuations can be neglected.

This process follows *Newtonian Cooling*. The amount of heat transfer is proportional to the difference between the temperatures of the two objects in thermal contact. This agrees with basic intuition; heat transfer is much more rapid when touching a hot frying pan than a warm phone charger.

The equations describing the heat transfer in this abstract situation contain nothing other than this concept:

$$\frac{dT_C}{dt} = -\lambda(T_C - T_H) \qquad \frac{dT_H}{dt} = -\lambda(T_H - T_C)$$

- Eq 2.1.1, 2.1.2

These equations can be solved to give the following (full solution in appendix A.2):

$$T_C = \frac{1}{2}\big(T_C(0) + T_H(0)\big) + \frac{1}{2}\big(T_C(0) - T_H(0)\big)e^{-2\lambda t} \quad \text{- Eq 2.1.3}$$

$$T_H = \frac{1}{2}\big(T_H(0) + T_C(0)\big) + \frac{1}{2}\big(T_H(0) - T_C(0)\big)e^{-2\lambda t} \quad \text{- Eq 2.1.4}$$

This means both converge to the average of the two temperatures (which is the steady-state solution) with an exponential decay.

2.2 | Continuous Heat Diffusion

Now we're in a position to give the heat equation a conceptual explanation:

$$\frac{dT}{dt} = \kappa \frac{d^2T}{dx^2} \qquad \text{- Eq 2.0.1}$$

The left side of this equation should already make sense – the whole point here is to describe temperature change. The right side is less obvious – *what's the story with that second order derivative?* Putting this equation into words we get something like:

The rate of temperature change at a point is proportional to the concavity of temperature across space at that point.

So why concavity? Take a look at figure 2.2.1:

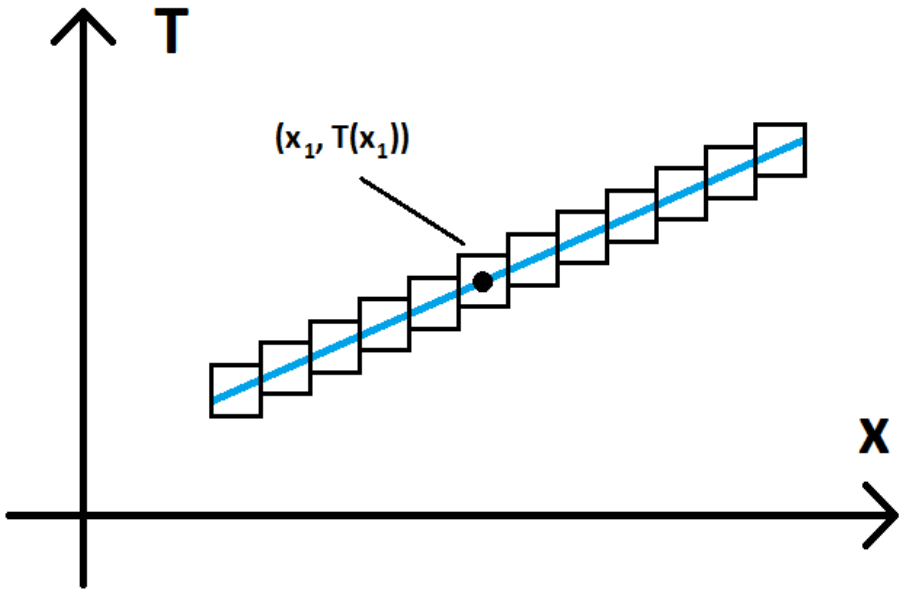


Fig 2.2.1: A linear temperature profile overlayed with an illustration of the discrete model.

Here we have zero concavity – across space temperature changes in a linear fashion. The result is zero overall heat diffusion (except the ends; if this extended infinitely then there would be no diffusion anywhere). Looking at the temperature at x_1, we see a higher temperature on its right. However, on its left there is a lower temperature, but the difference is exactly opposite. This makes the heat transfer exactly opposite, so that all heat flowing in from the right is exactly matched by the heat flowing out to the left.

Let's add in some concavity:

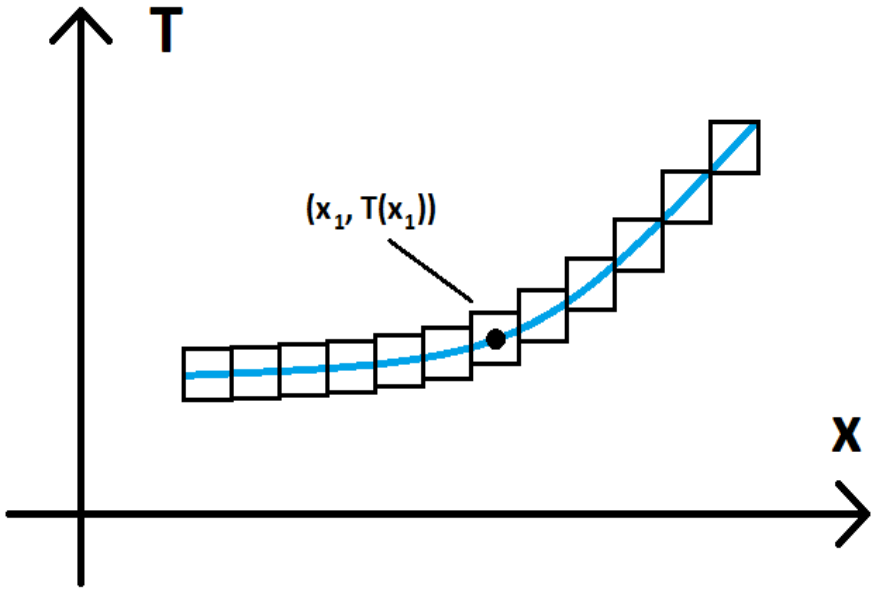


Fig 2.2.2: A non-linear temperature profile overlayed with an illustration of the discrete model.

This is the key. Once again at x_1, there is a hotter temperature on the right and a colder one on the left. However, the difference on the right is greater than the difference on the left. This means the heat flowing in is greater than the heat flowing out, so that overall, temperature increases. Concavity is what describes this imbalance necessary for temperature change, and this is why the second spatial derivative appears in the heat equation.

Newtonian cooling is discrete heat diffusion. Continuous heat diffusion is analogous; when discretised in space, Newtonian cooling is recovered. It is important to note that while Newtonian cooling is an approximation, continuous heat diffusion, as described by equation 2.0.1 (the heat equation), is also an approximation.

In the real world (or perhaps just a closer approximation to the real world), heat diffusion takes place on the atomic scale, where temperature is expressed by vibration. While using a relatively small number of thermal blocks and Newtonian cooling means the temperature resolution is too low, continuous heat diffusion goes too far the other way, making these blocks infinitesimal.

This means thinking about Newtonian cooling as an approximation to continuous heat diffusion is a little misguided. Technically, both approximate real heat diffusion, and the continuous model is just a *far* better approximation.

2.3 | Finite Differences

This project is mostly concerned with simulating the diffusion process. Derivatives are something of an abstract concept that don't directly reveal a numerical scheme. To overcome this, the heat equation is discretised by replacing the derivatives with difference operators. The result is a finite difference equation.

In doing this, both the time and spatial derivatives can be discretised at the same time. However, the spatial derivatives will instead be discretised first, to demonstrate that continuous heat diffusion is analogous to Newtonian cooling (more detail in appendix A.3):

$$\frac{dT}{dt} = \kappa \frac{d^2T}{dx^2} \qquad \text{- Eq 2.0.1}$$

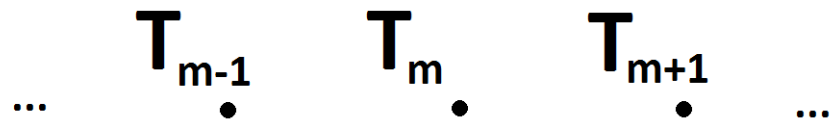
$$\frac{dT}{dt} = \kappa \frac{\Delta^2 T}{\Delta x^2}$$

$$\frac{dT_m}{dt} = -\frac{\kappa}{\Delta x^2}[(T_m - T_{m+1}) + (T_m - T_{m-1})] \qquad \text{- Eq 2.3.1}$$

$$\frac{dT_m}{dt} = \frac{\kappa}{\Delta x^2}(T_{m+1} - 2T_m + T_{m-1}) \qquad \text{- Eq 2.3.2}$$

In G. D. Smith's *Numerical Solution of Partial Differential Equations*, the discretised heat equation appears in the form of equation 2.3.2[1], albeit with different notation and time discretised. This obscures the conceptual understanding that continuous diffusion is being approximated by Newtonian cooling. Conceptual understanding is hindered further; as the book goes on to depict the method as involving infinitesimal points with some elusive theoretical connection. In this project the model is identical but is presented from a different perspective – those points are really divisions of the whole object into small masses. The only real difference between equation 2.3.1 and the equations from section 2.1 is that Newtonian cooling is now occurring with two other objects instead of one.

Smith's *Numerical Solution of PDEs*



This Project

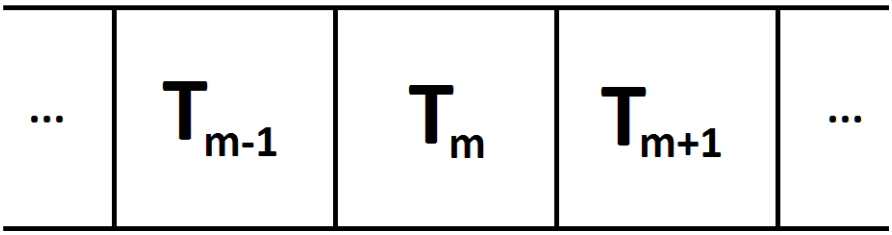


Fig 2.3.1: the conceptual difference between using discrete points and boxes.

Time must now be discretised to complete the process (see appendix A.4 for more detail):

$$\frac{dT_m}{dt} = -\frac{\kappa}{\Delta x^2}[(T_m - T_{m+1}) + (T_m - T_{m-1})] \quad \text{- Eq 2.3.1}$$

$$\frac{\Delta T_m}{\Delta t} = -\frac{\kappa}{\Delta x^2}[(T_m - T_{m+1}) + (T_m - T_{m-1})]$$

$$T_m^{(n+1)} = T_m^{(n)} - \frac{\kappa \Delta t}{\Delta x^2}\left[\left(T_m^{(n)} - T_{m+1}^{(n)}\right) + \left(T_m^{(n)} - T_{m-1}^{(n)}\right)\right] \quad \text{- Eq 2.3.3}$$

This project has actually focused on two-dimensional heat diffusion, so I will switch to these versions of the equations and introduce some new notation at this point. The derivation is in appendix A.4.

$$T^{(n+1)} = -\lambda \Delta t\left[\left(T^{(n)} - N_N^{(n)}\right) + \left(T^{(n)} - N_S^{(n)}\right) + \left(T^{(n)} - N_E^{(n)}\right) + \left(T^{(n)} - N_W^{(n)}\right)\right]$$

- Eq 2.3.4

$$T^{(n+1)} = (1 - 4\lambda \Delta t)T^{(n)} + \lambda \Delta t \sum N^{(n)} \quad \text{- Eq 2.3.5}$$

Here, N stands for *neighbour*. In two dimensions, each temperature cell is in thermal contact with four neighbours. These neighbours can be enumerated using the subscripts N, S, E, W, i.e. north, south, east and west. $N^{(n)}$ can be thought of as the set of these four neighbours, and the uppercase sigma, as per convention, means a summation of the elements in that set. The subscript for T has also been dropped; in this context it stands for any particular temperature cell. The arrangement is illustrated in figure 2.3.2.

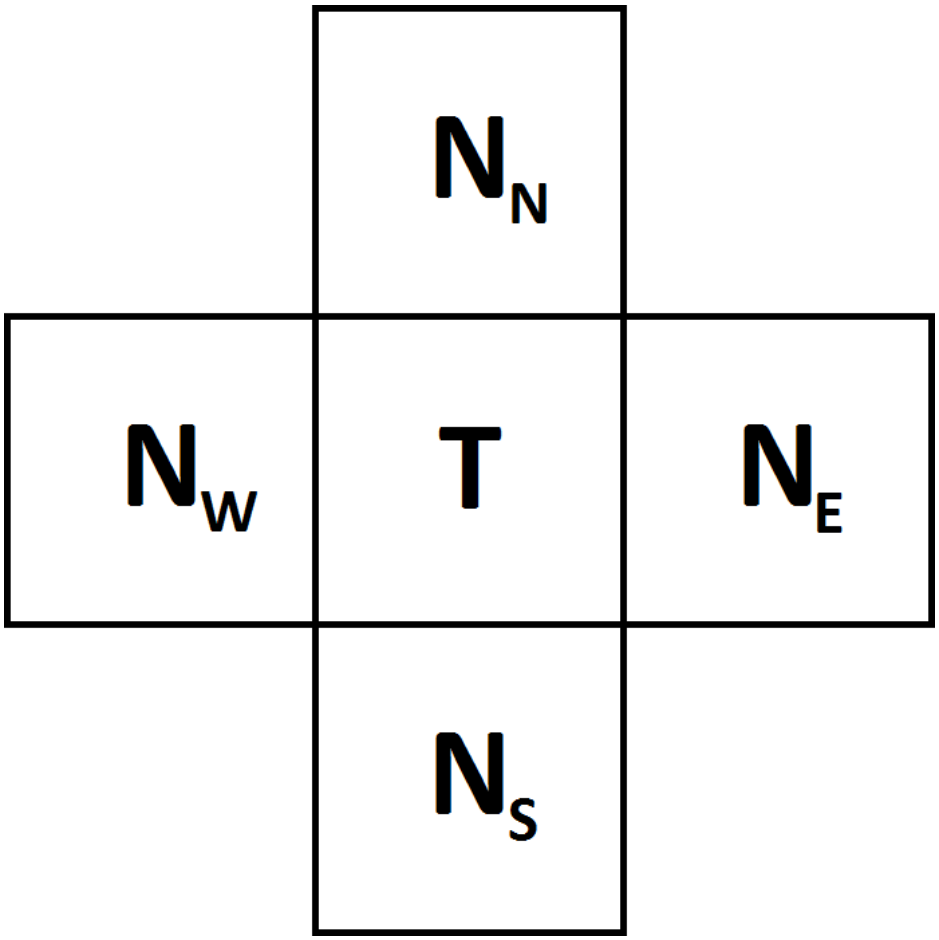


Fig 2.3.2: The neighbours of a temperature cell in 2D in the neighbour/N notation.

This notation takes advantage of what I call *relative subscripting*. I have found it very useful in identifying symmetry and taking advantage of it more seamlessly. More details are in appendix A.1.

Equation 2.3.5 is just the Forward-Euler discretisation. There are also the Partial Crank-Nicholson and Partial Backward-Euler discretisations. These are derived in appendices A.4, A.5, A.6.

$$T^{(n+1)} = (1 - 4\lambda\Delta t)T^{(n)} + \lambda\Delta t \sum N^{(n)}$$

- Forward-Euler, Eq 2.3.5

$$T^{(n+1)} = \frac{1 - 2\lambda\Delta t}{1 + 2\lambda\Delta t}T^{(n)} + \frac{\lambda\Delta t}{1 + 2\lambda\Delta t}\sum N^{(n)}$$

- Partial Crank-Nicholson, Eq 2.3.6

$$T^{(n+1)} = \frac{1}{1 + 4\lambda\Delta t}T^{(n)} + \frac{\lambda\Delta t}{1 + 4\lambda\Delta t}\sum N^{(n)}$$

- Partial Backward-Euler, Eq 2.3.7

An important feature of these schemes is that they are all of the following form:

$$T^{(n+1)} = xT^{(n)} + y\sum N^{(n)} \quad \text{- Eq 2.3.8}$$

With the following constraint:

$$x + 4y = 1 \quad \text{- Eq 2.3.9}$$

This means that all these schemes can be represented by a matrix equation, with a permutation matrix P:

$$T^{(n+1)} = PT^{(n)} \quad \text{- Eq 2.3.10}$$

Since y is a function of x, this means each scheme can be represented by the form of its x value as a function of λΔt. For example, the Partial Crank-Nicholson discretisation is wholly described by equation 2.3.11:

$$x = \frac{1 - 2\lambda\Delta t}{1 + 2\lambda\Delta t} \quad \text{- Eq 2.3.11}$$

Conceptually, this means that for each time step, the numerical scheme replaces each temperature with a weighted average of itself and its neighbours. x is the weight for the central temperature; the amount of itself it keeps, and y is the amount it pulls in from each neighbour.

Chapter 3

Implementation

3.1 | Core

The core of each program is a class – HeatModelRod, HeatModelPlate and HeatModelCube for 1D, 2D and 3D simulations respectively. Each model stores the temperatures, the x and y values (mSelfWeight, mNeighbourWeight) and the dimensions. Each has a Step() function which advances the model by one timestep. Figure 3.1.1 shows the Step() function from heatModelPlate.cpp:

```
47 void HeatModelPlate::Step()
48 {
49     mTemps.Flip();
50
51     std::vector<double>& curr = mTemps.Curr();
52     std::vector<double>& last = mTemps.Buf();
53
54     std::size_t iEnd = mVoxelHeight - 1;
55     for (std::size_t i = 1u; i != iEnd; ++i)
56     {
57         std::size_t jStart = i * mVoxelWidth + 1u;
58         std::size_t jEnd = jStart + mVoxelWidth - 2u;
59
60         for (std::size_t j = jStart; j != jEnd; ++j)
61         {
62             curr[j] = mSelfWeight * last[j] + mNeighbourWeight * (
63                 last[j - 1] +
64                 last[j + 1] +
65                 last[j - mVoxelWidth] +
66                 last[j + mVoxelWidth]);
67         }
68     }
69 }
```

Fig 3.1.1: The core code of the 2D simulation.

As shown in figure 3.1.1, the temperatures are stored in a vector, and each node's neighbours are found arithmetically, rather than being explicitly associated with the node.

The class FlowGraph (available in the source code) was written to abstract the specific way the nodes are connected from the calculation. FlowGraph has a nested class, Node. A FlowGraph is setup by declaring a number of nodes and specifying the connections between them. Each connection has a weight, so that on each timestep each Node's temperature would be replaced with a weighted average of the nodes it is connected to. Unfortunately switching to FlowGraph resulted in an increase of execution time by a factor of approximately 3.5, and that was deemed unacceptable.

It's certainly possible some improvement could be made to FlowGraph, but the message is clear – abstraction is not necessarily free.

3.2 | Colouring

Colouring is handled by the function object class TempColorConverter.

```
#ifndef TEMP_COLOR_CONVERTER_HPP
#define TEMP_COLOR_CONVERTER_HPP

class TempColorConverter
{
private:
    double mMinTemp;
    double mMaxTemp;

public:
    TempColorConverter(double minTemp, double maxTemp)
    :
        mMinTemp(minTemp),
        mMaxTemp(maxTemp)
    {
    }

    Color operator()(double temp) const
    {
        // (Code trimmed. See source code.)
    }
};

#endif // TEMP_COLOR_CONVERTER_HPP
```

Fig 3.2.1: The TempColorConverter class; a function object which implements temperature colouring.

The choice of colouring scheme is hard-coded (not out of necessity), so if you want to use one other than contours, you'll have to uncomment/comment the appropriate segments of code. A few different colouring schemes were experimented with.

The simplest colouring scheme is a simple conversion and scaling of temperature to hue values. The minimum temperature is given the colour blue (cold) and the maximum temperature is given the colour red (hot), with linear interpolation around the larger segment of the colour wheel in between. All colours are fully saturated with 50% luminosity (100% would make them white). Temperatures colder than the minimum are made black and temperatures hotter than the maximum are made white.

Another scheme is like the one above except the colour palette is restricted to 10 colours. This makes the movement of temperature a little easier to see as the movement of the boundary between two colours can be observed. However, it removes a lot of information.

The scheme hard-coded in the supplied source code is one which implements contour lines. This is achieved by treating 50 evenly distributed narrow bands of temperature differently, turning their luminosity up to 80%. It is otherwise identical to the first scheme.

3.3 | Tools

The source code contains 2,500 lines. The core calculation represents only a small proportion of this. A significant amount of code is dedicated to the several supporting tools in the software. The tools abstract a variety of tasks away from the code which is more closely associated with the simulation. One of them is TempColorConverter from the previous section. The rest are detailed here.

FileParser

This class handles the parsing of the configuration files. The nice thing about it is that when a configuration is malformed it tells you exactly where the problem is. For example, trying to parse the configuration in figure 3.3.1 for steelRodImager produces an error:

```
 1 Left temperature = 20C
 2 Right temperature = 60C
 3 Default temperature = 0C
 4 Rod length = 1m
 5 Length step = 0.00125m
 6 Time length = 43200
 7 Time step = 0.1s
 8 Time steps per pixel = 600
 9 Thermal diffusivity = 0.0000042m2s-1
10 CSV print times = 0s, 3600s, 7200s, 10800s
```

Fig 3.3.1: A malformed configuration file.

```
andrew@lenovo-server:~/dev/cpp/heatTransfer$ steelRodImager
steelRodConfig.txt:7:1: Expected 's' while parsing string "s"
```

Fig 3.3.2: Output from the FileParser when attempting to use the configuration file from figure 3.3.1.

The length of time is supposed to be in seconds, and the FileParser class is saying that the ‘s’ was missing at the end of “Time length”.

BufferPair

This is a template class which stores a pair of values which can be efficiently swapped. The two values are stored in a two-length array and an internal index is swapped instead of the actual data. This is used by each core model to store the temperature data. On each time step the temperature profiles are swapped and the current temperature profile is calculated from the swapped out temperatures.

```
#ifndef BUFFER_PAIR_HPP
#define BUFFER_PAIR_HPP

template <typename T>
class BufferPair
{
private:
    T mData[2];
    int mCurrBuffer;

public:
    BufferPair() : mCurrBuffer(0) { }

    T& Curr() { return mData[mCurrBuffer]; }
    const T& Curr() const { return mData[mCurrBuffer]; }

    T& Buf() { return mData[!mCurrBuffer]; }
    const T& Buf() const { return mData[!mCurrBuffer]; }

    void Flip() { mCurrBuffer = !mCurrBuffer; }
};

#endif // BUFFER_PAIR_HPP
```

Fig 3.3.2: BufferPair class.

ProgressBar

The ProgressBar class was created before this project, and makes it easy to display a progress bar one the command line while each program runs.

```
andrew@lenovo-server:~/dev/cpp/heatTransfer$ selfWeightAnalysis
Simulating diffusion 100% [======================]
Calculating x values  30% [======|               ]
```

Fig 3.3.3: Example output from the ProgressBar class.

ProgressBar requires exclusive use of the output stream during its operation. It is constructed by specifying the output stream (intended to be standard output or standard error), the total number of progress points, the width of the bar and whether the bar should be cleared on completion. During the calculation for which the user wishes to display progress, progress points are added to the ProgressBar using the AddProgress() method.

SimpleImageMaker

This class takes care of creating a portable pixmap (*.ppm) image. It makes the process as simple as possible rather than attempting to offer any image manipulation features. After the SimpleImageMaker is constructed with the filename, height and width, all the user needs to do is input the colour of each pixel by repeatedly calling AddPixel().

```
#include <cassert>

#include "simpleImageMaker.hpp"

#include "color.hpp"

SimpleImageMaker::SimpleImageMaker(
    const std::string& fileName,
    int height,
    int width)
:
    mStream(fileName.c_str(), std::ios::binary),
    mHeight(height),
    mWidth(width),
    mPixelsWritten(0)
{
    assert(height > 0);
    assert(width > 0);

    mStream << "P6" << std::endl;
    mStream << width << ' ' << height << std::endl;
    mStream << 255 << std::endl;
}

SimpleImageMaker::~SimpleImageMaker()
{
    assert(mPixelsWritten == mHeight * mWidth);
    mStream.close();
}

void SimpleImageMaker::AddPixel(Color c)
{
    mStream << c.ToASCII();
    ++mPixelsWritten;
}
```

Fig 3.3.4: Implementation of the SimpleImageMaker class.

3.4 | Optimisation

The simulation software is not interactive in nature. The user sets the configuration, executes the program, and waits a noticeable and sometimes very long time for it to finish. There is no spare time to spend on computation while waiting for input. This means any improvement in speed is valuable because it will be used to either reduce waiting times or run a more accurate simulation.

Speed was a consideration throughout development and tests were almost always timed. Many design choices were made in favour speed, such as the heat model classes being separate for 1D, 2D and 3D, instead of being covered by a single generalised model class.

Anything non-essential resulting in a noticeable decrease in performance was either fixed or thrown out. The FlowGraph discussed in section 3.1 was the most extreme example. Another example was the ProgressBar, which was initially consuming far too much time (approx. 4 seconds) because it would clear and recreate the progress bar on the screen every time progress was updated. This was changed so that now the progress bar is updated a maximum of 1000 times, and consumes approximately 0.06 seconds on my machine.

Current performance could be improved by taking more advantage of symmetry, but this has to be done on a case-by-case basis as symmetry takes a variety of forms in different situations. A large improvement could also be made by threading the core execution, sharing the computation between multiple CPU cores.

Compiler optimisation was also utilised. The GNU C++ compiler (g++) offers many optimisation options, of which I used the following:

```
-O3 -fexpensive-optimisations -fomit-frame-pointer
```

I first tried these early in development, when one of the first working versions of steelRodImager was completed. The result was and still is a staggering nine-fold increase in performance.

Chapter 4

Findings

4.1 | Example Images

The following is a collection of example images from the simulation software developed for this project.

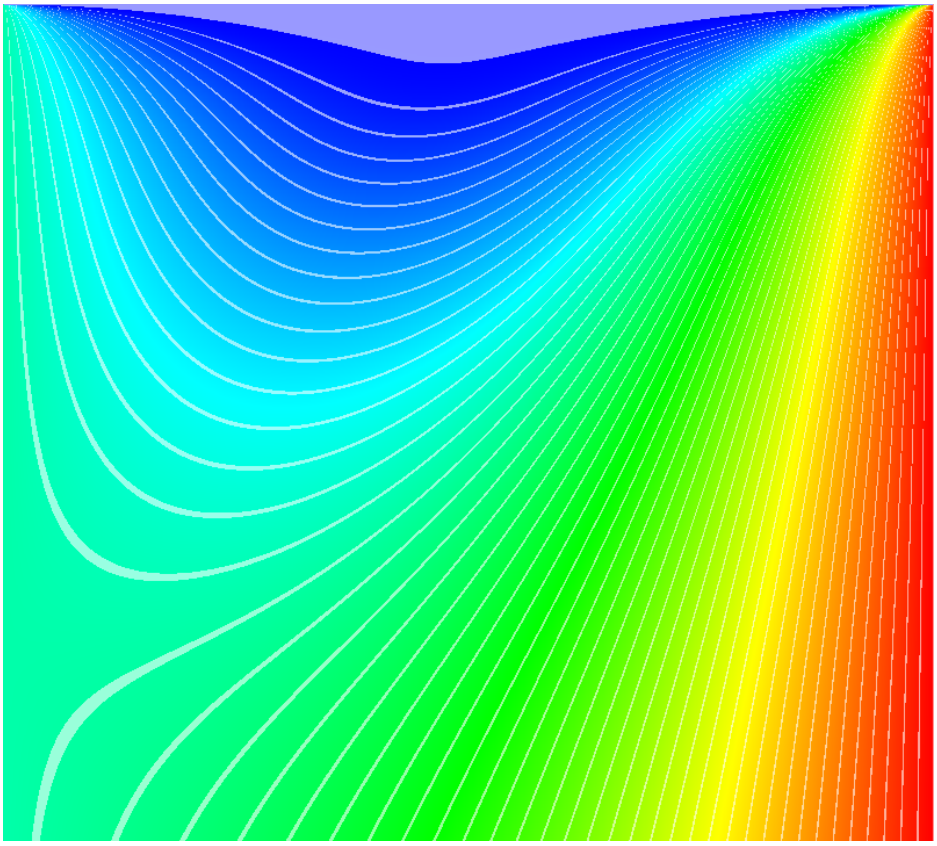


Fig 4.1.1: Time series of the temperature profile of a steel rod.

Figure 4.1.1 is a representation of a steel rod one metre long. Initially, its temperature is 0°C but its left end is held at 20°C and its right end at 60°C. Each line of the image represents the profile of the rod at a point in time. The total time shown is 12 hours.

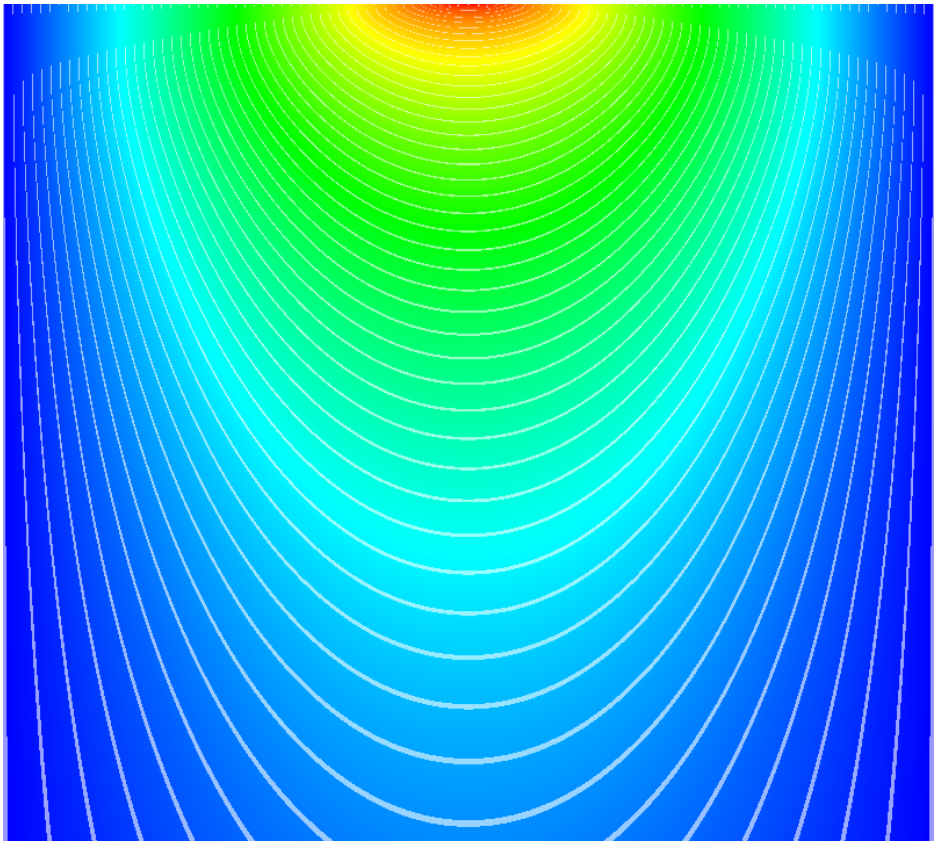


Fig 4.1.2: Symmetrical time series of a steel rod; only the left half was simulated.

Figure 4.1.2 also represents a rod one metre in length. The simulation time is also 12 hours. However, its initial temperature is 0°C at the ends, 60°C in the middle and linearly interpolated in between. The left-right symmetry was exploited to speed up its calculation by a factor of two. This was achieved by simulating only the left half. The right-most node (which is in the middle of the image) was placed only in thermal contact with its left neighbour, since its right neighbour would always be equal to it, and so no heat would ever transfer there.

It turns out that the choice of diffusivity and scale are fairly arbitrary simulation choices. Changing the diffusivity is the same thing as re-scaling time, and changing the size is like using a magnifying glass and re-scaling in time. For this reason the 2D examples all have the same configuration for these two attributes – they are steel plates 0.7m wide and 0.5m tall. Each also has the outer edges fixed at 0°C, unless stated otherwise.

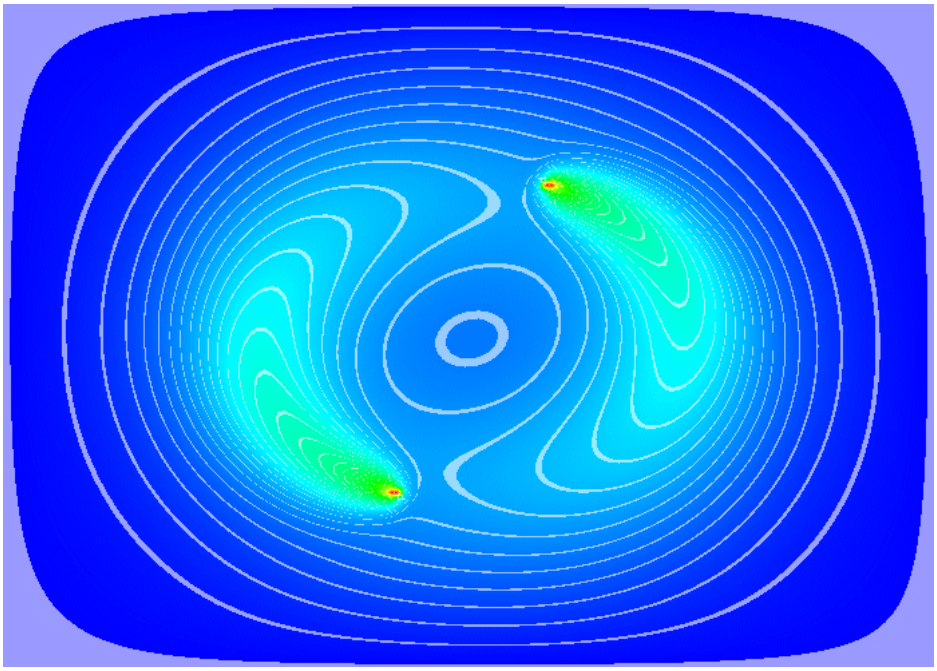


Fig 4.1.3: Temperature profile of a steel plate with two rotating hot points.

Figure 4.1.3 illustrates a snapshot after 21 minutes, 40 seconds. The plate is initally 0°C uniformly and has two 60°C points which rotate anticlockwise at a speed of approx. 0.007 radians per second. Trails of diffusing heat from each rotating point are clearly visible. In video, contours can be seen contracting and disappearing in the centre, indicating the rising temperature there.

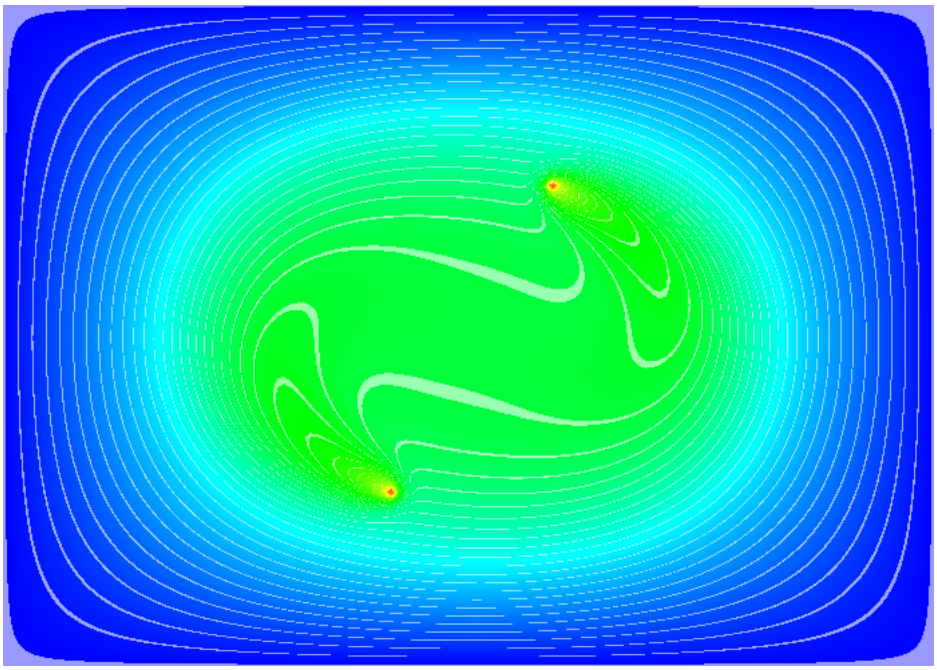


Fig 4.1.4: Steady-state temperature profile from the same system as above.

Figure 4.1.4 is an image from the same simulation as Figure 4.1.3, but is taken at approx. 16.5 hours. After this much time, the image approximates one profile from the dynamic steady state of the system, where the temperature distribution is periodic as the hot points rotate.

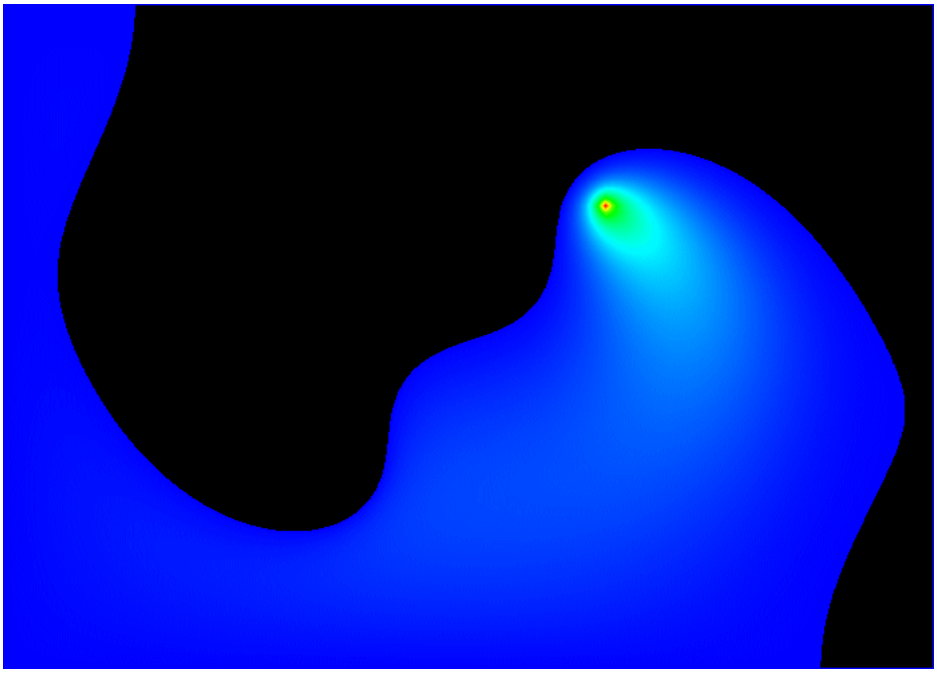


Fig 4.1.5: Temperature profile demonstrating rotation symmetry. Half the profile is below the minimum temperature.

Figure 4.1.5 is a variation of the previous two examples. Instead of having two 60°C points stirring up heat on an initially 0°C plate, the plate begins at 30°C and has one rotating point at 60°C and the other at 0°C. The colouring is also different, ranging from 30°C to 60°C, which leaves half of the image black. This illustrates the system's 180° rotation symmetry which is also present in the other models with two rotating points. It would be possible to exploit this symmetry and speed up the calculation by approximately a factor of two, but this was not implemented.

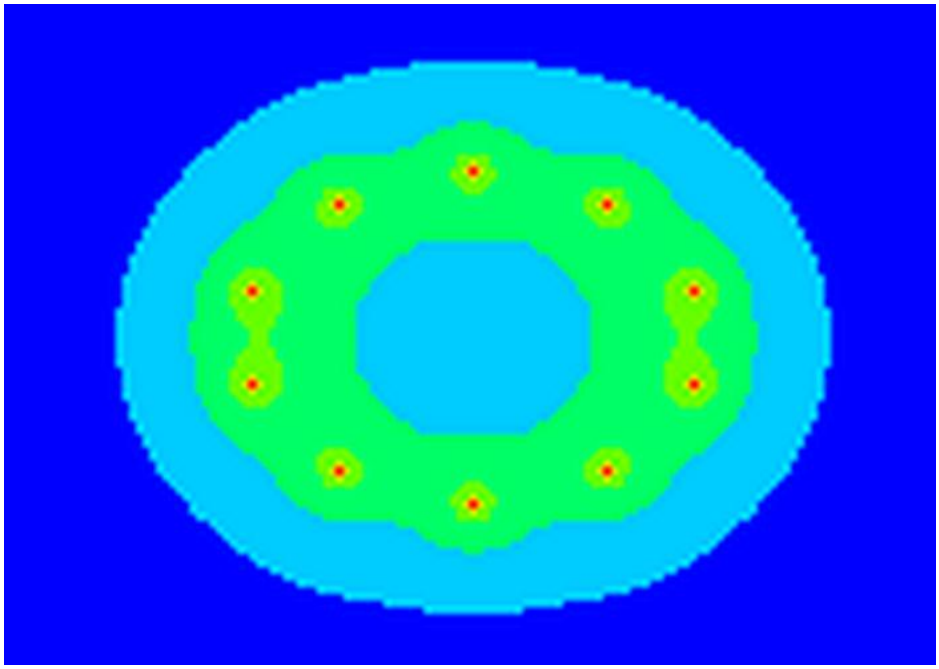


Fig 4.1.6: Temperature profile of a steel plate with 10 hot points.

Figure 4.1.6 is the result of 16 minutes, 50 seconds of simulation, starting at a uniform temperature of 0°C and with ten fixed 60°C points. It has a blotchy appearance, because it is a recreation of an image produced before linear interpolation was implemented. Instead, an image where each pixel represents a node in the calculation is produced first, and then the image was enlarged by a factor of five with 3rd party software (PhotoFiltre[5]). The disadvantage of this is that the colour boundaries are aliased according to the original pixels, which are enlarged to 5x5 blocks.

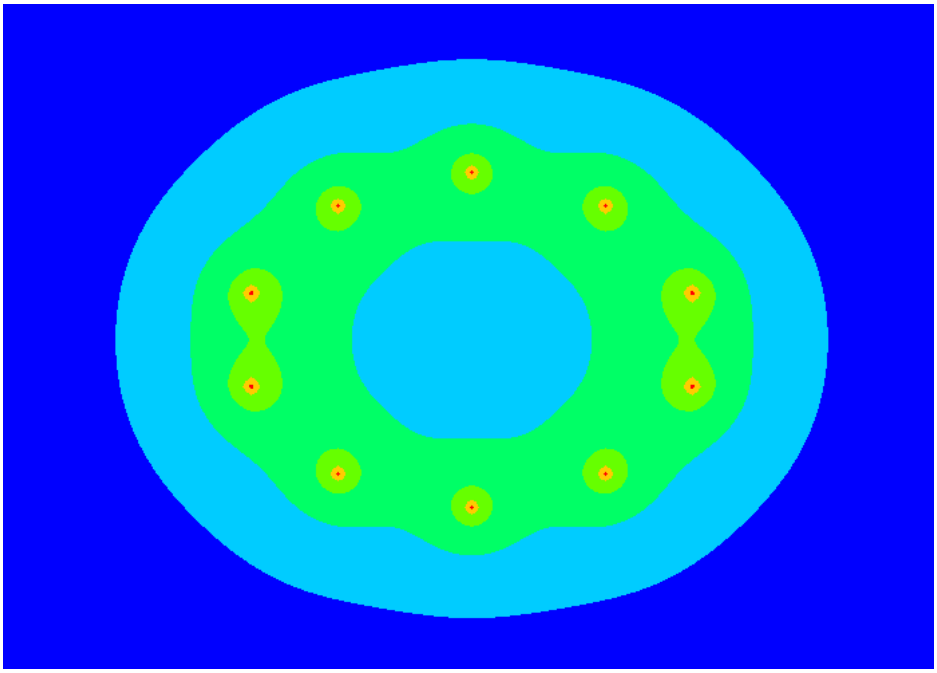


Fig 4.1.7: Temperature profile of a steel plate with 10 hot points.

Figure 4.1.7 is the same as figure 4.1.6 except that linear interpolation of the temperature is performed before the colour scheme is applied. This allows a smooth illustration of temperature boundaries, avoiding a blotchy appearance.

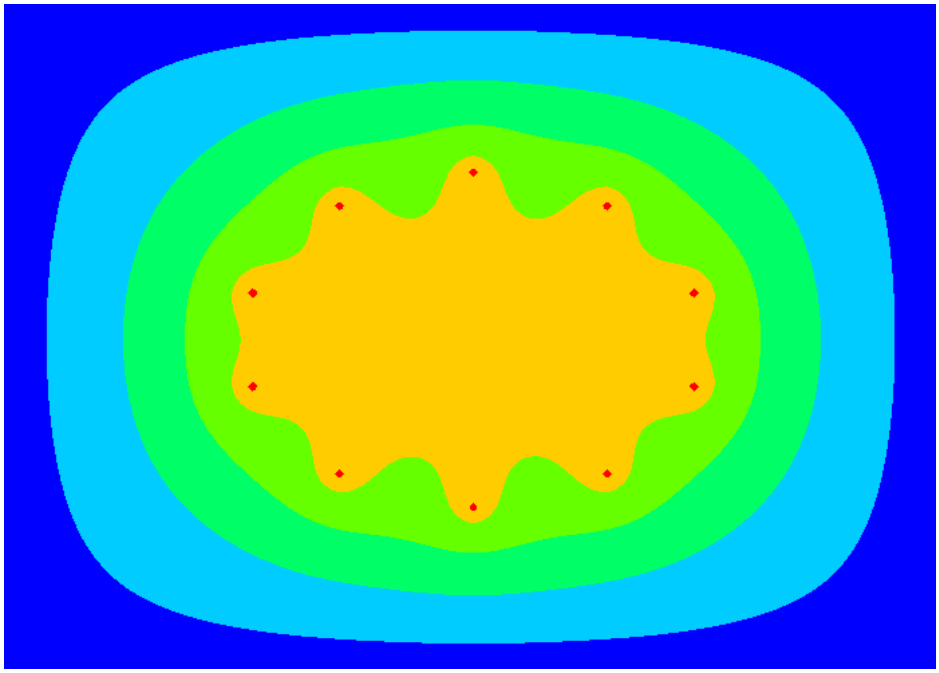


Fig 4.1.8: Steady state profile of the same system in figure 4.1.7.

Figure 4.1.8 is from the same simulation as 4.1.7 but after approx. 16.5 hours, approximating the steady state (probably identical to the accuracy of the colour scheme). Between each fixed 60°C point, the cool influence of the fixed 0°C edges can be seen bleeding in towards the centre.

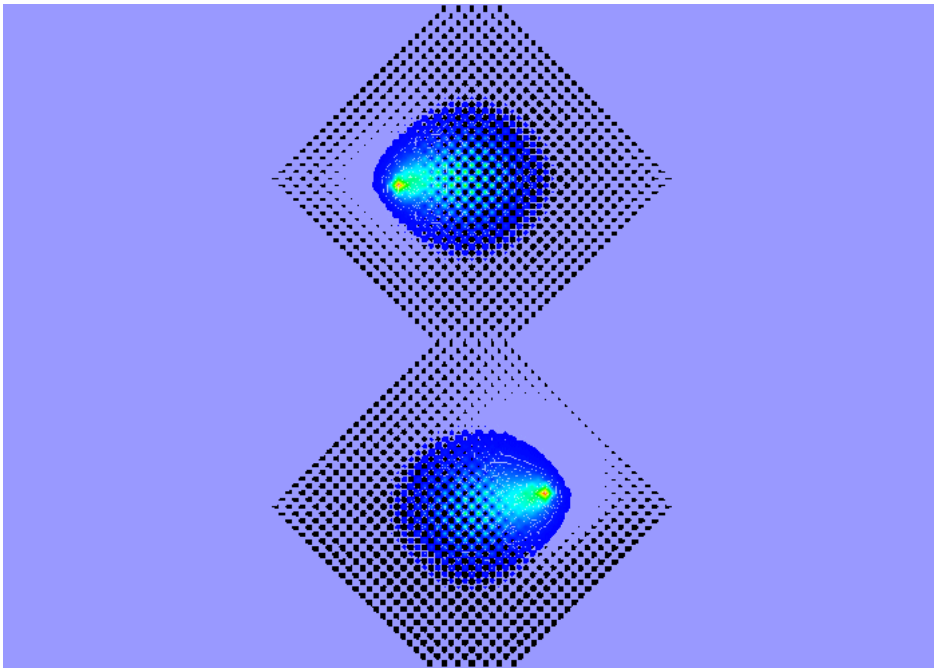


Fig 4.1.9: Temperature profile from an unstable system.

Figure 4.1.9 illustrates the propagation of an unstable error, 50 seconds in. For this simulation, $\lambda\Delta t=0.2688$, 7.5% above the stability condition. Each timestep, the coverage of the error expands by one node. This is a key part of the proof in appendix A.10.

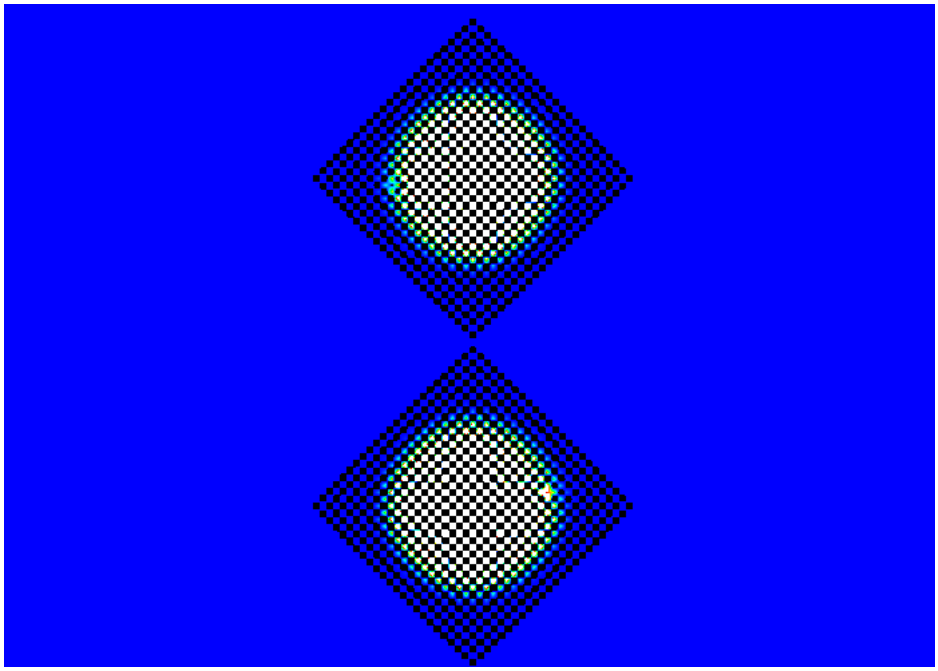


Fig 4.1.10: Temperature profile from a very unstable system.

Figure 4.1.10 is a more extreme version of figure 4.1.9, also 50 seconds in. This time $\lambda\Delta t=0.336$, 34.4% above the stability condition. Although this system is more unstable, the error has less coverage because a fewer number of timesteps have occurred.

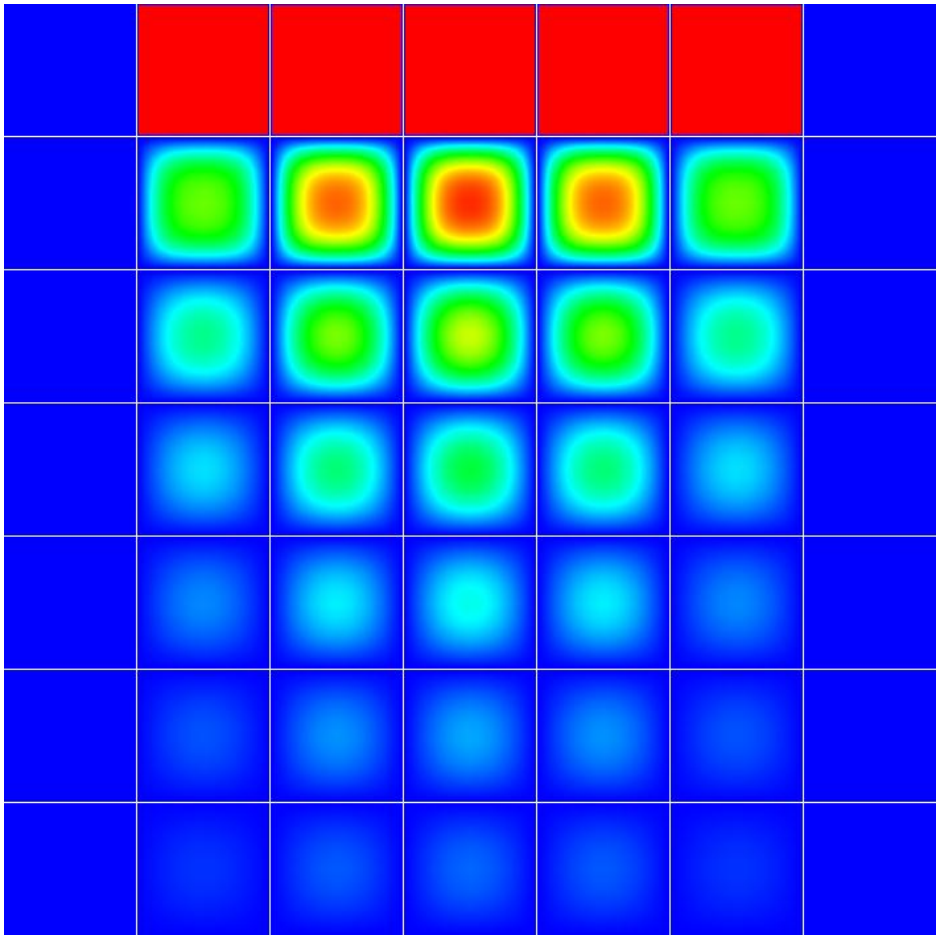


Fig 4.1.11: Time series of a 3D temperature profile.

Figure 4.1.11 is a 3D simulation. It represents a steel cube with side length 50cm, with interior temperature starting at 60°C and outer faces fixed at 0°C. The representation is similar to the technique often used to display the brain scans from magnetic resonance imaging (MRI). Each row is a series of cross-sectional slices of a temperature profile. The left and right slices are the outer faces of the cube and the slices in between are evenly distributed through the cube. The top row shows the initial temperature profile, the second at 1,000 seconds, third at 2,000 seconds and so on until the last row at 8,000 seconds. This illustrates the way the centre of an object is the slowest to change its temperature from an external influence, while those points closer to the exterior change more rapidly.

4.2 | Stability

Figure 4.2.1 illustrates a temperature profile from an unstable simulation:

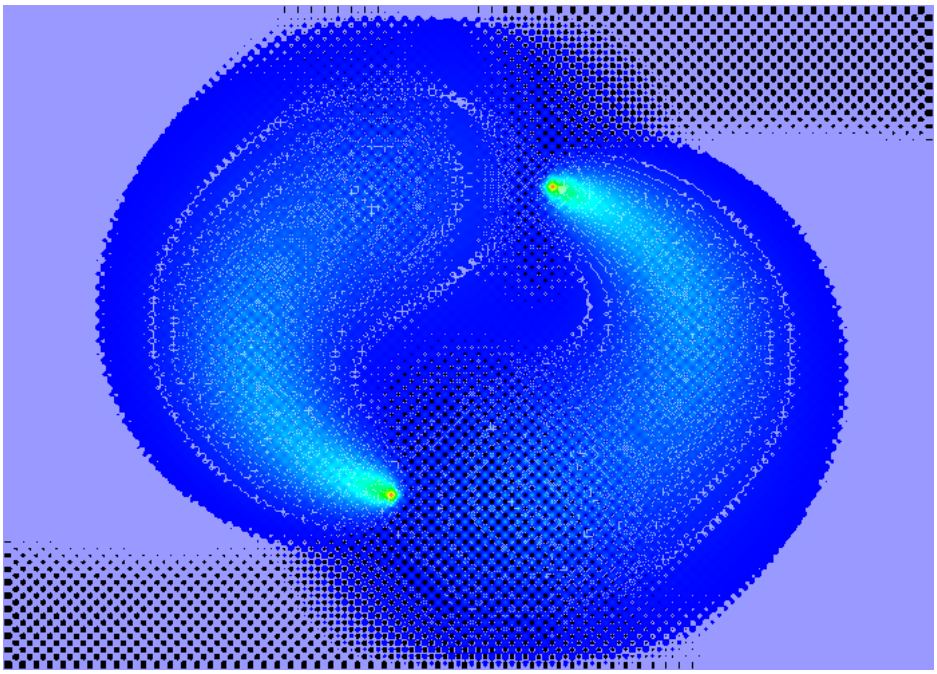


Fig 4.2.1: Temperature profile from a slightly unstable system.

The nature of the instability with these methods is that the difference between each temperature and its neighbours oscillates more and more violently. When the parameters are less unfavourable, but still not ideal, these oscillations are still present but they decay with each timestep.

To analyse the stability, an error of a specific form is introduced. Because the error is in a specific form, only a subspace of the parameters is proven to be unstable. The complement of this subspace, as far as the proof is concerned, could contain some parameters which are also unstable, but I conjecture that this is not the case; that the complement space contains only stable parameters.

Recalling equation 2.3.10, each relevant scheme can be represented by a matrix equation:

$$T^{(n+1)} = PT^{(n)} \qquad \text{- Eq 2.3.10}$$

Thanks to matrix multiplication being distributive, if an error is introduced, its evolution is independent and identical to the evolution of the system:

$$T^{(n+1)} + \varepsilon^{(n+1)} = P\left(T^{(n)} + \varepsilon^{(n)}\right) = PT^{(n)} + P\varepsilon^{(n)}$$

$$\varepsilon^{(n+1)} = P\varepsilon^{(n)}$$

Now consider the grid of error values (two-dimensional model here) classified into A values and B values as defined by relations 4.2.1 and 4.2.2:

$$\varepsilon_{i,j} \epsilon A \Leftrightarrow i + j = 0 \; mod \; 2 \qquad \text{- Rel 4.2.1}$$

$$\varepsilon_{i,j} \epsilon B \Leftrightarrow i + j = 1 \; mod \; 2 \qquad \text{- Rel 4.2.2}$$

This is illustrated in figure 4.2.2:

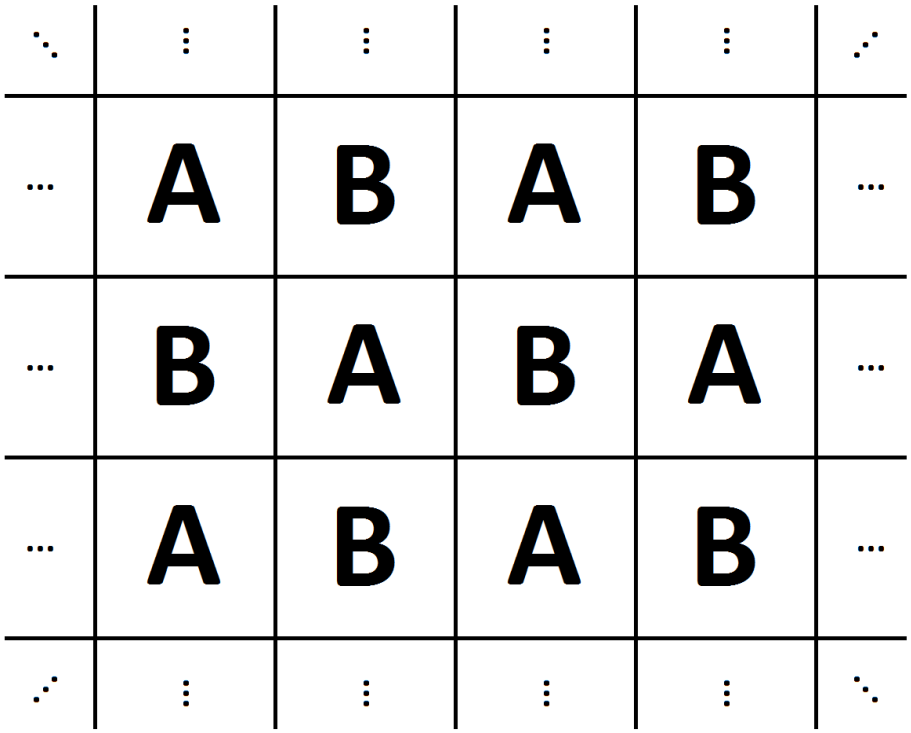


Fig 4.2.2: Arrangement of the A and B value classification.

As suggested by relations 4.2.1, 4.2.2, 'A' and 'B' in figure 4.2.2 are only labels. The A values are not assumed to be all equal, and they could be given subscripts to differentiate them, but there just aren't any relevant or helpful subscripts to use, similarly for the B values.

From here we take advantage of relative subscripts and use N, S, E, W as subscripts for the neighbour values, similar to the way they are introduced in section 2.3. On each timestep, each A value is replaced with a weighted average; x the weight of itself and y the weight of its neighbours, and similarly for each B value. This is described in equations 4.2.1, 4.2.2:

$$A_0^{(n+1)} = xA_0^{(n)} + y\left(B_N^{(n)} + B_S^{(n)} + B_E^{(n)} + B_W^{(n)}\right) \quad \text{- Eq 4.2.1}$$

$$B_0^{(n+1)} = xB_0^{(n)} + y\left(A_N^{(n)} + A_S^{(n)} + A_E^{(n)} + A_W^{(n)}\right) \quad \text{- Eq 4.2.2}$$

And here is the key to the solution – for each class of equations we do a *sum of each equation in that class*. This allows the $B_N^{(n)}$ term to become $\sum B^{(n)}$, because every B value is north of some A value, so each of $B_N^{(n)}$, $B_S^{(n)}$, $B_E^{(n)}$, $B_W^{(n)}$ becomes the same summation term.

$$\sum A^{(n+1)} = x\sum A^{(n)} + y\left(\sum B^{(n)} + \sum B^{(n)} + \sum B^{(n)} + \sum B^{(n)}\right)$$

$$\sum A^{(n+1)} = x\sum A^{(n)} + 4y\sum B^{(n)} \quad \text{- Eq 4.2.3}$$

$$\sum B^{(n+1)} = x\sum B^{(n)} + y\left(\sum A^{(n)} + \sum A^{(n)} + \sum A^{(n)} + \sum A^{(n)}\right)$$

$$\sum B^{(n+1)} = x\sum B^{(n)} + 4y\sum A^{(n)} \qquad \text{- Eq 4.2.4}$$

Now equations 4.2.3 and 4.2.4 are subtracted:

$$\sum A^{(n+1)} - \sum B^{(n+1)} = x\sum A^{(n)} + 4y\sum B^{(n)} - x\sum B^{(n)} - 4y\sum A^{(n)}$$

$$\sum A^{(n+1)} - \sum B^{(n+1)} = (x-4y)\left(\sum A^{(n)} - \sum B^{(n)}\right) \qquad \text{- Eq 4.2.5}$$

And since $x + 4y = 1$ (equation 2.3.9), $4y = 1 - x$. Substitute this into equation 4.2.5:

$$\sum A^{(n+1)} - \sum B^{(n+1)} = (2x-1)\left(\sum A^{(n)} - \sum B^{(n)}\right)$$

$$\sum A^{(n+h)} - \sum B^{(n+h)} = (2x-1)^h\left(\sum A^{(n)} - \sum B^{(n)}\right) \qquad \text{- Eq 4.2.6}$$

Clearly, we require relation 4.2.3 for the system to be stable:

$$|2x-1| \leq 1$$

$$0 \leq x \leq 1 \qquad \text{- Eq 4.2.3}$$

It is shown in appendix A.10 that if relation 4.2.3 is not met then if a single temperature is permuted, any bound will be exceeded by at least one temperature value after enough time steps.

By their nature, each applicable discretisation always satisfies $x \leq 1$, so it's really just the following relation that needs to be considered:

$$x \geq 0 \qquad \text{- Eq 4.2.4}$$

From this point, it is simple to derive stability conditions for each timestep. For Forward-Euler, recall equation 2.3.5:

$$T^{(n+1)} = (1 - 4\lambda\Delta t)T^{(n)} + \lambda\Delta t\sum N^{(n)}$$

$$x = 1 - 4\lambda\Delta t \geq 0$$

$$\Delta t \leq \frac{1}{4\lambda}$$

- Forward-Euler stability condition, relation 4.2.5

For Crank-Nicholson, recall equation 2.3.6:

$$T^{(n+1)} = \frac{1-2\lambda\Delta t}{1+2\lambda\Delta t}T^{(n)} + \frac{\lambda\Delta t}{1+2\lambda\Delta t}\sum N^{(n)}$$

$$x = \frac{1-2\lambda\Delta t}{1+2\lambda\Delta t} \geq 0$$

$$\Delta t \leq \frac{1}{2\lambda}$$

- Crank-Nicholson stability condition, relation 4.2.6

For Backward-Euler, recall equation 2.3.7:

$$T^{(n+1)} = \frac{1}{1+4\lambda\Delta t}T^{(n)} + \frac{\lambda\Delta t}{1+4\lambda\Delta t}\sum N^{(n)}$$

$$x = \frac{1}{1+4\lambda\Delta t} \geq 0$$

(no stability condition)

Figure 4.3.3 is a chart plotting the x values of each discretisation against λΔt. The stability condition of each method can be seen where they cross the λΔt axis:

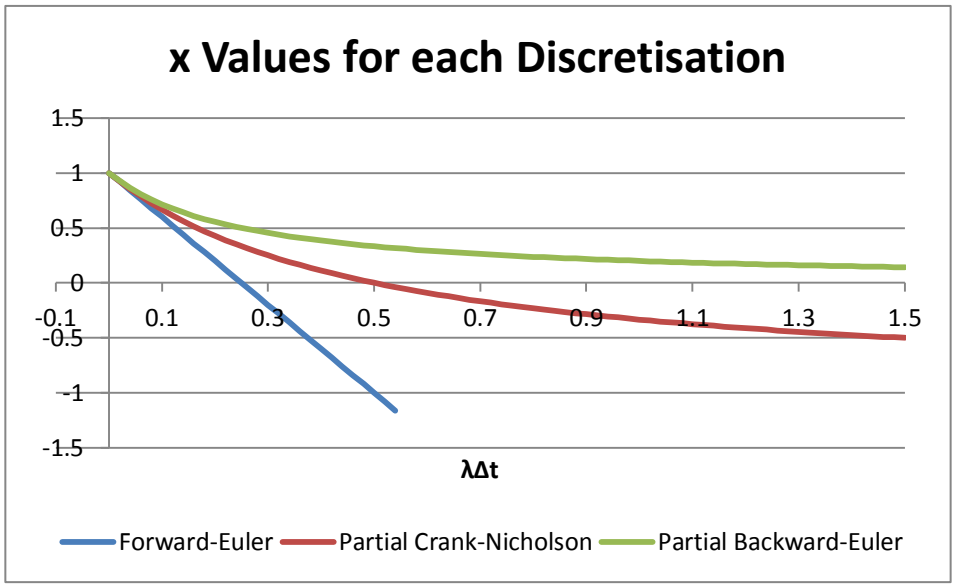


Fig 4.2.3: Plot of the x values from each discretisation.

4.3 | A Crank-Nicholson Example

Because the Partial Crank-Nicholson Method has been analysed elsewhere, an example and discussion of the real Crank-Nicholson Method is given here. As illustrated in figure 4.3.1, the example will be two-dimensional.

T	T	T	T	T
T	T_{11}	T_{12}	T_{13}	T
T	T_{21}	T_{22}	T_{23}	T
T	T_{31}	T_{32}	T_{33}	T
T	T	T	T	T

Fig 4.3.1: A small 2D system to demonstrate the Crank-Nicholson method.

This is actually the smallest possible 2D example which includes all of the features of a larger model. The only "normal" cell is T_{22}; every other cell is either a fixed boundary or adjacent to a fixed boundary. The outer cells lacking subscripts represent a fixed boundary condition of 0°C.

The Crank-Nicholson scheme is derived in appendix A.6. The solution is equation A.6.2:

$$T_0^{(+)} - \frac{\lambda\Delta t}{2(1+p\lambda\Delta t)}\sum_{i=1}^{p}\left(T_{\widehat{m}_i}^{(+)} + T_{-\widehat{m}_i}^{(+)}\right) = \frac{1-p\lambda\Delta t}{1+p\lambda\Delta t}T_0^{(0)} + \frac{\lambda\Delta t}{2(1+p\lambda\Delta t)}\sum_{i=1}^{p}\left(T_{\widehat{m}_i}^{(0)} + T_{-\widehat{m}_i}^{(0)}\right)$$

- Eq A.6.2

If the two-dimensional case is taken (p=2) and we use the neighbour/N notation illustrated in figure 2.3.2, we get:

$$T_0^{(+)} - \frac{\lambda\Delta t}{2(1+2\lambda\Delta t)}\sum_{i=1}^{2}\left(T_{\widehat{m}_i}^{(+)} + T_{-\widehat{m}_i}^{(+)}\right) = \frac{1-2\lambda\Delta t}{1+2\lambda\Delta t}T_0^{(0)} + \frac{\lambda\Delta t}{2(1+2\lambda\Delta t)}\sum_{i=1}^{2}\left(T_{\widehat{m}_i}^{(0)} + T_{-\widehat{m}_i}^{(0)}\right)$$

$$T^{(n+1)} - \frac{\lambda\Delta t}{2(1+2\lambda\Delta t)}\sum N^{(n+1)} = \frac{1-2\lambda\Delta t}{1+2\lambda\Delta t}T^{(n)} + \frac{\lambda\Delta t}{2(1+2\lambda\Delta t)}\sum N^{(n)}$$

- Eq 4.3.1

The material shall be stainless steel which makes $\kappa = 4.2 * 10^{-6} m^2 s^{-1}$ [2] (the thermal diffusivity of stainless steel). The timestep will be ten seconds ($\Delta t = 10$) and the spatial step will be 1.449cm ($\Delta x = 0.01449$). These values have been chosen to be relatable and make $\lambda\Delta t = \kappa\Delta t/\Delta x^2 = 0.2$, which is a little inside the stability condition for Forward-Euler, making it somewhat reasonable.

If we substitute these values into 4.3.1, we get:

$$T^{(n+1)} - \frac{0.2}{2.8}\sum N^{(n+1)} = \frac{0.6}{1.4}T^{(n)} + \frac{0.2}{2.8}\sum N^{(n)}$$

$$T^{(n+1)} - \frac{1}{14}\sum N^{(n+1)} = \frac{6}{14}T^{(n)} + \frac{1}{14}\sum N^{(n)}$$

$$14T^{(n+1)} - \sum N^{(n+1)} = 6T^{(n)} + \sum N^{(n)}$$

- Eq 4.3.2

Equation 4.3.2 is now used as a template to generate each element of the system of equations. For cells other than T_{22}, which touch a boundary, the boundary temperatures that appear in its equation are removed because they equal zero. For example, equation 4.3.3 applies the template to T_{31}:

$$14T_{31}^{(+)} - T_{41}^{(+)} - T_{21}^{(+)} - T_{32}^{(+)} - T_{30}^{(+)} = 6T_{31}^{(0)} + T_{41}^{(0)} + T_{21}^{(0)} + T_{32}^{(0)} + T_{30}^{(0)}$$

- Eq 4.3.3

The terms $T_{41}^{(+)}$, $T_{30}^{(0)}$, etc. are boundary temperatures equal to zero, so they are removed:

$$14T_{31}^{(+)} - T_{21}^{(+)} - T_{32}^{(+)} = 6T_{31}^{(0)} + T_{21}^{(0)} + T_{32}^{(0)}$$

The complete system is presented in equations 4.3.4 to 4.3.12:

$$14T_{11}^{(+)} - T_{12}^{(+)} - T_{21}^{(+)} = 6T_{11}^{(0)} + T_{12}^{(0)} + T_{21}^{(0)}$$

$$14T_{12}^{(+)} - T_{11}^{(+)} - T_{13}^{(+)} - T_{22}^{(+)} = 6T_{12}^{(0)} + T_{11}^{(0)} + T_{13}^{(0)} + T_{22}^{(0)}$$

$$14T_{13}^{(+)} - T_{12}^{(+)} - T_{23}^{(+)} = 6T_{13}^{(0)} + T_{12}^{(0)} + T_{23}^{(0)}$$

$$14T_{21}^{(+)} - T_{11}^{(+)} - T_{22}^{(+)} - T_{31}^{(+)} = 6T_{21}^{(0)} + T_{11}^{(0)} + T_{22}^{(0)} + T_{31}^{(0)}$$

$$14T_{22}^{(+)} - T_{12}^{(+)} - T_{21}^{(+)} - T_{23}^{(+)} - T_{32}^{(+)} = 6T_{22}^{(0)} + T_{12}^{(0)} + T_{21}^{(0)} + T_{23}^{(0)} + T_{32}^{(0)}$$

$$14T_{23}^{(+)} - T_{13}^{(+)} - T_{22}^{(+)} - T_{33}^{(+)} = 6T_{23}^{(0)} + T_{13}^{(0)} + T_{22}^{(0)} + T_{33}^{(0)}$$

$$14T_{31}^{(+)} - T_{21}^{(+)} - T_{32}^{(+)} = 6T_{31}^{(0)} + T_{21}^{(0)} + T_{32}^{(0)}$$

$$14T_{32}^{(+)} - T_{22}^{(+)} - T_{31}^{(+)} - T_{33}^{(+)} = 6T_{32}^{(0)} + T_{22}^{(0)} + T_{31}^{(0)} + T_{33}^{(0)}$$

$$14T_{33}^{(+)} - T_{23}^{(+)} - T_{32}^{(+)} = 6T_{33}^{(0)} + T_{23}^{(0)} + T_{32}^{(0)}$$

- Eq 4.3.4-12

Equation 4.3.13 combines equations 4.3.4 to 4.3.12 into a matrix equation:

$$\begin{pmatrix} 14 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 14 & -1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 14 & 0 & 0 & -1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 14 & -1 & 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & -1 & 14 & -1 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & -1 & 14 & 0 & 0 & -1 \\ 0 & 0 & 0 & -1 & 0 & 0 & 14 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & -1 & 14 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 14 \end{pmatrix} \begin{pmatrix} T_{11}^{(+)} \\ T_{12}^{(+)} \\ T_{13}^{(+)} \\ T_{21}^{(+)} \\ T_{22}^{(+)} \\ T_{23}^{(+)} \\ T_{31}^{(+)} \\ T_{32}^{(+)} \\ T_{33}^{(+)} \end{pmatrix} = \begin{pmatrix} 6 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 6 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 6 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 6 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 6 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 6 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 6 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 6 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 6 \end{pmatrix} \begin{pmatrix} T_{11}^{(0)} \\ T_{12}^{(0)} \\ T_{13}^{(0)} \\ T_{21}^{(0)} \\ T_{22}^{(0)} \\ T_{23}^{(0)} \\ T_{31}^{(0)} \\ T_{32}^{(0)} \\ T_{33}^{(0)} \end{pmatrix}$$

- Eq 4.3.13

The solution now is fairly straightforward; multiply both sides of the equation on the left by the inverse of the first matrix in equation 4.3.13. FreeMat (a free MATLAB clone) was used for the calculation:

$$\begin{pmatrix} T_{11}^{(+)} \\ T_{12}^{(+)} \\ T_{13}^{(+)} \\ T_{21}^{(+)} \\ T_{22}^{(+)} \\ T_{23}^{(+)} \\ T_{31}^{(+)} \\ T_{32}^{(+)} \\ T_{33}^{(+)} \end{pmatrix} = \begin{pmatrix} 0.4435 & 0.1047 & 0.0076 & 0.1047 & 0.0152 & 0.0016 & 0.0076 & 0.0016 & 0.0002 \\ 0.1047 & 0.4511 & 0.1047 & 0.0152 & 0.1064 & 0.0152 & 0.0016 & 0.0078 & 0.0016 \\ 0.0076 & 0.1047 & 0.4435 & 0.0016 & 0.0152 & 0.1047 & 0.0002 & 0.0016 & 0.0076 \\ 0.1047 & 0.0152 & 0.0016 & 0.4511 & 0.1064 & 0.0078 & 0.1047 & 0.0152 & 0.0016 \\ 0.0152 & 0.1064 & 0.0152 & 0.1064 & 0.4590 & 0.1064 & 0.0152 & 0.1064 & 0.0152 \\ 0.0016 & 0.0152 & 0.1047 & 0.0078 & 0.1064 & 0.4511 & 0.0016 & 0.0152 & 0.1047 \\ 0.0076 & 0.0016 & 0.0002 & 0.1047 & 0.0152 & 0.0016 & 0.4435 & 0.1047 & 0.0076 \\ 0.0016 & 0.0078 & 0.0016 & 0.0152 & 0.1064 & 0.0152 & 0.1047 & 0.4511 & 0.1047 \\ 0.0002 & 0.0016 & 0.0076 & 0.0016 & 0.0152 & 0.1047 & 0.0076 & 0.1047 & 0.4435 \end{pmatrix} \begin{pmatrix} T_{11}^{(0)} \\ T_{12}^{(0)} \\ T_{13}^{(0)} \\ T_{21}^{(0)} \\ T_{22}^{(0)} \\ T_{23}^{(0)} \\ T_{31}^{(0)} \\ T_{32}^{(0)} \\ T_{33}^{(0)} \end{pmatrix}$$

- Eq 4.3.14

This, however, is difficult to relate to, because the 2D array of temperatures has had to be forced into a one dimensional series. It's a good representation for computation, but it's much easier to see what's going on by interpreting the solution in the following way:

$$T_{22}^{(+)} = \sum \left[\begin{pmatrix} 0.0152 & 0.1064 & 0.0152 \\ 0.1064 & 0.4590 & 0.1064 \\ 0.0152 & 0.1064 & 0.0152 \end{pmatrix} .* \begin{pmatrix} T_{11}^{(0)} & T_{12}^{(0)} & T_{13}^{(0)} \\ T_{21}^{(0)} & T_{22}^{(0)} & T_{23}^{(0)} \\ T_{31}^{(0)} & T_{32}^{(0)} & T_{33}^{(0)} \end{pmatrix} \right]$$

$$T_{11}^{(+)} = \sum \left[\begin{pmatrix} 0.4435 & 0.1047 & 0.0076 \\ 0.1047 & 0.0152 & 0.0016 \\ 0.0076 & 0.0016 & 0.0002 \end{pmatrix} .* \begin{pmatrix} T_{11}^{(0)} & T_{12}^{(0)} & T_{13}^{(0)} \\ T_{21}^{(0)} & T_{22}^{(0)} & T_{23}^{(0)} \\ T_{31}^{(0)} & T_{32}^{(0)} & T_{33}^{(0)} \end{pmatrix} \right]$$

$$T_{12}^{(+)} = \sum \left[\begin{pmatrix} 0.1047 & 0.4511 & 0.1047 \\ 0.0152 & 0.1064 & 0.0152 \\ 0.0016 & 0.0078 & 0.0016 \end{pmatrix} .* \begin{pmatrix} T_{11}^{(0)} & T_{12}^{(0)} & T_{13}^{(0)} \\ T_{21}^{(0)} & T_{22}^{(0)} & T_{23}^{(0)} \\ T_{31}^{(0)} & T_{32}^{(0)} & T_{33}^{(0)} \end{pmatrix} \right]$$

- Eq 4.3.15-17

The '.*' operator has been borrowed from numerical packages such as MATLAB to represent element-wise multiplication, where the resulting matrix is found by multiplying each element only by the element in the same position in the other matrix. Only three cases are shown in equations 4.3.15 to 4.3.17 as the others can be found via rotation. This process, like the Forward-Euler and other simpler methods, replaces each value with a weighted average of itself and neighbouring values. As a visual guide, the elements in the right-hand matrix are in each case coloured an amount of green according to the weight for that cell (pale as the more distant elements are, this is after some scaling in the colour selection in an effort to make them more visible).

Something important to note is that a consequence of using this method is that the calculation for each cell involves every other cell in the grid. This means that the calculation for each cell requires $O(s^2)$ time, where s is the side length of the grid. This is a significant disadvantage when comparing to the Forward-Euler method, where the calculation for each cell is $O(1)$. It could be the case that zeros appear in the permutation matrices of larger systems, potentially reducing the computation time, but it appears that this is not the case (a 7x7 system was tested, and there were no zeros in there either). One technique could be to set a threshold for cell weight below which the calculation of that neighbour cell would be neglected. This could significantly reduce the computation time as it appears the cell weights rapidly go to zero with increasing distance, so depending on the threshold the vast majority of the entries of the permutation matrix would become zero. However, its validity would have to be justified.

4.4 | Pseudo-Continuous Approaches

Three additional methods were developed for comparison. Each of them employs a "pseudo-continuous" approach, where instead of discretising in time, assumptions are made about the cells further out than the immediate neighbours and a relatively small system of differential equations are solved instead.

Pseudo-Continuous Relaxed

This approach makes the assumption that the net heat transfer between the immediate neighbours and the more distant neighbours is zero within each timestep. This is described by equations 4.4.1 and 4.4.2:

$$\frac{dT}{dt} = -\lambda\left(4T - \sum N\right) \quad \text{- Eq 4.4.1}$$

$$\frac{dN_0}{dt} = -\lambda(N_0 - T) \quad \text{- Eq 4.4.2}$$

Note: equation 4.4.2 is yet another example of relative subscripting. The '0' subscript under 'N' can be replaced by any of the neighbour subscripts N, S, E, W.

Equations 4.4.1 and 4.4.2 are solved in appendix A.7 to produce equation 4.4.3:

$$T = \frac{1}{5}\left(T(0) + \sum N(0)\right) + \frac{1}{5}\left(4T(0) - \sum N(0)\right)e^{-5\lambda t} \quad \text{- Eq 4.4.3}$$

From here it is a simple matter of substituting $t = \Delta t$ and changing to the appropriate notation:

$$T^{(n+1)} = \frac{1}{5}\left(1 + 4e^{-5\lambda\Delta t}\right)T^{(n)} + \frac{1}{5}\left(1 - e^{-5\lambda\Delta t}\right)\sum N^{(n)}$$

$$x = \frac{1}{5}\left(1 + 4e^{-5\lambda\Delta t}\right) \quad \text{- Eq 4.4.4}$$

This scheme has some desirable properties; it is unconditionally free of both oscillations and instability! It may also seem superior because it takes some continuous heat transfer into account instead of being an outright discretisation. However, as will be explained in section 4.5, it appears Pseudo-Continuous Relaxed is not favourable despite these properties.

Pseudo-Continuous Fixed

This method fixes the neighbour temperatures as constant during each timestep instead of allowing heat transfer with the central temperature. The rationale for this is that each neighbour is more than likely adjacent to a cell with the same temperature difference as the difference with the central temperature; a situation similar to that illustrated in figure 2.2.1. Further, the other two cells each

neighbour is adjacent to may well resist any change in the neighbour cell's temperature. This is described in equation 4.4.5:

$$\underset{\textit{(N constant)}}{\frac{dT}{dt} = -\lambda\left(4T - \sum N\right)} \quad \text{- Eq 4.4.5}$$

This system is solved in appendix A.8. The result is equation A.8.3:

$$T = \frac{1}{4}\sum N(0) + \left(T(0) - \frac{1}{4}\sum N(0)\right)e^{-4\lambda t}$$

Similar to the Pseudo-Continuous Relaxed method above, a little rearranging reveals the numerical scheme:

$$T^{(n+1)} = e^{-4\lambda\Delta t}T^{(n)} + \frac{1}{4}\left(1 - e^{-4\lambda\Delta t}\right)\sum N^{(n)}$$

$$x = e^{-4\lambda\Delta t} \quad \text{- Eq 4.4.6}$$

Pseudo-Continuous Paraboloid

The Pseudo-Continuous Paraboloid approach results from some out-of-the-box thinking. Keeping things simple and reducing computation time by only using the central temperature and its neighbours on each timestep, what's the *best* possible approach?

All of the other methods are strictly discretised in space. What if the temperature values were interpolated continuously using a 2nd order Taylor series? Discretisation is used in large part to avoid the continuous analytical solution, but it turns out the result of the aforementioned interpolation is a paraboloid (or parabola in one dimension). A paraboloid is a special case of the heat equation which is easy to solve. Figure 4.4.1 illustrates a paraboloid.

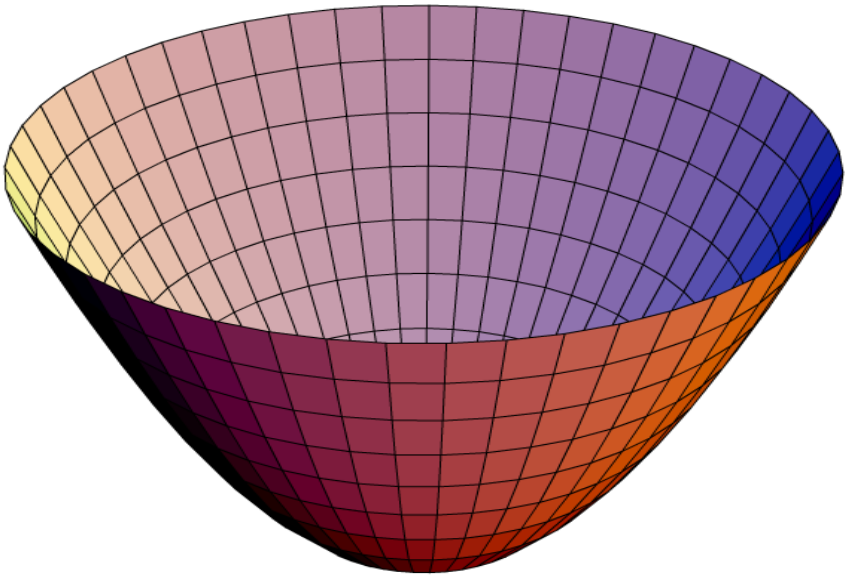


Fig 4.4.1: A paraboloid. (Public domain[4].)

The special thing about paraboloids making them easy to solve in the heat equation is that they have a constant concavity. At t=0, this means the rate of change of temperature is constant. Intuitively, one would suppose that the paraboloid will hold its shape and only be translated along the T axis through time. This guess can easily be verified in the one dimensional heat equation:

$$\frac{dT}{dt} = \kappa \frac{d^2T}{dx^2} \quad \text{- Eq 2.0.1}$$

$$T(x,t) = ax^2 + bx + c + vt \quad \text{- Proposed solution}$$

Substitute into the left side of equation 2.0.1:

$$\frac{d}{dt}(ax^2 + bx + c + vt) = v$$

Substitute into the right side of equation 2.0.1:

$$\kappa \frac{d^2}{dt^2}(ax^2 + bx + c + vt) = 2\kappa a$$

Simply let $v = 2\kappa a$ and the proposed solution satisfies the heat equation. This is done in more detail in appendix A.9.

It turns out that the Pseudo-Continuous Paraboloid method is equivalent to the Forward-Euler method, and this can be understood conceptually. When a paraboloid interpolation is applied the rate of change of T_0 is equal to the initial rate of change and is constant. This is what is assumed by the Forward-Euler method, so the result should be the same. This is proven in appendix A.9.

Hence the numerical scheme is the same as equation 2.3.5:

$$T^{(n+1)} = (1 - 4\lambda\Delta t)T^{(n)} + \lambda\Delta t \sum N^{(n)}$$

$$x = 1 - 4\lambda\Delta t \qquad \text{- Eq 4.4.7}$$

It is a surprising suggestion that the Forward-Euler method represents the best possible use of the information afforded to it. This will be supported by computing x values numerically in section 4.5.

x Value Comparison

Figure 4.4.2 is a chart comparing the x values of all the applicable schemes so far (non-partial Crank-Nicholson and Backward-Euler are exempt because their form does not allow them to be represented with only x and y values). Keep in mind that the Forward-Euler plot also represents the Pseudo-Continuous Paraboloid plot.

It can be seen that each method is close to agreement for small values of λΔt. As the timestep increases, the methods diverge. The choice between these methods is really about deciding how to cope with large timesteps.

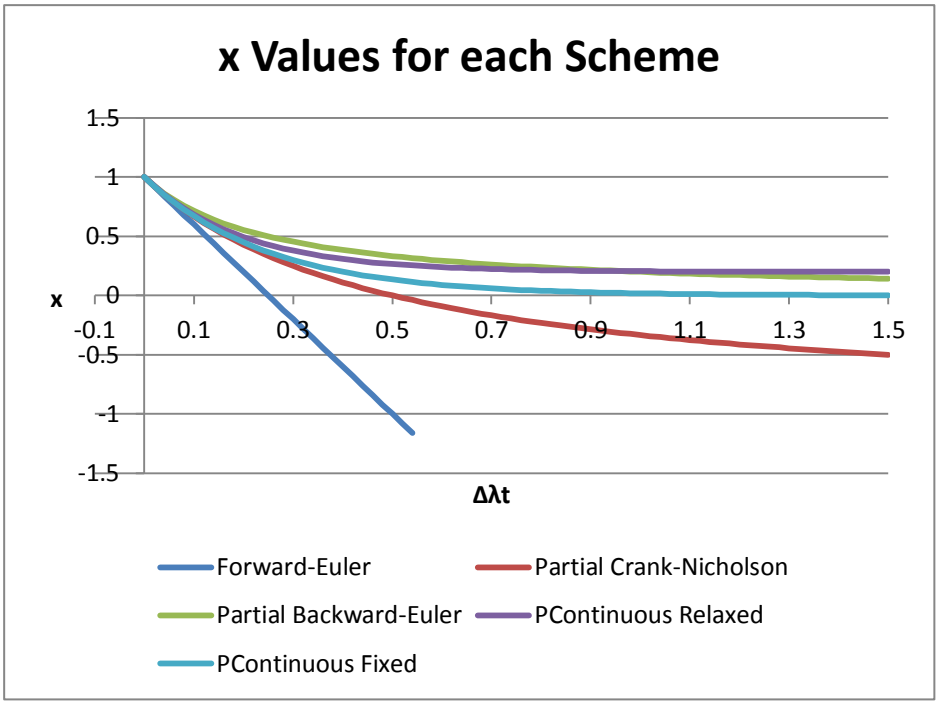


Figure 4.4.2: Plot of x values including those from the pseudo-continuous methods.

4.5 | Numerical x Values

After deriving this variety of schemes, all with different answers on how to choose the x value, it is natural to want to know which one is best. One approach is to test x values numerically. This is achieved by using a very small timestep to simulate the accurate solution, and calculating the x value required to produce the same result for a larger timestep. The calculation is as follows:

Recall equation 2.3.8:

$$T^{(n+1)} = xT^{(n)} + y\sum N^{(n)}$$

Instead of one timestep in the future, we change this to some arbitrary h number of small timesteps in the future:

$$T^{(n+h)} = xT^{(n)} + y\sum N^{(n)}$$

We also substitute $y = (1/4)(1-x)$ from equation 2.3.9:

$$T^{(n+h)} = xT^{(n)} + \frac{1}{4}(1-x)\sum N^{(n)} \qquad \text{- Eq 4.5.1}$$

Because the calculation for $T^{(n+h)}$ is known from the calculation with a small number of timesteps, we can rearrange equation 4.5.1 to determine the x value for the large timestep encompassing h small timesteps:

$$x = \frac{T^{(n+h)} - \frac{1}{4}\sum N^{(n)}}{T^{(n)} - \frac{1}{4}\sum N^{(n)}} \qquad \text{- Eq 4.5.2}$$

The x value calculated will be different for each temperature cell. This is because when dividing the large timestep into small timesteps information from more distant cells is taken into consideration. The numerical x value is taken to be the mean of the different values obtained.

Figure 4.5.1 is a chart including the numerical x values and the x values from the derived schemes. An initial temperature profile similar to figure 1.1 was used; the rotating hot points were discontinued, and the x values were measured as above.

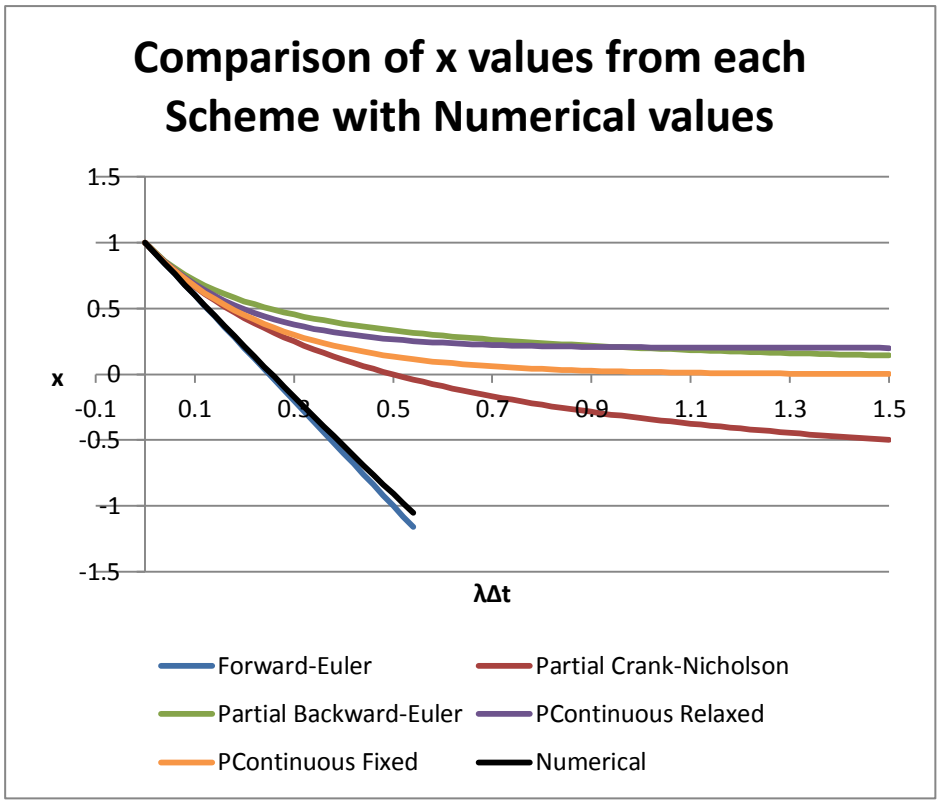


Fig 4.5.1: Plot of x values including numerically obtained values.

Figure 4.5.1 illustrates that according to this technique of numerically determining the x values, the Forward-Euler/Pseudo-Continuous Paraboloid method is a very clear front-runner. This suggests that the Pseudo-Continuous Paraboloid method has merit and that the Forward-Euler method, at least for heat diffusion, may deserve more credit than is widely given to it.

Chapter 5

Final Thoughts

The majority of the time put into this project has been focused on producing the images and video for visualising heat diffusion. Directly, this has mostly served only to confirm intuition. However, conceptual understanding has been emphasised in this report, and it has certainly helped to combine some meaningful example data with the plethora of equations presented. There is a motivating factor there because otherwise the whole point of the mathematical work, what it represents, can be difficult to relate to.

The inclusion of the “Partial Crank-Nicholson” and “Partial Backward-Euler” methods is due to an unfortunate oversight during the vast majority of this project. They were borne out of my incorrect derivation of the real versions of those methods. The criticisms levelled at these alternative versions of the well-known methods were originally levelled at the real versions of those methods. I realised the significance of this, and so spent some effort producing material about it.

I did not realise my mistake because these mistaken methods produced acceptable looking output and even appeared to have the right properties. Both are stable past the stability condition of Forward-Euler, Crank-Nicholson exhibits oscillations for large time steps, and Backward-Euler is unconditionally stable.

In hindsight, taking more advantage of consultation, where I could have detailed my initial derivation of these schemes, could have easily avoided this mistake. Further, the discovery was made when writing this report by thinking through my work in the necessary detail. As the old adage goes, “to teach is to learn”; report writing is important not only for the reader, but also for the writer. Basically, doing more of what I was supposed to do would have gone a long way.

In the end, this project is really about discretising partial differential equations (PDEs), and the choices for implementing them using finite difference methods. Exclusive focus has been given to the heat equation, but with more time, this work could be improved by visiting examples from other PDEs.

Bibliography

G. D. Smith, Numerical Solution of Partial Differential Equations: Finite Difference Methods, 1985, Oxford University Press, United States

Haberman, Applied Partial Differential Equations, 2004, Pearson Education, United States

http://en.wikipedia.org/wiki/Crank%E2%80%93Nicolson_method

http://en.wikipedia.org/wiki/Unit_vector

http://en.wikipedia.org/wiki/Circumflex

http://en.wikipedia.org/wiki/Thermal_diffusivity

http://en.wikipedia.org/wiki/Tridiagonal_matrix_algorithm

http://en.wikipedia.org/wiki/File:Paraboloid_of_Revolution.png

http://photofiltre.free.fr/

References

[1] G. D. Smith, Numerical Solution of Partial Differential Equations: Finite Difference Methods, p13, 1985, Oxford University Press, United States

[2] http://en.wikipedia.org/wiki/Thermal_diffusivity

[3] http://en.wikipedia.org/wiki/Tridiagonal_matrix_algorithm

[4] http://en.wikipedia.org/wiki/File:Paraboloid_of_Revolution.png, permission section

[5] http://photofiltre.free.fr/

Appendix

Most of this appendix details the full working for deriving equations used in this report. This detail is included to make the report as accessible as possible. The goal is to make the majority of it readily understandable at the undergraduate level and with the exclusion of Laplace transforms, it will hopefully also be largely understandable even by highly competent secondary school students.

A.1 | Relative Subscripting

Relative subscripting is a technique used to simplify the algebra in this project. It is used lightly in the main sections of this report, but it used very heavily in the derivations and proofs here in the appendix.

There are a few key ideas. First, relative subscripting is really a form of shorthand:

$$T_{m+1} = T_+$$

$$T_m = T_0$$

$$T_{m-1} = T_-$$

$$T_{m+2} = T_{++}$$

One of the techniques to make this work well is to use, where appropriate, modular arithmetic in the subscripts so that, for example, $X_+ = X_{[0]}$ when the central value is $X_{[3]}$ and the modulus is 4.

Also, vector arithmetic and unit vector notation are used when multiple subscripts (or an *unspecified arbitrary number* of subscripts) are required. In unit vector notation, a circumflex on an axis variable (or other vector) denotes a unit vector in the direction of that axis. For example, $\hat{u}$ denotes a unit vector in the direction of the u axis.

One example used involves axes for variables whose domain is the integers. In an arbitrary number of dimensions p, and temperature discretised in space, the form of a general temperature is:

$$T_{m_1,m_2,\dots m_p} = T_0$$

$m_1, m_2, \dots m_p$ are variables whose domain is the integers, but they can still be interpreted as axes. The subscript can also be seen as a vector in the domain $\mathbb{Z}^p$. As such, $\widehat{m_i}$ is a unit vector in the direction of the m_i axis. Consider the following expression:

$$T_{m_1,m_2,\dots m_{i-1},m_i+1,m_{i+1},\dots m_{j-1},m_j-1,m_{j+1},\dots m_p}$$

Using relative subscripting, this has the alternative much simpler and practical representation:

$$T_{\widehat{m_i}-\widehat{m_j}}$$

A.2 | A Newtonian Cooling Solution

Start from equations 2.1.1 and 2.1.2:

$$\frac{dT_C}{dt} = -\lambda(T_C - T_H) \qquad \frac{dT_H}{dt} = -\lambda(T_H - T_C)$$

Use a subscripting scheme $T_C = T_{[0]}$, $T_C = T_{[1]}$ (subscripts are equivalence classes modulo 2). Now use relative subscripting, condensing it into one equation:

$$\frac{dT_0}{dt} = -\lambda(T_0 - T_+)$$

Now take the Laplace transform of each side:

$$\mathcal{L}\left(\frac{dT_0}{dt}\right) = \mathcal{L}\big(-\lambda(T_0 - T_+)\big)$$

$$s\hat{T}_0 - T_0(0) = -\lambda\big(\hat{T}_0 - \hat{T}_+\big)$$

$$(s+\lambda)\hat{T}_0 = T_0(0) + \lambda\hat{T}_+$$

$$\hat{T}_0 = \frac{1}{s+\lambda}T_0(0) + \frac{\lambda}{s+\lambda}\hat{T}_+ \qquad \text{- Eq A.2.1}$$

Increment the subscripts for the equation to produce its partner:

$$\hat{T}_+ = \frac{1}{s+\lambda}T_+(0) + \frac{\lambda}{s+\lambda}\hat{T}_0$$

And substitute into equation A.2.1:

$$\hat{T}_0 = \frac{1}{s+\lambda}T_0(0) + \frac{\lambda}{s+\lambda}\left(\frac{1}{s+\lambda}T_+(0) + \frac{\lambda}{s+\lambda}\hat{T}_0\right)$$

$$\hat{T}_0 = \frac{1}{s+\lambda}T_0(0) + \frac{\lambda}{(s+\lambda)^2}T_+(0) + \frac{\lambda^2}{(s+\lambda)^2}\hat{T}_0$$

$$\left(1 - \frac{\lambda^2}{(s+\lambda)^2}\right)\hat{T}_0 = \frac{1}{s+\lambda}T_0(0) + \frac{\lambda}{(s+\lambda)^2}T_+(0)$$

$$\left(\frac{(s+\lambda)^2}{(s+\lambda)^2} - \frac{\lambda^2}{(s+\lambda)^2}\right)\hat{T}_0 = \frac{1}{s+\lambda}T_0(0) + \frac{\lambda}{(s+\lambda)^2}T_+(0)$$

$$\frac{s^2 + 2s\lambda + \lambda^2 - \lambda^2}{(s+\lambda)^2}\hat{T}_0 = \frac{1}{s+\lambda}T_0(0) + \frac{\lambda}{(s+\lambda)^2}T_+(0)$$

$$\frac{s(s+2\lambda)}{(s+\lambda)^2}\hat{T}_0 = \frac{1}{s+\lambda}T_0(0) + \frac{\lambda}{(s+\lambda)^2}T_+(0)$$

$$\hat{T}_0 = \frac{s+\lambda}{s(s+2\lambda)}T_0(0) + \frac{\lambda}{s(s+2\lambda)}T_+(0)$$

$$\hat{T}_0 = \frac{s}{s(s+2\lambda)}T_0(0) + \frac{\lambda}{s(s+2\lambda)}T_0(0) + \frac{\lambda}{s(s+2\lambda)}T_+(0)$$

$$\hat{T}_0 = \frac{1}{s+2\lambda}T_0(0) + \frac{\lambda}{s(s+2\lambda)}\big(T_0(0) + T_+(0)\big) \qquad \text{- Eq A.2.2}$$

We now use the method of partial fractions to expand $\lambda/[s(s+2\lambda)]$:

$$\frac{\lambda}{s(s+2\lambda)} = \frac{A}{s} + \frac{B}{s+2\lambda}$$

$$\lambda = A(s+2\lambda) + Bs$$

$$\lambda = (A+B)s + 2A\lambda$$

$$A + B = 0 \qquad 2A = 1$$

$$A = \frac{1}{2} \qquad B = -\frac{1}{2}$$

$$\frac{\lambda}{s(s+2\lambda)} = \frac{1}{2}\left(\frac{1}{s} - \frac{1}{s+2\lambda}\right)$$

Substitute this back into equation A.2.2:

$$\hat{T}_0 = \frac{1}{s+2\lambda}T_0(0) + \frac{1}{2}\left(\frac{1}{s} - \frac{1}{s+2\lambda}\right)\big(T_0(0) + T_+(0)\big)$$

Apply the inverse Laplace transform:

$$\mathcal{L}^{-1}\big(\hat{T}_0\big) = \mathcal{L}^{-1}\left(\frac{1}{s+2\lambda}T_0(0) + \frac{1}{2}\left(\frac{1}{s} - \frac{1}{s+2\lambda}\right)\big(T_0(0) + T_+(0)\big)\right)$$

$$T_0 = e^{-2\lambda t}T_0(0) + \frac{1}{2}\big(1 - e^{-2\lambda t}\big)\big(T_0(0) + T_+(0)\big)$$

$$T_0 = \frac{1}{2}\big(T_0(0) + T_+(0)\big) + \frac{1}{2}\big(T_0(0) - T_+(0)\big)e^{-2\lambda t}$$

Revert to traditional subscripting, yielding the two solutions:

$$T_C = \frac{1}{2}\big(T_C(0) + T_H(0)\big) + \frac{1}{2}\big(T_C(0) - T_H(0)\big)e^{-2\lambda t}$$

$$T_H = \frac{1}{2}\big(T_H(0) + T_C(0)\big) + \frac{1}{2}\big(T_H(0) - T_C(0)\big)e^{-2\lambda t}$$

A.3 | Spatial Discretisation

This project deals with the heat equation in multiple dimensions. So instead of starting with equation 2.0.1 (the heat equation), the heat equation for a generalised number of dimensions is used instead:

$$\frac{dT}{dt} = \kappa \sum_{i=1}^{p} \frac{d^2T}{dx_i^2}$$

Note – p is the number of dimensions, n is used elsewhere for the timestep index, as well as a few other letters which would otherwise be more familiar.

Replace derivatives with difference operators on the right hand side:

$$\frac{dT}{dt} = \kappa \sum_{i=1}^{p} \frac{\Delta^2T}{\Delta x_i^2}$$

Now require that each spatial step will be of equal size Δx. To do otherwise just introduces unnecessary mess:

$$\frac{dT}{dt} = \kappa \sum_{i=1}^{p} \frac{\Delta^2T}{\Delta x^2}$$

$$\frac{dT}{dt} = \frac{\kappa}{\Delta x^2} \sum_{i=1}^{p} \Delta^2T \qquad \text{- Eq A.3.1}$$

The unit vector notation will be combined with relative subscripting from this point. Please see appendix A.1 for more information.

Continue from A.3.1:

$$\frac{dT}{dt} = \frac{\kappa}{\Delta x^2} \sum_{i=1}^{p} \Delta^2T$$

$$\frac{dT_0}{dt} = \frac{\kappa}{\Delta x^2} \sum_{i=1}^{p} \Delta\left(T_{\widehat{m}_i} - T_0\right)$$

$$\frac{dT_0}{dt} = \frac{\kappa}{\Delta x^2} \sum_{i=1}^{p} \left(\Delta T_{\widehat{m}_i} - \Delta T_0\right)$$

Now use a backwards difference:

$$\frac{dT_0}{dt} = \frac{\kappa}{\Delta x^2} \sum_{i=1}^{p} \left[\left(T_{\widehat{m}_i} - T_0\right) - \left(T_0 - T_{-\widehat{m}_i}\right)\right]$$

$$\frac{dT_0}{dt} = -\frac{\kappa}{\Delta x^2}\sum_{i=1}^{p}\left[\left(T_0 - T_{\widehat{m_i}}\right) + \left(T_0 - T_{-\widehat{m_i}}\right)\right]$$

And there it is; even in an arbitrary number of dimensions each cell is simply undergoing Newtonian cooling with each of its neighbours. The only thing left is a little tidying by declaring lambda:

$$\lambda = \frac{\kappa}{\Delta x^2} \quad \text{- Eq A.3.2}$$

$$\frac{dT_0}{dt} = -\lambda\sum_{i=1}^{p}\left[\left(T_0 - T_{\widehat{m_i}}\right) + \left(T_0 - T_{-\widehat{m_i}}\right)\right] \quad \text{- Eq A.3.3}$$

A.4 | Forward-Euler

Start from equation A.3.3

$$\frac{dT_0}{dt} = -\lambda \sum_{i=1}^{p} \left[\left(T_0 - T_{\widehat{m_i}} \right) + \left(T_0 - T_{-\widehat{m_i}} \right) \right]$$

Now replace the time derivative with difference operators:

$$\frac{\Delta T_0}{\Delta t} = -\lambda \sum_{i=1}^{p} \left[\left(T_0 - T_{\widehat{m_i}} \right) + \left(T_0 - T_{-\widehat{m_i}} \right) \right]$$

Expand, and introduce a (relative) superscript in parentheses to denote the time step, now that more than one time step is in the equation:

$$\frac{\mathrm{T}_0^{(+)} - T_0^{(0)}}{\Delta t} = -\lambda \sum_{i=1}^{p} \left[\left(T_0^{(0)} - T_{\widehat{m_i}}^{(0)} \right) + \left(T_0^{(0)} - T_{-\widehat{m_i}}^{(0)} \right) \right]$$

$$\mathrm{T}_0^{(+)} = T_0^{(0)} - \lambda \Delta t \sum_{i=1}^{p} \left[\left(T_0^{(0)} - T_{\widehat{m_i}}^{(0)} \right) + \left(T_0^{(0)} - T_{-\widehat{m_i}}^{(0)} \right) \right] \quad \text{- Eq A.4.1}$$

Now substitute p=1 for the one-dimensional Forward-Euler scheme:

$$\mathrm{T}_0^{(+)} = T_0^{(0)} - \lambda \Delta t \sum_{i=1}^{1} \left[\left(T_0^{(0)} - T_{\widehat{m_i}}^{(0)} \right) + \left(T_0^{(0)} - T_{-\widehat{m_i}}^{(0)} \right) \right]$$

$$\mathrm{T}_0^{(+)} = T_0^{(0)} - \lambda \Delta t \left[\left(T_0^{(0)} - T_{\widehat{m_1}}^{(0)} \right) + \left(T_0^{(0)} - T_{-\widehat{m_1}}^{(0)} \right) \right]$$

Change to traditional subscripting and use m instead of m_1 since there's only one spatial dimension:

$$\mathrm{T}_m^{(\mathrm{n}+1)} = T_m^{(n)} - \lambda \Delta t \left[\left(T_m^{(n)} - T_{m+1}^{(n)} \right) + \left(T_m^{(n)} - T_{m-1}^{(n)} \right) \right]$$

For the two-dimensional version, substitute p=2 into equation A.4.1:

$$\mathrm{T}_0^{(+)} = T_0^{(0)} - \lambda \Delta t \sum_{i=1}^{2} \left[\left(T_0^{(0)} - T_{\widehat{m_i}}^{(0)} \right) + \left(T_0^{(0)} - T_{-\widehat{m_i}}^{(0)} \right) \right]$$

$$\mathrm{T}_0^{(+)} = T_0^{(0)} - \lambda \Delta t \left[\left(T_0^{(0)} - T_{\widehat{m_1}}^{(0)} \right) + \left(T_0^{(0)} - T_{-\widehat{m_1}}^{(0)} \right) + \left(T_0^{(0)} - T_{\widehat{m_2}}^{(0)} \right) + \left(T_0^{(0)} - T_{-\widehat{m_2}}^{(0)} \right) \right]$$

Change to traditional superscripts and the neighbour/N notation:

$$T^{(n+1)} = T^{(n)} - \lambda \Delta t \left[\left(T^{(n)} - N_E^{(n)} \right) + \left(T^{(n)} - N_W^{(n)} \right) + \left(T^{(n)} - N_N^{(n)} \right) + \left(T^{(n)} - N_S^{(n)} \right) \right]$$

$$T^{(n+1)} = T^{(n)} - \lambda \Delta t \left(4T^{(n)} - \sum N^{(n)} \right)$$

A.5 | Backward-Euler

Like Forward-Euler, we start from equation A.3.3:

$$\frac{dT_0}{dt} = -\lambda \sum_{i=1}^{p} \left[\left(T_0 - T_{\widehat{m}_i} \right) + \left(T_0 - T_{-\widehat{m}_i} \right) \right]$$

And replace the time derivative with difference operators:

$$\frac{\Delta T_0}{\Delta t} = -\lambda \sum_{i=1}^{p} \left[\left(T_0 - T_{\widehat{m}_i} \right) + \left(T_0 - T_{-\widehat{m}_i} \right) \right]$$

But here comes the (formally) different step. When expanding ΔT_0, superscripts are introduced to denote the time step, and on the right side of the equation, it must be decided whether the T terms belong to the current time step or the next time step. With forward Euler, it is assumed to be the current time step, but this method uses the next time step, which makes it implicit, since the next time step is not yet known:

$$\frac{\mathrm{T}_0^{(+)} - \mathrm{T}_0^{(0)}}{\Delta t} = -\lambda \sum_{i=1}^{p} \left[\left(T_0^{(+)} - T_{\widehat{m}_i}^{(+)} \right) + \left(T_0^{(+)} - T_{-\widehat{m}_i}^{(+)} \right) \right] \quad \text{- Eq A.5.1}$$

$$\mathrm{T}_0^{(+)} = \mathrm{T}_0^{(0)} - \lambda \Delta t \sum_{i=1}^{p} \left[\left(T_0^{(+)} - T_{\widehat{m}_i}^{(+)} \right) + \left(T_0^{(+)} - T_{-\widehat{m}_i}^{(+)} \right) \right]$$

$$\mathrm{T}_0^{(+)} = \mathrm{T}_0^{(0)} - \lambda \Delta t \sum_{i=1}^{p} \left[\left(T_0^{(+)} - T_{\widehat{m}_i}^{(+)} \right) + \left(T_0^{(+)} - T_{-\widehat{m}_i}^{(+)} \right) \right]$$

$$\mathrm{T}_0^{(+)} = \mathrm{T}_0^{(0)} - 2p\lambda \Delta t T_0^{(+)} + \lambda \Delta t \sum_{i=1}^{p} \left(T_{\widehat{m}_i}^{(+)} + T_{-\widehat{m}_i}^{(+)} \right)$$

$$\mathrm{T}_0^{(+)} + 2p\lambda \Delta t T_0^{(+)} = \mathrm{T}_0^{(0)} + \lambda \Delta t \sum_{i=1}^{p} \left(T_{\widehat{m}_i}^{(+)} + T_{-\widehat{m}_i}^{(+)} \right)$$

$$(1 + 2p\lambda \Delta t)\mathrm{T}_0^{(+)} = \mathrm{T}_0^{(0)} + \lambda \Delta t \sum_{i=1}^{p} \left(T_{\widehat{m}_i}^{(+)} + T_{-\widehat{m}_i}^{(+)} \right)$$

$$\mathrm{T}_0^{(+)} = \frac{1}{1 + 2p\lambda \Delta t} \mathrm{T}_0^{(0)} + \frac{\lambda \Delta t}{1 + 2p\lambda \Delta t} \sum_{i=1}^{p} \left(T_{\widehat{m}_i}^{(+)} + T_{-\widehat{m}_i}^{(+)} \right) \quad \text{- Eq A.5.2}$$

$$\mathrm{T}_0^{(+)} - \frac{\lambda \Delta t}{1 + 2p\lambda \Delta t} \sum_{i=1}^{p} \left(T_{\widehat{m}_i}^{(+)} + T_{-\widehat{m}_i}^{(+)} \right) = \frac{1}{1 + 2p\lambda \Delta t} \mathrm{T}_0^{(0)} \quad \text{- Eq A.5.3}$$

The difficulty with this is that it leaves us with a system of equations instead of the final solution. In

one dimension, the system is tri-diagonal, and can be solved efficiently using the tri-diagonal matrix algorithm[3]. However, in higher dimensions, a far more costly matrix inversion must be performed. The same situation occurs for the Crank-Nicholson method, and so implementing it is similar to the example given in section 4.3.

An alternative is to apply Backward-Euler partially (the "Partial Backward-Euler" method), so that only T_0 is taken from the next time step, which avoids the need to solve a system of equations. So instead of equation A.5.1:

$$\frac{\mathrm{T}_0^{(+)} - \mathrm{T}_0^{(0)}}{\Delta t} = -\lambda \sum_{i=1}^{p} \left[\left(T_0^{(+)} - T_{\widehat{m}_i}^{(+)} \right) + \left(T_0^{(+)} - T_{-\widehat{m}_i}^{(+)} \right) \right]$$

We have instead:

$$\frac{\mathrm{T}_0^{(+)} - \mathrm{T}_0^{(0)}}{\Delta t} = -\lambda \sum_{i=1}^{p} \left[\left(T_0^{(+)} - T_{\widehat{m}_i}^{(0)} \right) + \left(T_0^{(+)} - T_{-\widehat{m}_i}^{(0)} \right) \right]$$

And via identical algebra we get the alternative to A.5.2:

$$\mathrm{T}_0^{(+)} = \frac{1}{1 + 2p\lambda\Delta t} \mathrm{T}_0^{(0)} + \frac{\lambda\Delta t}{1 + 2p\lambda\Delta t} \sum_{i=1}^{p} \left(T_{\widehat{m}_i}^{(0)} + T_{-\widehat{m}_i}^{(0)} \right)$$

In two dimensions, we get:

$$\mathrm{T}_0^{(+)} = \frac{1}{1 + 4\lambda\Delta t} \mathrm{T}_0^{(0)} + \frac{\lambda\Delta t}{1 + 4\lambda\Delta t} \sum_{i=1}^{2} \left(T_{\widehat{m}_i}^{(0)} + T_{-\widehat{m}_i}^{(0)} \right)$$

$$\mathrm{T}_0^{(+)} = \frac{1}{1 + 4\lambda\Delta t} \mathrm{T}_0^{(0)} + \frac{\lambda\Delta t}{1 + 4\lambda\Delta t} \left(T_{\widehat{m}_1}^{(0)} + T_{-\widehat{m}_1}^{(0)} + T_{\widehat{m}_2}^{(0)} + T_{-\widehat{m}_2}^{(0)} \right)$$

Switch to neighbour/N notation:

$$\mathrm{T}^{(\mathrm{n}+1)} = \frac{1}{1 + 4\lambda\Delta t} \mathrm{T}^{(\mathrm{n})} + \frac{\lambda\Delta t}{1 + 4\lambda\Delta t} \sum N^{(n)} \qquad \text{- Eq A.5.4}$$

A.6 | Crank-Nicholson

Start from equation A.3.3 and replace the time derivative with difference operators:

$$\frac{dT_0}{dt} = -\lambda \sum_{i=1}^{p} \left[\left(T_0 - T_{\widehat{m}_i} \right) + \left(T_0 - T_{-\widehat{m}_i} \right) \right]$$

$$\frac{\Delta T_0}{\Delta t} = -\lambda \sum_{i=1}^{p} \left[\left(T_0 - T_{\widehat{m}_i} \right) + \left(T_0 - T_{-\widehat{m}_i} \right) \right]$$

Like Backward-Euler, the key part of Crank-Nicholson is the choice of which timestep to use for the right hand side when ΔT_0 is expanded. Instead of taking the current time step or the next time step, Crank-Nicholson takes the average of the two:

$$\frac{T_0^{(+)} - T_0^{(0)}}{\Delta t} = -\frac{1}{2}\lambda \sum_{i=1}^{p} \left[\left(T_0^{(0)} - T_{\widehat{m}_i}^{(0)} \right) + \left(T_0^{(0)} - T_{-\widehat{m}_i}^{(0)} \right) + \left(T_0^{(+)} - T_{\widehat{m}_i}^{(+)} \right) + \left(T_0^{(+)} - T_{-\widehat{m}_i}^{(+)} \right) \right]$$

- Equation A.6.1

$$T_0^{(+)} = T_0^{(0)} - \frac{1}{2}\lambda \Delta t \sum_{i=1}^{p} \left[\left(T_0^{(0)} - T_{\widehat{m}_i}^{(0)} \right) + \left(T_0^{(0)} - T_{-\widehat{m}_i}^{(0)} \right) + \left(T_0^{(+)} - T_{\widehat{m}_i}^{(+)} \right) + \left(T_0^{(+)} - T_{-\widehat{m}_i}^{(+)} \right) \right]$$

$$T_0^{(+)} = T_0^{(0)} - p\lambda \Delta t T_0^{(0)} - p\lambda \Delta t T_0^{(+)} + \frac{1}{2}\lambda \Delta t \sum_{i=1}^{p} \left(T_{\widehat{m}_i}^{(0)} + T_{-\widehat{m}_i}^{(0)} + T_{\widehat{m}_i}^{(+)} + T_{-\widehat{m}_i}^{(+)} \right)$$

$$(1 + p\lambda \Delta t) T_0^{(+)} = (1 - p\lambda \Delta t) T_0^{(0)} + \frac{1}{2}\lambda \Delta t \sum_{i=1}^{p} \left(T_{\widehat{m}_i}^{(0)} + T_{-\widehat{m}_i}^{(0)} \right) + \frac{1}{2}\lambda \Delta t \sum_{i=1}^{p} \left(T_{\widehat{m}_i}^{(+)} + T_{-\widehat{m}_i}^{(+)} \right)$$

$$T_0^{(+)} = \frac{1 - p\lambda \Delta t}{1 + p\lambda \Delta t} T_0^{(0)} + \frac{\lambda \Delta t}{2(1 + p\lambda \Delta t)} \sum_{i=1}^{p} \left(T_{\widehat{m}_i}^{(0)} + T_{-\widehat{m}_i}^{(0)} \right) + \frac{\lambda \Delta t}{2(1 + p\lambda \Delta t)} \sum_{i=1}^{p} \left(T_{\widehat{m}_i}^{(+)} + T_{-\widehat{m}_i}^{(+)} \right)$$

$$T_0^{(+)} - \frac{\lambda \Delta t}{2(1 + p\lambda \Delta t)} \sum_{i=1}^{p} \left(T_{\widehat{m}_i}^{(+)} + T_{-\widehat{m}_i}^{(+)} \right) = \frac{1 - p\lambda \Delta t}{1 + p\lambda \Delta t} T_0^{(0)} + \frac{\lambda \Delta t}{2(1 + p\lambda \Delta t)} \sum_{i=1}^{p} \left(T_{\widehat{m}_i}^{(0)} + T_{-\widehat{m}_i}^{(0)} \right)$$

- Equation A.6.2

So, like the Backward-Euler method and equation A.5.3, we are left with a system of equations instead of the final solution. One way to get around this would be to use a “Partial Crank-Nicholson” method, where only T_0 has values from the next timestep taken into account, instead of both T_0 and its neighbours. Using this strategy, instead of equation A.6.1:

$$\frac{T_0^{(+)} - T_0^{(0)}}{\Delta t} = -\frac{1}{2}\lambda \sum_{i=1}^{p} \left[\left(T_0^{(0)} - T_{\widehat{m}_i}^{(0)} \right) + \left(T_0^{(0)} - T_{-\widehat{m}_i}^{(0)} \right) + \left(T_0^{(+)} - T_{\widehat{m}_i}^{(+)} \right) + \left(T_0^{(+)} - T_{-\widehat{m}_i}^{(+)} \right) \right]$$

We get:

$$\frac{T_0^{(+)} - T_0^{(0)}}{\Delta t} = -\lambda \sum_{i=1}^{p} \left[\left(\frac{1}{2}\left(T_0^{(0)} + T_0^{(+)} \right) - T_{\widehat{m}_i}^{(0)} \right) + \left(\frac{1}{2}\left(T_0^{(0)} + T_0^{(+)} \right) - T_{-\widehat{m}_i}^{(0)} \right) \right]$$

$$T_0^{(+)} = T_0^{(0)} - \lambda \Delta t \sum_{i=1}^{p} \left[\left(\frac{1}{2}\left(T_0^{(0)} + T_0^{(+)} \right) - T_{\widehat{m}_i}^{(0)} \right) + \left(\frac{1}{2}\left(T_0^{(0)} + T_0^{(+)} \right) - T_{-\widehat{m}_i}^{(0)} \right) \right]$$

$$T_0^{(+)} = T_0^{(0)} - p\lambda \Delta t T_0^{(0)} - p\lambda \Delta t T_0^{(+)} + \lambda \Delta t \sum_{i=1}^{p} \left(T_{\widehat{m}_i}^{(0)} + T_{-\widehat{m}_i}^{(0)} \right)$$

$$(1 + p\lambda \Delta t) T_0^{(+)} = (1 - p\lambda \Delta t) T_0^{(0)} + \lambda \Delta t \sum_{i=1}^{p} \left(T_{\widehat{m}_i}^{(0)} + T_{-\widehat{m}_i}^{(0)} \right)$$

$$T_0^{(+)} = \frac{1 - p\lambda \Delta t}{1 + p\lambda \Delta t} T_0^{(0)} + \frac{\lambda \Delta t}{1 + p\lambda \Delta t} \sum_{i=1}^{p} \left(T_{\widehat{m}_i}^{(0)} + T_{-\widehat{m}_i}^{(0)} \right)$$

Switching to the neighbour/N notation and substituting p=2, this becomes:

$$T^{(n+1)} = \frac{1 - 2\lambda \Delta t}{1 + 2\lambda \Delta t} T^{(n)} + \frac{\lambda \Delta t}{1 + 2\lambda \Delta t} \sum N^{(n)} \qquad \text{- Eq A.6.3}$$

A.7 | Pseudo-Continuous Relaxed

Start with equation A.3.3:

$$\frac{dT_0}{dt} = -\frac{\kappa}{\Delta x^2}\sum_{i=1}^{p}\left[\left(T_0 - T_{\widehat{m_i}}\right) + \left(T_0 - T_{-\widehat{m_i}}\right)\right]$$

Then change to the neighbour/N notation. In the summation, T_0 appears twice for every dimension, i.e. 2p times, and every neighbour is subtracted once. So in the new notation the same equation is:

$$\frac{dT}{dt} = -\frac{\kappa}{\Delta x^2}\left(2pT - \sum N\right)$$

Also, kappa/delta-x squared is replaced by lambda:

$$\frac{dT}{dt} = -\lambda\left(2pT - \sum N\right) \quad \text{- Eq A.7.1}$$

It is possible to solve this entire system, but that would be costly; it could be *thousands* of equations. A simple assumption is made instead – that no net outside heat transfer occurs for each neighbour cell. This means the equation for each neighbour is:

$$\frac{dN_0}{dt} = -\lambda(N_0 - T) \quad \text{- Eq A.7.2}$$

Now each T can be solved as a system of 2p + 1 differential equations; it is quickly reduced to 2 differential equations, and is actually very similar to A.2. But first Laplace transforms are applied to A.7.1:

$$\mathcal{L}\left(\frac{dT}{dt}\right) = \mathcal{L}\left(-\lambda\left(2pT - \sum N\right)\right)$$

$$s\hat{T} - T(0) = -\lambda\left(2p\hat{T} - \sum \hat{N}\right)$$

$$(s + 2p\lambda)\hat{T} = T(0) + \lambda\sum \hat{N}$$

$$\hat{T} = \frac{1}{s + 2p\lambda}T(0) + \frac{\lambda}{s + 2p\lambda}\sum \hat{N} \quad \text{- Eq A.7.3}$$

Now apply Laplace transforms to A.7.2:

$$\mathcal{L}\left(\frac{dN_0}{dt}\right) = \mathcal{L}\left(-\lambda(N_0 - T)\right)$$

$$s\hat{N}_0 - N_0(0) = -\lambda\left(\hat{N}_0 - \hat{T}\right)$$

$$(s + \lambda)\hat{N}_0 = N_0(0) + \lambda\hat{T}$$

$$\hat{N}_0 = \frac{1}{s + \lambda}N_0(0) + \frac{\lambda}{s + \lambda}\hat{T} \quad \text{- Eq A.7.4}$$

Now a sum is taken across all equations in A.7.4:

$$\sum \hat{N} = \frac{1}{s+\lambda}\sum N(0) + \frac{2p\lambda}{s+\lambda}\hat{T}$$

Substitute into A.7.3:

$$\hat{T} = \frac{1}{s+2p\lambda}T(0) + \frac{\lambda}{s+2p\lambda}\left(\frac{1}{s+\lambda}\sum N(0) + \frac{2p\lambda}{s+\lambda}\hat{T}\right)$$

$$\hat{T} = \frac{1}{s+2p\lambda}T(0) + \frac{\lambda}{(s+\lambda)(s+2p\lambda)}\sum N(0) + \frac{2p\lambda^2}{(s+\lambda)(s+2p\lambda)}\hat{T}$$

$$\left(1 - \frac{2p\lambda^2}{(s+\lambda)(s+2p\lambda)}\right)\hat{T} = \frac{1}{s+2p\lambda}T(0) + \frac{\lambda}{(s+\lambda)(s+2p\lambda)}\sum N(0)$$

$$\frac{(s+\lambda)(s+2p\lambda) - 2p\lambda^2}{(s+\lambda)(s+2p\lambda)}\hat{T} = \frac{1}{s+2p\lambda}T(0) + \frac{\lambda}{(s+\lambda)(s+2p\lambda)}\sum N(0)$$

$$\frac{s^2 + (2p+1)s\lambda + 2p\lambda^2 - 2p\lambda^2}{(s+\lambda)(s+2p\lambda)}\hat{T} = \frac{1}{s+2p\lambda}T(0) + \frac{\lambda}{(s+\lambda)(s+2p\lambda)}\sum N(0)$$

$$\frac{s(s+(2p+1)\lambda)}{(s+\lambda)(s+2p\lambda)}\hat{T} = \frac{1}{s+2p\lambda}T(0) + \frac{\lambda}{(s+\lambda)(s+2p\lambda)}\sum N(0)$$

$$s(s+(2p+1)\lambda)\hat{T} = (s+\lambda)T(0) + \lambda\sum N(0)$$

$$\hat{T} = \frac{s+\lambda}{s(s+(2p+1)\lambda)}T(0) + \frac{\lambda}{s(s+(2p+1)\lambda)}\sum N(0)$$

$$\hat{T} = \frac{s}{s(s+(2p+1)\lambda)}T(0) + \frac{\lambda}{s(s+(2p+1)\lambda)}\left(T(0) + \sum N(0)\right)$$

$$\hat{T} = \frac{1}{s+(2p+1)\lambda}T(0) + \frac{\lambda}{s(s+(2p+1)\lambda)}\left(T(0) + \sum N(0)\right)$$

- Equation A.7.5

Now use a method of partial fractions separation:

$$\frac{\lambda}{s(s+(2p+1)\lambda)} = \frac{A}{s} + \frac{B}{s+(2p+1)\lambda}$$ - Eq A.7.6

$$\lambda = A[s+(2p+1)\lambda] + Bs$$

$$\lambda = (A+B)s + (2p+1)\lambda A$$

$$A + B = 0 \qquad (2p+1)\lambda A = \lambda$$

$$A = \frac{1}{2p+1} \qquad B = -\frac{1}{2p+1}$$

Substitute into equation A.7.6:

$$\frac{\lambda}{s(s+(2p+1)\lambda)} = \frac{\left(\frac{1}{2p+1}\right)}{s} + \frac{\left(-\frac{1}{2p+1}\right)}{s+(2p+1)\lambda}$$

$$\frac{\lambda}{s(s+(2p+1)\lambda)} = \frac{1}{2p+1}\left(\frac{1}{s} - \frac{1}{s+(2p+1)\lambda}\right)$$

Substitute into equation A.7.5:

$$\hat{T} = \frac{1}{s+(2p+1)\lambda}T(0) + \frac{1}{2p+1}\left(\frac{1}{s} - \frac{1}{s+(2p+1)\lambda}\right)\left(T(0) + \sum N(0)\right)$$

Take the inverse Laplace transform:

$$\mathcal{L}^{-1}\left(\hat{T}\right) = \mathcal{L}^{-1}\left(\frac{1}{s+(2p+1)\lambda}T(0) + \frac{1}{2p+1}\left(\frac{1}{s} - \frac{1}{s+(2p+1)\lambda}\right)\left(T(0) + \sum N(0)\right)\right)$$

$$T = e^{-(2p+1)\lambda t}T(0) + \frac{1}{2p+1}\left(1 - e^{-(2p+1)\lambda t}\right)\left(T(0) + \sum N(0)\right)$$

Rearrange:

$$T = \frac{1}{2p+1}\left(T(0) + \sum N(0)\right) + \frac{1}{2p+1}\left(2pT(0) - \sum N(0)\right)e^{-(2p+1)\lambda t}$$

For two dimensions, i.e. p=2:

$$T = \frac{1}{5}\left(T(0) + \sum N(0)\right) + \frac{1}{5}\left(4T(0) - \sum N(0)\right)e^{-5\lambda t} \qquad \text{- Eq A.7.7}$$

A.8 | Pseudo-Continuous Fixed

For this scheme, each central temperature is governed by A.7.1, which is really A.3.3 in a different notation:

$$\frac{dT}{dt} = -\lambda\left(2pT - \sum N\right)$$

But the neighbours are assumed to be constant during each time step, so $\sum N = \sum N\,(0)$, which is of course also constant. We apply a Laplace transform to each side:

$$\mathcal{L}\left(\frac{dT}{dt}\right) = \mathcal{L}\left(-\lambda\left(2pT - \sum N(0)\right)\right)$$

$$s\hat{T} - T(0) = -\lambda\left(2p\hat{T} - \frac{1}{s}\sum N(0)\right)$$

$$(s + 2p\lambda)\hat{T} = T(0) + \frac{\lambda}{s}\sum N(0)$$

$$\hat{T} = \frac{1}{s + 2p\lambda}T(0) + \frac{\lambda}{s(s + 2p\lambda)}\sum N(0) \qquad \text{- Eq A.8.1}$$

Now use a method of partial fractions separation:

$$\frac{\lambda}{s(s + 2p\lambda)} = \frac{A}{s} + \frac{B}{s + 2p\lambda} \qquad \text{- Eq A.8.2}$$

$$\lambda = A(s + 2p\lambda) + Bs$$

$$\lambda = (A + B)s + 2p\lambda A$$

$$A + B = 0 \qquad\qquad 2p\lambda A = \lambda$$

$$A = \frac{1}{2p} \qquad\qquad B = -\frac{1}{2p}$$

Substitute into equation A.8.2:

$$\frac{\lambda}{s(s + 2p\lambda)} = \frac{\left(\frac{1}{2p}\right)}{s} + \frac{\left(-\frac{1}{2p}\right)}{s + 2p\lambda}$$

$$\frac{\lambda}{s(s + 2p\lambda)} = \frac{1}{2p}\left(\frac{1}{s} - \frac{1}{s + 2p\lambda}\right)$$

Substitute into A.8.1:

$$\hat{T} = \frac{1}{s + 2p\lambda}T(0) + \frac{1}{2p}\left(\frac{1}{s} - \frac{1}{s + 2p\lambda}\right)\sum N(0)$$

Take the inverse Laplace transform of both sides:

$$\mathcal{L}^{-1}(\hat{T}) = \mathcal{L}^{-1}\left(\frac{1}{s+2p\lambda}T(0) + \frac{1}{2p}\left(\frac{1}{s} - \frac{1}{s+2p\lambda}\right)\sum N(0)\right)$$

$$T = e^{-2p\lambda t}T(0) + \frac{1}{2p}\left(1 - e^{-2p\lambda t}\right)\sum N(0)$$

$$T = \frac{1}{2p}\sum N(0) + \left(T(0) - \frac{1}{2p}\sum N(0)\right)e^{-2p\lambda t}$$

In two dimensions, i.e. p=2:

$$T = \frac{1}{4}\sum N(0) + \left(T(0) - \frac{1}{4}\sum N(0)\right)e^{-4\lambda t} \qquad \text{- Eq A.8.3}$$

A.9 | Pseudo-Continuous Paraboloid

We start by fitting the constant moving paraboloid. The coordinates are relative to the central temperature (note that x is a vector):

$$T(x,t) = \sum_{i=1}^{p} \left(a_i x_i^2 + b_i x_i\right) + c + vt \qquad \text{- Eq A.9.1}$$

Now at t=0, the paraboloid should fit the central initial temperature:

$$T(0,0) = \sum_{i=1}^{p} (a_i * 0^2 + b_i * 0) + c + v * 0 = T_0(0)$$

$$c = T_0(0)$$

Substitute into equation A.9.1:

$$T(x,t) = \sum_{i=1}^{p} \left(a_i x_i^2 + b_i x_i\right) + T_0(0) + vt \qquad \text{- Eq A.9.2}$$

Now the moving paraboloid must satisfy the heat equation:

$$\frac{dT}{dt} = \kappa \sum_{i=1}^{p} \frac{d^2T}{dx_i^2}$$

Substitute equation A.9.1 into here:

$$v = \kappa \sum_{i=1}^{p} 2a_i$$

$$v = 2\kappa \sum_{i=1}^{p} a_i$$

Substitute into equation A.9.2:

$$T(x,t) = \sum_{i=1}^{p} \left(a_i x_i^2 + b_i x_i\right) + T_0(0) + \left(2\kappa \sum_{i=1}^{p} a_i\right) t \qquad \text{- Eq A.9.3}$$

The only thing we're really interested in from this paraboloid is the central temperature at t=Δt, so we need to find:

$$T_0(\Delta t) = T(0,\Delta t) = \sum_{i=1}^{p} (a_i * 0 * 0 + b_i * 0) + T_0(0) + \left(2\kappa \sum_{i=1}^{p} a_i\right) \Delta t$$

$$T_0(\Delta t) = T_0(0) + \left(2\kappa \sum_{i=1}^{p} a_i\right)\Delta t \qquad \text{- Eq A.9.4}$$

Hence all we really need to know is the a_i values. Continuing from A.9.3, we require that the paraboloid match all neighbours one cell over in each positive direction:

$$T\left(\Delta x * \hat{x_j}, 0\right) = \sum_{i=1}^{p}\left(a_i x_i^2 + b_i x_i\right) + T_0(0) + \left(2\kappa \sum_{i=1}^{p} a_i\right) * 0 = T_{\widehat{m_j}}(0)$$

$x_i = 0$ except for $i = j$ and $x_j = \Delta x$ so:

$$T\left(\Delta x * \hat{x_j}, 0\right) = \sum_{i \neq j}^{p}(a_i 0 * 0 + b_i * 0) + a_j \Delta x^2 + b_j \Delta x + T_0(0) = T_{\widehat{m_j}}(0)$$

$$a_j \Delta x^2 + b_j \Delta x + T_0(0) = T_{\widehat{m_j}}(0)$$

$$a_j \Delta x^2 + b_j \Delta x = T_{\widehat{m_j}}(0) - T_0(0) \qquad \text{- Eq A.9.5}$$

Same goes for all neighbours one cell over in each negative direction:

$$T\left(-\Delta x * \hat{x_j}, 0\right) = \sum_{i=1}^{p}\left(a_i x_i^2 + b_i x_i\right) + T_0(0) + \left(2\kappa \sum_{i=1}^{p} a_i\right) * 0 = T_{-\widehat{m_j}}(0)$$

$x_i = 0$ except for $i = j$ and $x_j = -\Delta x$ so:

$$T\left(-\Delta x * \hat{x_j}, 0\right) = \sum_{i \neq j}^{p}(a_i 0 * 0 + b_i * 0) + a_j(-\Delta x)^2 + b_j(-\Delta x) + T_0(0) = T_{-\widehat{m_j}}(0)$$

$$a_j \Delta x^2 - b_j \Delta x + T_0(0) = T_{-\widehat{m_j}}(0)$$

$$a_j \Delta x^2 - b_j \Delta x = T_{-\widehat{m_j}}(0) - T_0(0) \qquad \text{- Eq A.9.6}$$

Now take the sum of equations A.9.5 and A.9.6:

$$a_j \Delta x^2 + b_j \Delta x + a_j \Delta x^2 - b_j \Delta x = T_{\widehat{m_j}}(0) - T_0(0) + T_{-\widehat{m_j}}(0) - T_0(0)$$

$$2a_j \Delta x^2 = \left(T_{\widehat{m_j}}(0) - T_0(0)\right) + \left(T_{-\widehat{m_j}}(0) - T_0(0)\right)$$

$$a_j = \frac{1}{2\Delta x^2}\left[\left(T_{\widehat{m_j}}(0) - T_0(0)\right) + \left(T_{-\widehat{m_j}}(0) - T_0(0)\right)\right]$$

Substitute into A.9.4:

$$T_0(\Delta t) = T_0(0) + \left(2\kappa \sum_{i=1}^{p} \frac{1}{2\Delta x^2} \left[\left(T_{\widehat{m}_i}(0) - T_0(0) \right) + \left(T_{-\widehat{m}_i}(0) - T_0(0) \right) \right] \right) \Delta t$$

$$T_0(\Delta t) = T_0(0) + \frac{2\kappa \Delta t}{2\Delta x^2} \sum_{i=1}^{p} \left[\left(T_{\widehat{m}_i}(0) - T_0(0) \right) + \left(T_{-\widehat{m}_i}(0) - T_0(0) \right) \right]$$

Cancelling the 2s and recalling the definition $\lambda = \kappa / \Delta x^2$ (equation A.3.2):

$$T_0(\Delta t) = T_0(0) + \lambda \Delta t \sum_{i=1}^{p} \left[\left(T_{\widehat{m}_i}(0) - T_0(0) \right) + \left(T_{-\widehat{m}_i}(0) - T_0(0) \right) \right]$$

$T_0(\Delta t)$ is the next time step, so switching to alternative notation ($T_0(\Delta t) = T_0^{(+)}$) and reversing the exterior and interior signs of the summation:

$$T_0^{(+)} = T_0(0) - \lambda \Delta t \sum_{i=1}^{p} \left[\left(T_0^{(0)} - T_{\widehat{m}_i}^{(0)} \right) + \left(T_0^{(0)} - T_{-\widehat{m}_i}^{(0)} \right) \right]$$

This is identical to A.4.1; the Forward-Euler method.

A.10 | A Stability Lemma

Given an infinite grid with one non-zero value, divided into A values and B values and equation 4.2.5:

$$\sum A^{(n+1)} - \sum B^{(n+1)} = (x - 4y)\left(\sum A^{(n)} - \sum B^{(n)}\right)$$

For any bound, is there a value at some timestep exceeding that bound? This is the subject of the lemma.

The proof is based on the idea that the absolute value of the sum of a class of values increases exponentially, but the non-zero values in that sum only cover a quadratically expanding area.

If the grid is finite but is able to be classified into A and B values to allow equation 4.2.5, then a similar proof is based on the idea that the absolute value of the sum of a class of values increases exponentially, but its values cover at most a fixed finite area. I have done some testing and, fascinatingly, if a fixed grid cannot be divided into A and B values to allow equation 4.2.5 (by having uneven dimensions), then the stability condition $0 \leq x \leq 1$ does not apply.

Let $T_{0,0}^{(0)}$ with relative subscripting be the single positive non-zero starting value, i.e.:

$$T_{i,j}^{(0)} = 0 \; except \; i = j = 0, T_{0,0}^{(0)} > 0 \qquad \text{- Eq A.10.1}$$

Let P_n represent the statement that:

$$T_{i,j}^{(n)} = 0 \; for \; |i| + |j| > n \qquad \text{- Eq A.10.2}$$

For $n = 0$, if $|i| + |j| > n = 0$ then we can't have that $i = j = 0$ because then $|i| + |j| = |0| + |0| = 0$. Hence from equation A.10.1, $T_{i,j}^{(0)} = 0 \; for \; |i| + |j| > 0$, hence P_0 is true.

If P_n for some n,

If $|i| + |j| > n + 1$ then $|i| + |j| > n$, $|i - 1| + |j| > n$, $|i| + |j - 1| > n$, $|i + 1| + |j| > n$, $|i| + |j + 1| > n$

Hence $|i| + |j| > n + 1$ implies $T_{i,j}^{(n)}$ and all its neighbours are zero because of P_n.

$$T_{i,j}^{(n)} = T_{i-1,j}^{(n)} = T_{i,j-1}^{(n)} = T_{i+1,j}^{(n)} = T_{i,j+1}^{(n)} = 0$$

From equation 2.3.8,

$$T^{(n+1)} = xT^{(n)} + y\sum N^{(n)}$$

Hence

$$T_{i,j}^{(n+1)} = xT_{i,j}^{(n)} + y\left(T_{i-1,j}^{(n)} + T_{i,j-1}^{(n)} + T_{i+1,j}^{(n)} + T_{i,j+1}^{(n)}\right)$$

$$T_{i,j}^{(n+1)} = x * 0 + y(0 + 0 + 0 + 0)$$

$$T_{i,j}^{(n+1)} = 0$$

Therefore P_n implies $T_{i,j}^{(n+1)} = 0 \; for \; |i| + |j| > n + 1$, hence P_n implies P_{n+1}.

Since $P_0 \wedge (P_n \Longrightarrow P_{n+1})$, P_n is true for all $n \geq 0$. Therefore:

$$\left(T_{i,j}^{(n)} = 0 \; for \; |i| + |j| > n\right) \forall n \geq 0$$

From this, we have

$$T_{i,j}^{(n)} \neq 0 \Longrightarrow |i| + |j| \leq n$$

If $|i| + |j| \leq n$ then $|i| \leq n$ and $|j| \leq n$, which defines a 2n+1 by 2n+1 square region. So there are at most $(2n + 1)^2$ non-zero values at any timestep n.

From equation 4.2.6, we have:

$$\sum A^{(n+h)} - \sum B^{(n+h)} = (2x - 1)^h \left(\sum A^{(n)} - \sum B^{(n)}\right)$$

$$\sum A^{(n)} - \sum B^{(n)} = (2x - 1)^n \left(\sum A^{(0)} - \sum B^{(0)}\right)$$

Take the absolute value of both sides:

$$\left|\sum A^{(n)} - \sum B^{(n)}\right| = |2x - 1|^n \left|\sum A^{(0)} - \sum B^{(0)}\right|$$

Since at n=0 the only non-zero value is $T_{0,0}^{(0)}$,

$$\left|\sum A^{(n)} - \sum B^{(n)}\right| = T_{0,0}^{(0)} |2x - 1|^n$$

Let $|2x - 1| = r$, so if we don't have the claimed stability condition $0 \leq x \leq 1$ then $r > 1$.

$$\left|\sum A^{(n)} - \sum B^{(n)}\right| = T_{0,0}^{(0)} r^n$$

Let $G^{(n)}$ be the class with the greater sum, $L^{(n)}$ be the class with the lesser sum (these may switch each timestep).

$$\sum G^{(n)} - \sum L^{(n)} = T_{0,0}^{(0)} r^n \qquad \text{- Eq A.10.3}$$

There is a small complication here since $\sum L^{(n)}$ may be (in fact usually is) negative. First, we show that the sum of all temperatures remains constant, starting from equation 2.3.8:

$$T^{(n+1)} = xT^{(n)} + y \sum N^{(n)}$$

And do a sum across all temperatures:

$$\sum T^{(n+1)} = x \sum T^{(n)} + 4y \sum T^{(n)}$$

$$\sum T^{(n+1)} = (x + 4y) \sum T^{(n)}$$

Substitute $x + 4y = 1$ (equation 2.3.9):

$$\sum T^{(n+1)} = \sum T^{(n)}$$

$$\sum T^{(n)} = \sum T^{(0)}$$

$$\sum T^{(n)} = T_{0,0}^{(0)} \quad \text{- Eq A.10.4}$$

Now each T value is labelled either A or B, hence G and L cover all temperature values, hence:

$$\sum G^{(n)} + \sum L^{(n)} = \sum T^{(n)}$$

Substitute in equation A.10.4:

$$\sum G^{(n)} + \sum L^{(n)} = T_{0,0}^{(0)}$$

Now add this to equation A.10.3:

$$\sum G^{(n)} - \sum L^{(n)} + \sum G^{(n)} + \sum L^{(n)} = T_{0,0}^{(0)} r^n + T_{0,0}^{(0)}$$

$$2 \sum G^{(n)} > T_{0,0}^{(0)} r^n$$

$$\sum G^{(n)} > \frac{1}{2} T_{0,0}^{(0)} r^n \quad \text{- Eq A.10.5}$$

Do a Taylor series in $f(n) = r^n$:

$$f(n) = e^{\log(r) n}$$

$$f(n) = 1 + \log(r)\, n + \frac{1}{2}[\log(r)\, n]^2 + \frac{1}{6}[\log(r)\, n]^3 + \cdots$$

$$f(n) > \frac{1}{6}[\log(r)\, n]^3$$

$$(Because\ \log(r) > 0\ since\ r > 1)$$

$$f(n) > \left[\frac{1}{6}\log^3(r)\right] n^3$$

$$r^n > \left[\frac{1}{6}\log^3(r)\right] n^3 \quad \text{- Eq A.10.6}$$

Substitute equation A.10.4 into equation A.10.5:

$$\sum G^{(n)} > \frac{1}{2} T_{0,0}^{(0)} \left[\frac{1}{6}\log^3(r)\right] n^3$$

Since at any timestep n there are at most $(2n+1)^2$ non-zero values, there are also at most $(2n+1)^2$ G values, regardless of how G values are distributed. Hence the average of the non-zero G values is greater than the following:

$$\frac{\sum G_{\neq 0}^{(n)}}{(2n+1)^2} > \frac{\frac{1}{2} T_{0,0}^{(0)} \left[\frac{1}{6}\log^3(r)\right] n^3}{(2n+1)^2}$$

$$\frac{\sum G_{\neq 0}^{(n)}}{(2n+1)^2} > \left[\frac{1}{6}\log^3(r)\, T_{0,0}^{(0)}\right] \frac{n^3}{2n^2+4n+1}$$

$$\frac{\sum G_{\neq 0}^{(n)}}{(2n+1)^2} > \left[\frac{1}{6}\log^3(r)\, T_{0,0}^{(0)}\right] \frac{n}{2+\frac{4}{n}+\frac{1}{n^2}}$$

$$(for\ n > 0)$$

Now $2 + 4/n + 1/n^2$ is monotonically decreasing, with its greatest value at $n = 1$ (restricted to the domain $n > 0$), so:

$$\frac{\sum G_{\neq 0}^{(n)}}{(2n+1)^2} > \left[\frac{1}{12}\log^3(r)\, T_{0,0}^{(0)}\right] \frac{n}{2+\frac{4}{1}+\frac{1}{1^2}}$$

$$\frac{\sum G_{\neq 0}^{(n)}}{(2n+1)^2} > \left[\frac{1}{84}\log^3(r)\, T_{0,0}^{(0)}\right] n \qquad \text{- Eq A.10.7}$$

A collection of values must have at least one value greater than or equal to its average. This is applied using a proof by contradiction.

Suppose $G_{\neq 0}^{(n)} \leq \left[\frac{1}{84}\log^3(r)\, T_{0,0}^{(0)}\right] n$ for all $G_{\neq 0}^{(n)}$, then since there are at most $(2n+1)^2$ values in $G_{\neq 0}^{(n)}$, we have that:

$$\sum G_{\neq 0}^{(n)} \leq (2n+1)^2 \left[\frac{1}{84}\log^3(r)\, T_{0,0}^{(0)}\right] n$$

$$\frac{\sum G_{\neq 0}^{(n)}}{(2n+1)^2} \leq \left[\frac{1}{84}\log^3(r)\, T_{0,0}^{(0)}\right] n$$

But this contradicts equation A.10.5. Hence there exists at least one $G_{\neq 0}^{(n)}$ value, hence at least one $T^{(n)}$ value satisfying:

$$T_{i,j}^{(n)} > \left[\frac{1}{84}\log^3(r)\, T_{0,0}^{(0)}\right] n\ for\ some\ i,j,for\ each\ n > 0 \qquad \text{- Eq A.10.8}$$

Given a bound B, If we choose n such that:

$$n > \frac{84B}{\log^3(r)\, T_{0,0}^{(0)}}$$

Then:

$$\left[\frac{1}{84}\log^3(r)\, T_{0,0}^{(0)}\right] n > B$$

And there exists $T_{i,j}^{(n)}$ such that:

$$T_{i,j}^{(n)} > \left[\frac{1}{84}\log^3(r)\, T_{0,0}^{(0)}\right] n > B$$

Therefore there exists $T_{i,j}^{(n)}$ such that:

$$T_{i,j}^{(n)} > B$$