

Intro to Research Computing with Python: Partial Differential Equations

Erik Spence

SciNet HPC Consortium

27 November 2014

Today's class

Today we will discuss the following topics:

- Basic approaches to solving PDEs.
- How to discretize equations.
- How to implement boundary conditions.
- Implicit versus explicit approaches.

Partial Differential Equations

Partial differential equations (PDEs) are differential equations which contain derivatives of more than one variable.

$$A \frac{\partial^2 \Phi}{\partial x^2} + B \frac{\partial^2 \Phi}{\partial x \partial y} + C \frac{\partial^2 \Phi}{\partial y^2} = F \left(x, y, \Phi, \frac{\partial \Phi}{\partial x}, \frac{\partial \Phi}{\partial y} \right)$$

For A, B, C constant, three classes of PDEs show up repeatedly in physical systems.

- If $B^2 - 4AC < 0$, the equation is called *elliptic* (Laplace's equation).
- If $B^2 - 4AC = 0$, the equation is called *parabolic* (Diffusion equation, Navier-Stokes).
- If $B^2 - 4AC > 0$, the equation is called *hyperbolic* (Wave equation).

How do we solve these problems?

Let's look at parabolic equations; in particular, let us look at the heat equation (diffusion equation).

$$\frac{\partial T}{\partial t} = k \frac{\partial^2 T}{\partial x^2},$$

where T is the temperature and k is the thermal diffusivity.

How do we solve this equation? By discretizing in both space and time, and marching an initial condition forward in time.

This is similar to the initial-value ODE solutions we saw last class, but now we have another derivative to deal with.

Dealing with time

Rewrite the right-hand side as an operator:

$$\frac{\partial T}{\partial t} = k \frac{\partial^2 T}{\partial x^2} \quad \rightarrow \quad \frac{\partial T}{\partial t} = FT$$

Basic approaches to dealing with the time part of the equation:

- Explicit methods, such as forward Euler:

$$\frac{T_{i+1} - T_i}{\Delta t} = FT_i \quad \rightarrow \quad T_{i+1} = (1 + \Delta t F)T_i$$

- Implicit methods, such as backward Euler:

$$\frac{T_{i+1} - T_i}{\Delta t} = FT_{i+1} \quad \rightarrow \quad (1 - \Delta t F)T_{i+1} = T_i$$

Explicit Methods

Methods are called explicit when only T_i is on the right side of the equation. Explicit methods have some nice features:

- They are very easy to implement.
- They are usually quick to calculate (no matrix inversions).
- Easier to parallelize, since the calculation is inherently local.
- There exist more-accurate explicit methods than forward Euler.

$$\frac{T_{i+1} - T_i}{\Delta t} = FT_i \quad \rightarrow \quad T_{i+1} = (1 + \Delta t F)T_i$$

But there are some serious downsides as well:

- They are not very accurate at low order ($\mathcal{O}(\Delta t)$ for forward Euler).
- They can be numerically unstable (though there are exceptions).

Implicit methods

Methods are called implicit when T_{i+1} is on both sides of the equation. Examples include backward Euler:

$$\frac{T_{i+1} - T_i}{\Delta t} = FT_{i+1} \quad \mathcal{O}(\Delta t)$$

and Crank-Nicolson (sometimes called semi-implicit):

$$\frac{T_{i+1} - T_i}{\Delta t} = (FT_{i+1} + FT_i)/2 \quad \mathcal{O}(\Delta t^2)$$

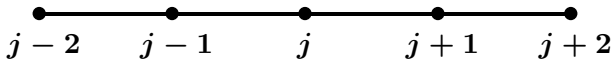
Implicit time stepping methods have some nice features:

- They are stable over a wide range of timestep sizes, sometimes unconditionally (not unconditionally accurate, though).
- Excellent for solving steady-state problems.

Downsides include being more difficult to code, and much more difficult to parallelize (inverting the operator depends upon all grid points).

Dealing with space

We need to calculate the second spatial derivatives. How best to do that? We discretize the x domain, and examine the Taylor expansion of some function f , centered around three different points:



$$f(x_{j-1}) = f(x_j) - (\Delta x) \frac{\partial f(x_j)}{\partial x} + \frac{(\Delta x)^2}{2!} \frac{\partial^2 f(x_j)}{\partial x^2} + \mathcal{O}(\Delta x^3)$$

$$f(x_j) = f(x_j)$$

$$f(x_{j+1}) = f(x_j) + (\Delta x) \frac{\partial f(x_j)}{\partial x} + \frac{(\Delta x)^2}{2!} \frac{\partial^2 f(x_j)}{\partial x^2} + \mathcal{O}(\Delta x^3)$$

Where $\Delta x = x_j - x_{j-1}$. This is known as the finite-difference method.

Calculating derivatives

We can write this as a matrix operation:

$$\begin{bmatrix} f(x_{j-1}) \\ f(x_j) \\ f(x_{j+1}) \end{bmatrix} = \begin{bmatrix} 1 & -\Delta x & \Delta x^2 \\ 1 & 0 & 0 \\ 1 & \Delta x & \Delta x^2 \end{bmatrix} \begin{bmatrix} f(x_j) \\ f'(x_j) \\ f''(x_j) \end{bmatrix}$$

To get the answer we invert the matrix:

$$\begin{bmatrix} f(x_j) \\ f'(x_j) \\ f''(x_j) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ \frac{-1}{2\Delta x} & 0 & \frac{1}{2\Delta x} \\ \frac{1}{\Delta x^2} & \frac{-2}{\Delta x^2} & \frac{1}{\Delta x^2} \end{bmatrix} \begin{bmatrix} f(x_{j-1}) \\ f(x_j) \\ f(x_{j+1}) \end{bmatrix}$$

$$f'(x_j) = \frac{\partial f(x_j)}{\partial x} = \frac{f(x_{j+1}) - f(x_{j-1})}{2\Delta x}$$

$$f''(x_j) = \frac{\partial^2 f(x_j)}{\partial x^2} = \frac{f(x_{j+1}) - 2f(x_j) + f(x_{j-1}))}{\Delta x^2}$$

Discretizing our equation

Let us discretize the variable T in both time and space, with $T(t_i, x_j) = T_{i,j}$. This then gives us

$$\begin{aligned}\frac{\partial T}{\partial t} &= k \frac{\partial^2 T}{\partial x^2}, \\ \frac{\partial T_{i,j}}{\partial t} &= k \frac{\partial^2 T_{i,j}}{\partial x^2}, \\ &= k \left[\frac{T_{i,j+1} - 2T_{i,j} + T_{i,j-1}}{\Delta x^2} \right].\end{aligned}$$

Discretizing our equation, continued

$$\frac{\partial T_{i,j}}{\partial t} = k \left[\frac{T_{i,j+1} - 2T_{i,j} + T_{i,j-1}}{\Delta x^2} \right]$$

If we write $T_{i,j}$ as a vector of values, we can rewrite our equation as a matrix operation. Note that there is one equation for each spatial point:

$$\frac{\partial}{\partial t} \begin{bmatrix} \vdots \\ T_{i,17} \\ T_{i,18} \\ T_{i,19} \\ T_{i,20} \\ T_{i,21} \\ \vdots \end{bmatrix} = k \begin{bmatrix} \vdots & & & & & & \\ \cdots & \frac{1}{\Delta x^2} & \frac{-2}{\Delta x^2} & \frac{1}{\Delta x^2} & 0 & 0 & \cdots \\ \cdots & 0 & \frac{1}{\Delta x^2} & \frac{-2}{\Delta x^2} & \frac{1}{\Delta x^2} & 0 & \cdots \\ \cdots & 0 & 0 & \frac{1}{\Delta x^2} & \frac{-2}{\Delta x^2} & \frac{1}{\Delta x^2} & \cdots \\ \vdots & & & & & & \end{bmatrix} \begin{bmatrix} \vdots \\ T_{i,17} \\ T_{i,18} \\ T_{i,19} \\ T_{i,20} \\ T_{i,21} \\ \vdots \end{bmatrix}$$

Which we can write as $\frac{\partial \vec{T}_i}{\partial t} = F \vec{T}_i$.

What about the edges?

The edges are a problem. Why? Well, consider the first point, $j = 0$:

$$\frac{\partial T_{i,0}}{\partial t} = k \left[\frac{T_{i,1} - 2T_{i,0} + T_{i,-1}}{\Delta x^2} \right]$$

There is no $j = -1$ point!

The solution is to not use the above equation to describe the edge points. Use a different equation instead. These are known as 'boundary conditions'; there are two general classes:

- Dirichlet: constant boundary value.
- Neumann: boundary values based on derivatives of the boundary values themselves.

How these conditions are implemented depends on the approach to solving the equation that is being used.

Example problem

Suppose we have a rod, of length 1. At $x = 0$ the temperature varies as $T(t, 0) = \sin(10t)$. At $x = 1$ the edge is kept at a constant temperature of $T(t, 1) = 0$. The thermal diffusivity is $k = 0.2$. Show how the temperature evolves in time and space.

How should we solve this problem?

As mentioned last lecture, there are two basic classes of approaches:

- explicit methods
- implicit methods

And combinations thereof.

Using the implicit method

As mentioned last lecture, explicit methods can be unstable. We will solve this using an implicit method. We start by returning to our equation:

$$\frac{\partial \vec{T}_i}{\partial t} = F \vec{T}_i$$

We rewrite this in the implicit form as:

$$\begin{aligned}\frac{\vec{T}_{i+1} - \vec{T}_i}{\Delta t} &= F \vec{T}_{i+1} \\ \vec{T}_{i+1} - \Delta t F \vec{T}_{i+1} &= \vec{T}_i \\ (\mathbb{1} - \Delta t F) \vec{T}_{i+1} &= \vec{T}_i\end{aligned}$$

where $\mathbb{1}$ is the identity matrix. Note that this equation takes the form $Ax = b$.

Implementing boundary conditions

The boundary conditions are implemented by modifying the operator to implement different equations.

$$(\mathbb{1} - \Delta t F) \vec{T}_{i+1} = \vec{T}_i$$

The equation for the boundary condition at $j = 0$ ($T_{i+1,0} = \sin(10t_{i+1})$) is then implemented by:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \dots \\ -\alpha & 1 + 2\alpha & -\alpha & 0 & 0 & \dots \\ 0 & -\alpha & 1 + 2\alpha & -\alpha & 0 & \dots \\ 0 & 0 & -\alpha & 1 + 2\alpha & -\alpha & \dots \\ & & & \vdots & & \end{bmatrix} \begin{bmatrix} T_{i+1,0} \\ T_{i+1,1} \\ T_{i+1,2} \\ T_{i+1,3} \\ \vdots \end{bmatrix} = \begin{bmatrix} \sin(10t_{i+1}) \\ T_{i,1} \\ T_{i,2} \\ T_{i,3} \\ \vdots \end{bmatrix}$$

Where we have defined a new constant: $\alpha = \Delta t k / (\Delta x^2)$.

Implementing the other boundary condition

$$(\mathbb{1} - \Delta t F) \vec{T}_{i+1} = \vec{T}_i$$

Assume that there are 100 points in x . The boundary condition equation at $x = 1$ ($T_{i+1,99} = 0$) is implemented similarly:

$$\begin{bmatrix} \vdots \\ \dots & -\alpha & 1 + 2\alpha & -\alpha & 0 & 0 \\ \dots & 0 & -\alpha & 1 + 2\alpha & -\alpha & 0 \\ \dots & 0 & 0 & -\alpha & 1 + 2\alpha & -\alpha \\ \dots & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \vdots \\ T_{i+1,96} \\ T_{i+1,97} \\ T_{i+1,98} \\ T_{i+1,99} \end{bmatrix} = \begin{bmatrix} \vdots \\ T_{i,97} \\ T_{i,97} \\ T_{i,98} \\ 0 \end{bmatrix}$$

The algorithm

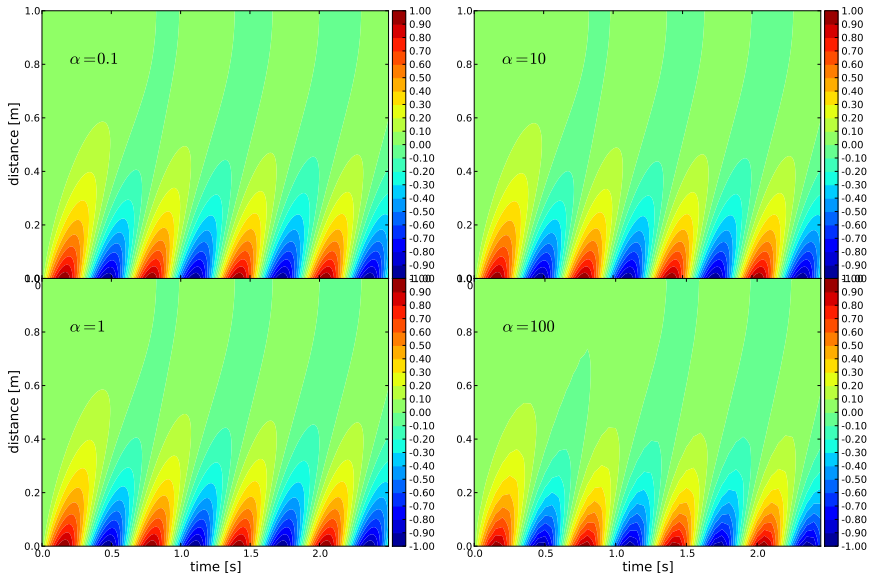
$$(\mathbb{1} - \Delta t F) \vec{T}_{i+1} = \vec{T}_i$$

So what is the actual process?

- ① Build the matrix operator $(\mathbb{1} - \Delta t F)$.
- ② Modify the operator to deal with the boundary conditions.
- ③ Initialize \vec{T} to zero.
- ④ Then loop:
 - ① Copy \vec{T}_i to a temporary variable \vec{b} .
 - ② Reset the edge values ($j = 0, 99$) of \vec{b} to implement the boundary conditions.
 - ③ Solve $(\mathbb{1} - \Delta t F) \vec{T}_{i+1} = \vec{b}$.

And repeat until done.

Results



Notes about the example

Some things to note:

- When solving $(\mathbb{1} - \Delta t F) \vec{T}_{i+1} = \vec{b}$ do NOT invert the operator. This may seem to be the intuitive thing to do, but it is incorrect. There are better, faster, and more accurate algorithms for solving this problem, such as `linalg.solve(A,b)`.
- Because we are using an implicit method, this solution will be numerically stable for arbitrary timestep size (this can be demonstrated mathematically). However, for accuracy of solution one must have $\alpha \ll 1$; this keeps the matrix dominated by the diagonal.
- We've used the most general method to solve $Ax = b$. However, you will notice that our operator is banded. In such situations there are special algorithms for solving $Ax = b$, the one to use in Python is `linalg.solve_banded`.

Using the explicit method

As mentioned last lecture, explicit methods can be unstable. Let's see how it performs in this case. This is the implicit version of the equation:

$$\frac{\vec{T}_{i+1} - \vec{T}_i}{\Delta t} = F\vec{T}_{i+1}$$

This is the explicit version:

$$\begin{aligned}\frac{\vec{T}_{i+1} - \vec{T}_i}{\Delta t} &= F\vec{T}_i \\ \vec{T}_{i+1} &= \vec{T}_i + \Delta t F\vec{T}_i \\ \vec{T}_{i+1} &= (\mathbb{1} + \Delta t F) \vec{T}_i\end{aligned}$$

This is obviously much more direct.

Implementing boundary conditions

The boundary conditions are once again implemented by modifying the operator.

$$\vec{T}_{i+1} = (\mathbb{1} + \Delta t F) \vec{T}_i$$

The equation for the boundary condition at $j = 0$ ($T_{i+1,0} = \sin(10t_{i+1})$) is then implemented by:

$$\begin{bmatrix} T_{i+1,0} \\ T_{i+1,1} \\ T_{i+1,2} \\ \vdots \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \dots \\ \alpha & 1 - 2\alpha & \alpha & 0 & 0 & \dots \\ 0 & \alpha & 1 - 2\alpha & \alpha & 0 & \dots \\ & & & \vdots & & \end{bmatrix} \begin{bmatrix} \sin(10t_{i+1}) \\ T_{i,1} \\ T_{i,2} \\ \vdots \end{bmatrix}$$

Where we have used the same definition: $\alpha = \Delta t k / (\Delta x^2)$.

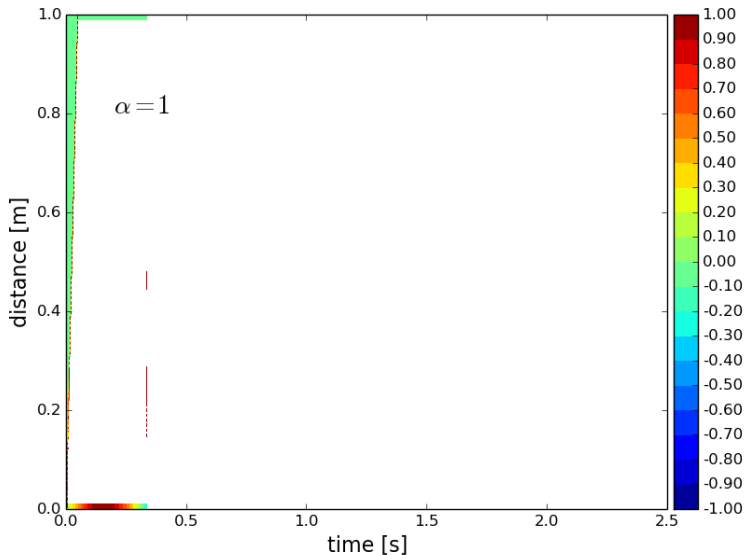
Implementing the other boundary condition

$$\vec{T}_{i+1} = (\mathbb{1} + \Delta t F) \vec{T}_i$$

Again assuming that there are 100 points in x . The boundary condition equation at $x = 1$ ($T_{i+1,99} = 0$) is implemented similarly:

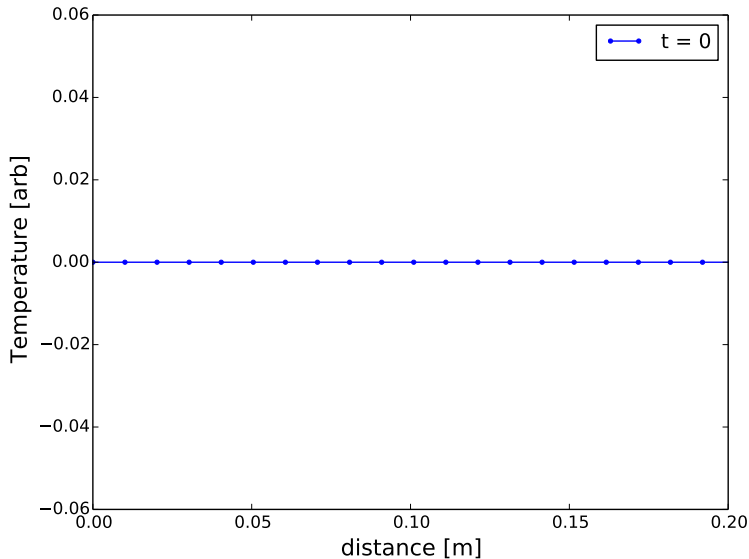
$$\begin{bmatrix} \vdots \\ T_{i+1,96} \\ T_{i+1,97} \\ T_{i+1,98} \\ T_{i+1,99} \end{bmatrix} = \begin{bmatrix} \vdots & & & & & & \\ \dots & \alpha & 1 - 2\alpha & \alpha & 0 & 0 & \\ \dots & 0 & \alpha & 1 - 2\alpha & \alpha & 0 & \\ \dots & 0 & 0 & \alpha & 1 - 2\alpha & \alpha & \\ \dots & 0 & 0 & 0 & 0 & 1 & \end{bmatrix} \begin{bmatrix} \vdots \\ T_{i,97} \\ T_{i,97} \\ T_{i,98} \\ 0 \end{bmatrix}$$

Explicit results

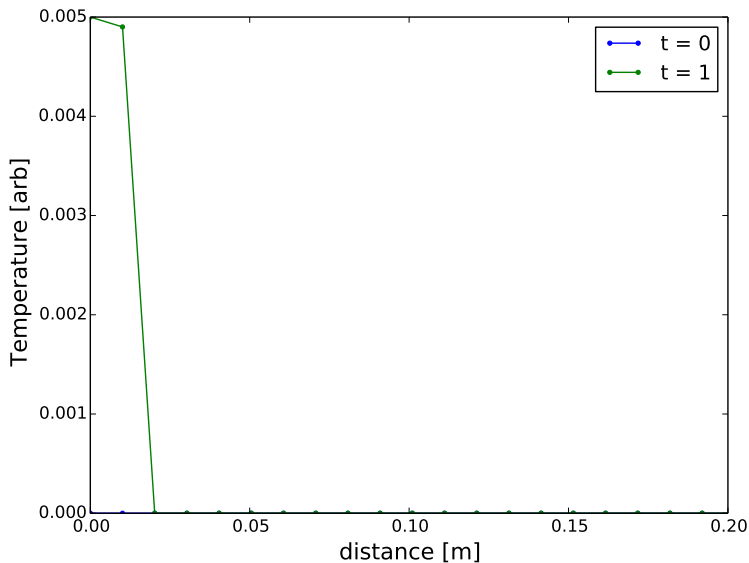


There's a problem here.

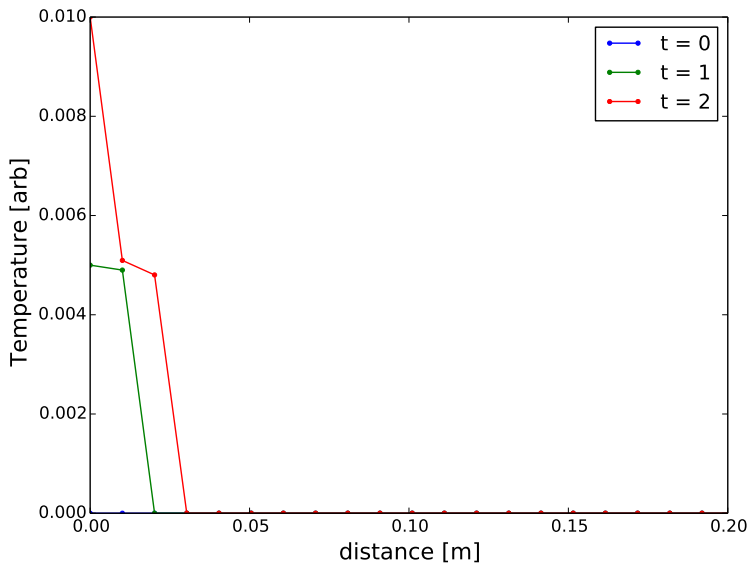
What happened?



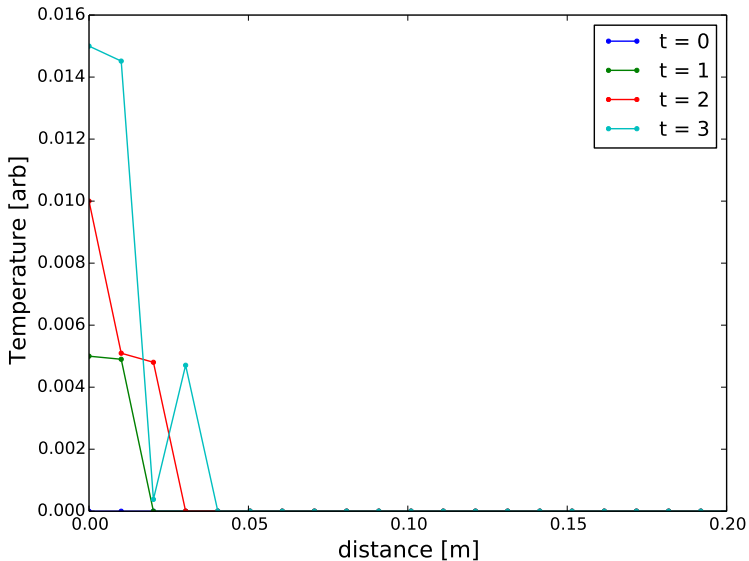
What happened?



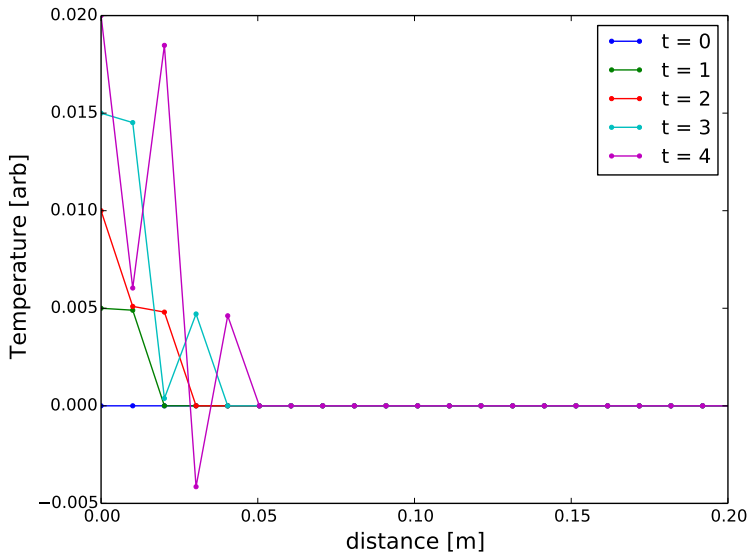
What happened?



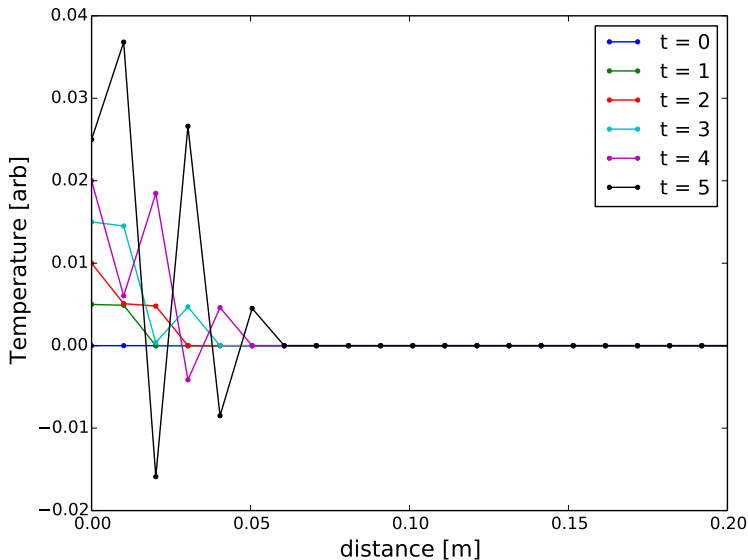
What happened?



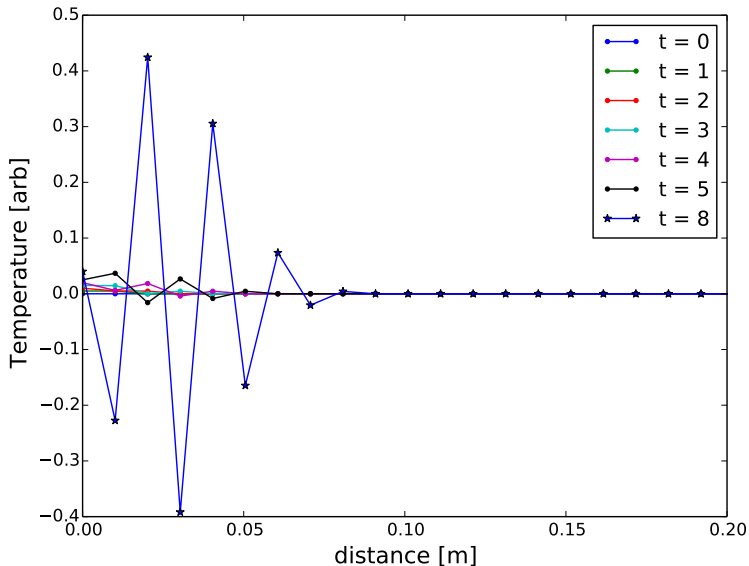
What happened?



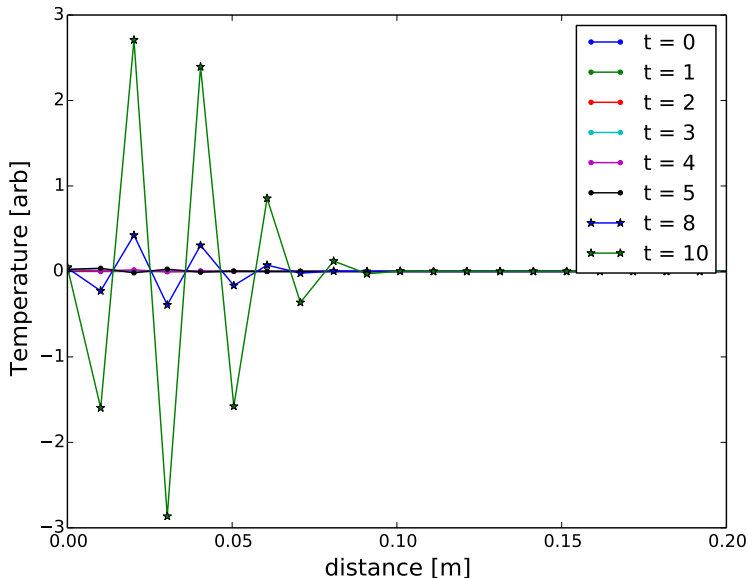
What happened?



What happened?



What happened?



What went wrong?

Consider a 5-point version of the problem, given by the operator below.

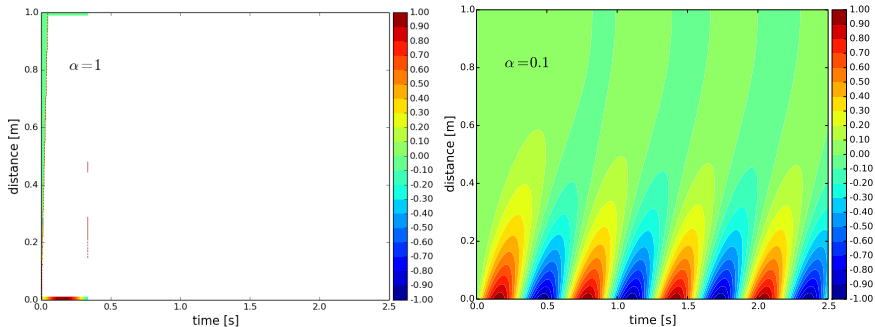
$$\begin{bmatrix} T_{i+1,0} \\ T_{i+1,1} \\ T_{i+1,2} \\ T_{i+1,3} \\ T_{i+1,4} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ \alpha & 1 - 2\alpha & \alpha & 0 & 0 & 0 \\ 0 & \alpha & 1 - 2\alpha & \alpha & 0 & 0 \\ 0 & 0 & \alpha & 1 - 2\alpha & \alpha & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \sin(10t_{i+1}) \\ T_{i,1} \\ T_{i,2} \\ T_{i,3} \\ 0 \end{bmatrix}$$

What are the conditions for this to be stable? We require the eigenvalues of the operator $|\lambda| \leq 1$, and thus $\alpha = \frac{\Delta t k}{\Delta x^2} \leq 0.5$. Which gives

$$\Delta t \leq \frac{\Delta x^2}{2k}$$

If you double the spatial resolution, you need to quadruple the temporal resolution, for stability.

Can we save it?



Yes, we just need to pick a better value of α .

Homework 4, scenario

Consider this scenario:

You live in Buffalo, in an apartment building which holds 500 people. After 2 metres of snow, everyone is trapped in the building. One apartment's dwellers suddenly become zombies. The zombies go rampaging through the building, turning the other apartment dwellers into zombies.

Fortunately, some (9) people know how to kill zombies, and they are able to teach the other people who live in the apartment building. The chance of surviving a zombie encounter differs between zombie killers and non-zombie killers, but if a person doesn't survive the encounter, they become a zombie.

Homework 4, equations

Consider the Modified Zombie Apocalypse equations:

$$\frac{\partial S}{\partial t} = -BSZ - ESK$$

$$\frac{\partial K}{\partial t} = -CKZ + ESK$$

$$\frac{\partial Z}{\partial t} = BSZ + CKZ - AKZ$$

Use ODEINT to solve this system of equations.

- S : number of regular people, who can't kill zombies
- K : number of zombie killers
- Z : number of zombies
- A : rate at which zombies are killed, by K
- B : rate at which regular people are turned into zombies
- C : rate at which zombie killers are turned into zombies
- E : rate at which zombie killers teach regular people how to kill zombies

Inspired by Munz *et al.*, Infectious Disease Modelling Research Progress, 2009

Homework 4, question

$$\frac{\partial S}{\partial t} = -BSZ - ESK$$

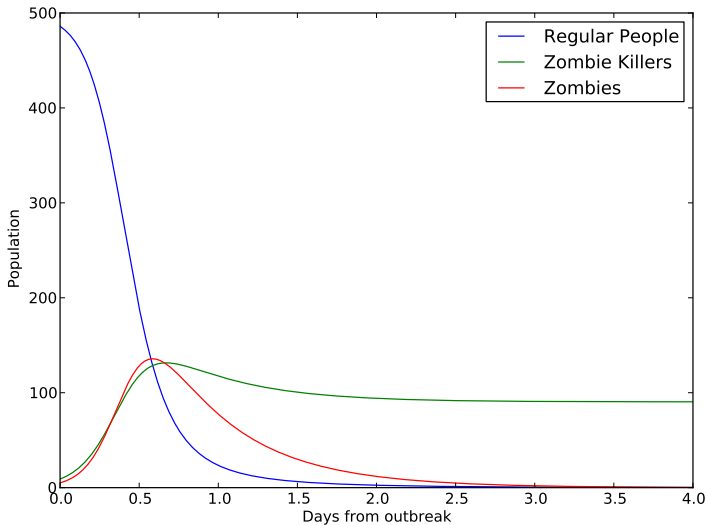
$$\frac{\partial K}{\partial t} = -CKZ + ESK$$

$$\frac{\partial Z}{\partial t} = BSZ + CKZ - AKZ$$

Assuming values $B = 0.02$, $E = 0.015$, $A = 0.03$, $C = 0.01$, $K_0 = 9$ (K at $t = 0$), how many initial zombies, Z_0 , does it take to turn the whole building population into zombies? Note that $S_0 = 491 - Z_0$.

Solve this for several values of Z_0 . Produce at least 2 plots, of the three populations versus time, one where the zombies win, and all normal people disappear, and one where they lose, and the zombies disappear. Submit your code, plots and 'hg log' output.

Results



$$A = 0.03, B = 0.02, C = 0.01, E = 0.015, Z_0 = 5$$