Intro to Research Computing with Python: Partial Differential Equations

Erik Spence

SciNet HPC Consortium

28 November 2013

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.

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

For $\boldsymbol{A, B, C}$ constant, three classes of PDEs show up repeatedly in physical systems.

- If $\boldsymbol{B^2 - 4AC < 0}$, the equation is called *elliptic*.
- If $\boldsymbol{B^2 - 4AC = 0}$, the equation is called *parabolic* (diffusive).
- If $\boldsymbol{B^2 - 4AC > 0}$, the equation is called *hyperbolic* (wavelike).

How do we solve these problems?

Let's start by looking at parabolic equations; in particular, let us look at the heat equation.

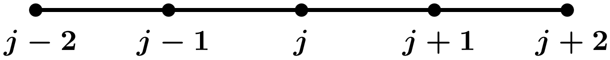
$$\frac{\partial T}{\partial t} = k\frac{\partial^2 T}{\partial x^2},$$

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.

Calculating derivatives

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:



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

Where $\Delta x = x_j - x_{j-1}$.

Calculating derivatives, continued

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:

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

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

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 use 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 as

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

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.

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

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

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

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

Implementing another boundary condition

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

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

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

The algorithm

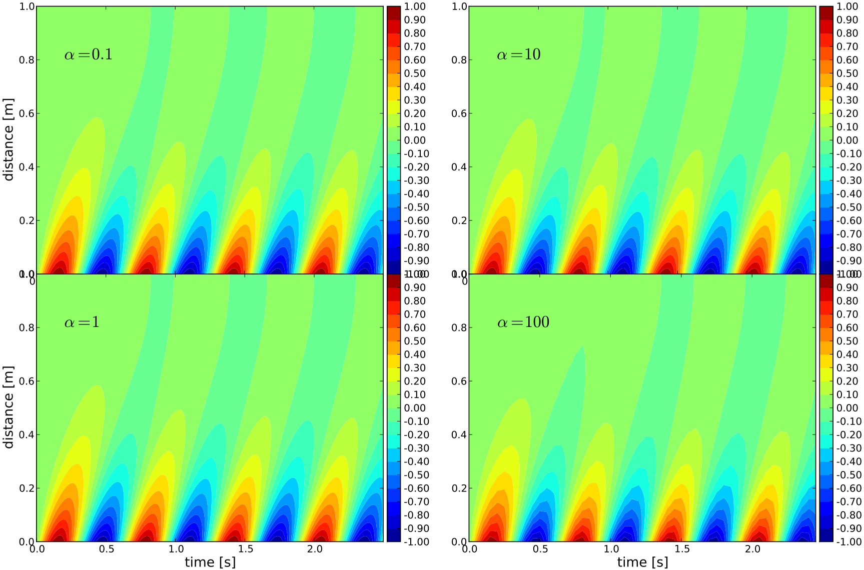
$$(\mathbb{1} - \Delta t F)\,\vec{T}_{i+1} = \vec{T}_i$$

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 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



α = 0.1
α = 10
α = 1
α = 100
distance [m]
time [s]
0.0
0.2
0.4
0.6
0.8
1.0
0.5
1.5
2.0
1.00
0.90
0.80
0.70
0.60
0.50
0.40
0.30
0.20
0.10
0.00
-0.10
-0.20
-0.30
-0.40
-0.50
-0.60
-0.70
-0.80
-0.90
-1.00

Notes about the example

Some things to note:

- When solving $(\mathbb{1} - \boldsymbol{\Delta t F})\vec{\boldsymbol{T}}_{\boldsymbol{i+1}} = \vec{\boldsymbol{b}}$ 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 $\boldsymbol{\alpha} \ll \mathbf{1}$. This keeps the matrix diagonal and increases accuracy.
- We've used the most general method to solve $\boldsymbol{Ax} = \boldsymbol{b}$. However, you will notice that our operator is banded. In such situations there are special algorithms for solving $\boldsymbol{Ax} = \boldsymbol{b}$, 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. We start by returning to our equation:

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

We rewrite this as

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

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: $\alpha = -\Delta t k/(\Delta x^2)$.

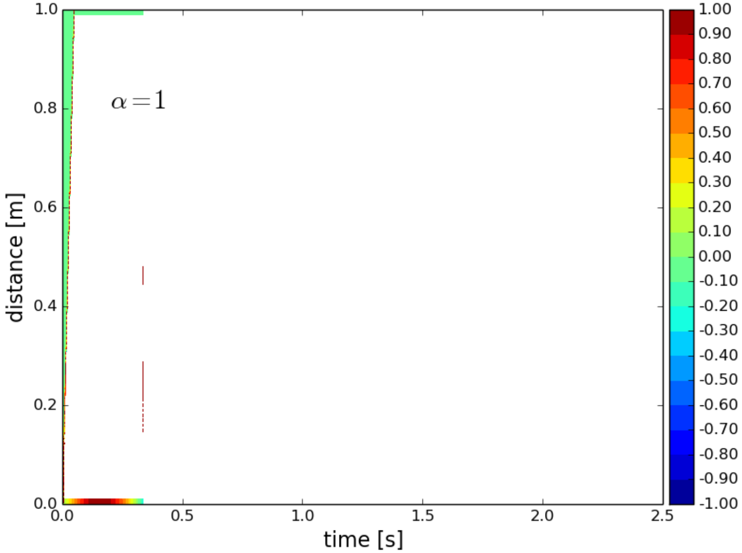
Implementing another boundary condition

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

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

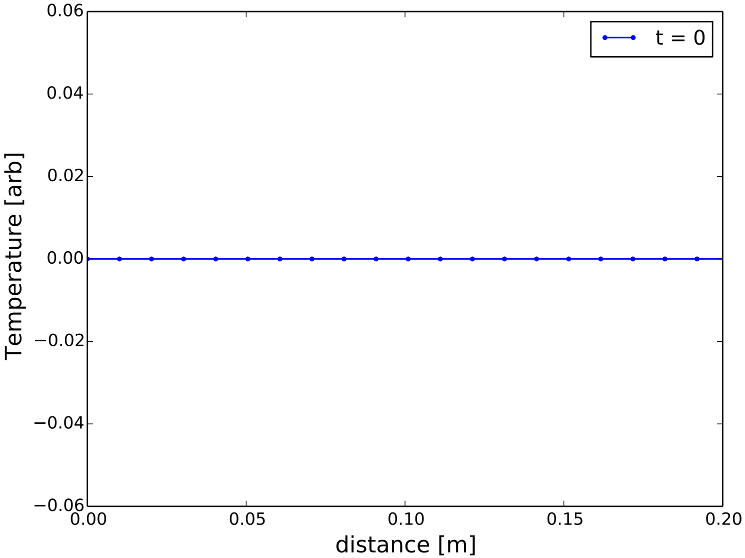
$$\begin{bmatrix} \vdots \\ T_{i+1,96} \\ T_{i+1,97} \\ T_{i+1,98} \\ T_{i+1,99} \end{bmatrix} = \begin{bmatrix} & & & \vdots & & \\ \ldots & -\alpha & 1+2\alpha & -\alpha & 0 & 0 \\ \ldots & 0 & -\alpha & 1+2\alpha & -\alpha & 0 \\ \ldots & 0 & 0 & -\alpha & 1+2\alpha & -\alpha \\ \ldots & 0 & 0 & 0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

Explicit results

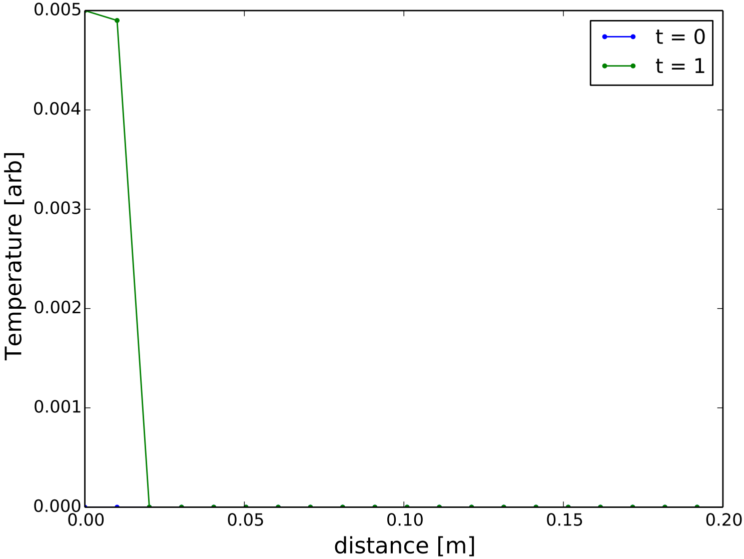


There's a problem here.

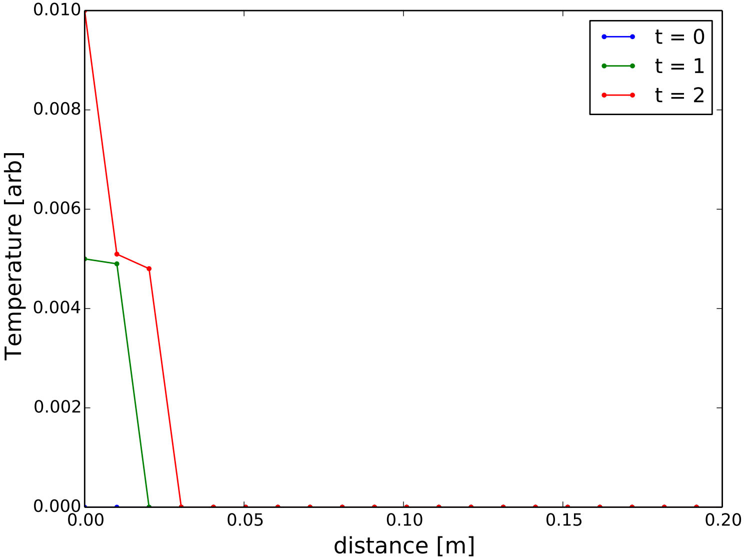
What happened?



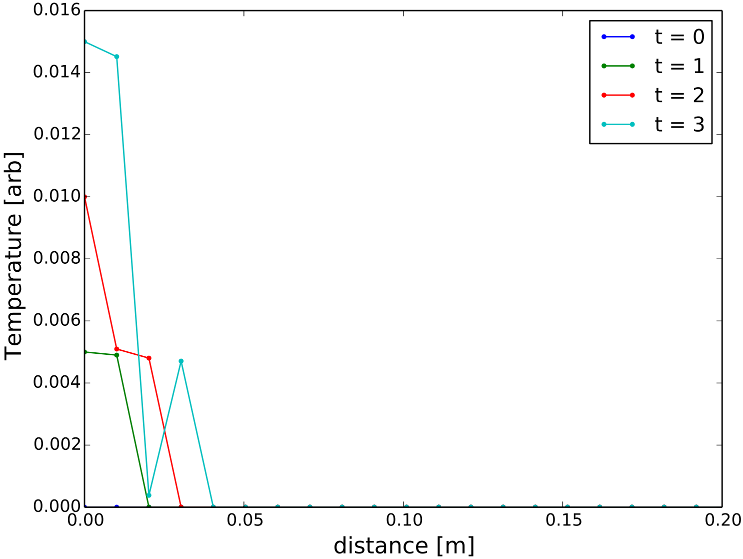
What happened?



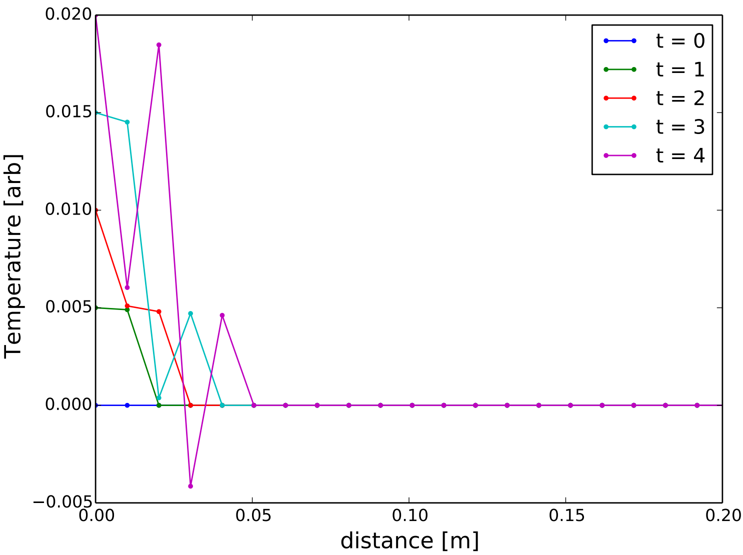
What happened?



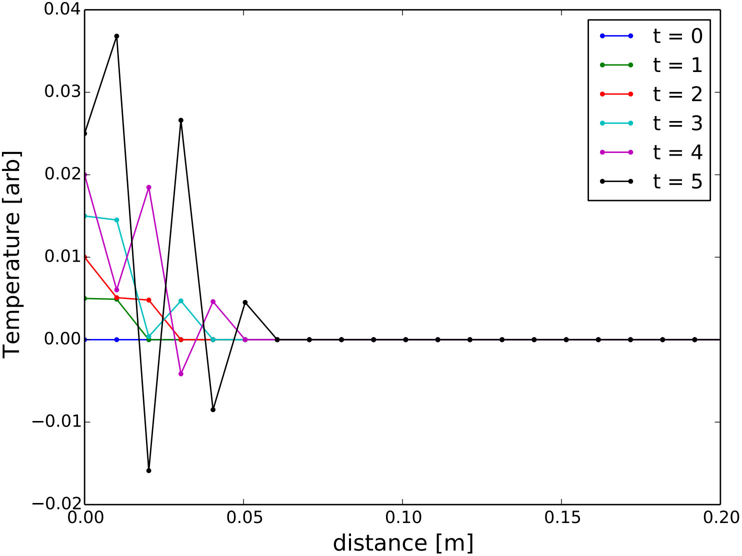
What happened?



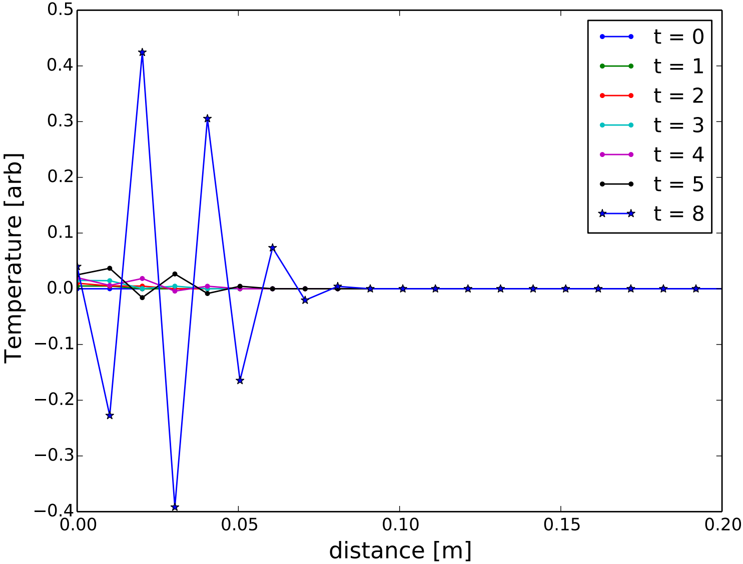
What happened?



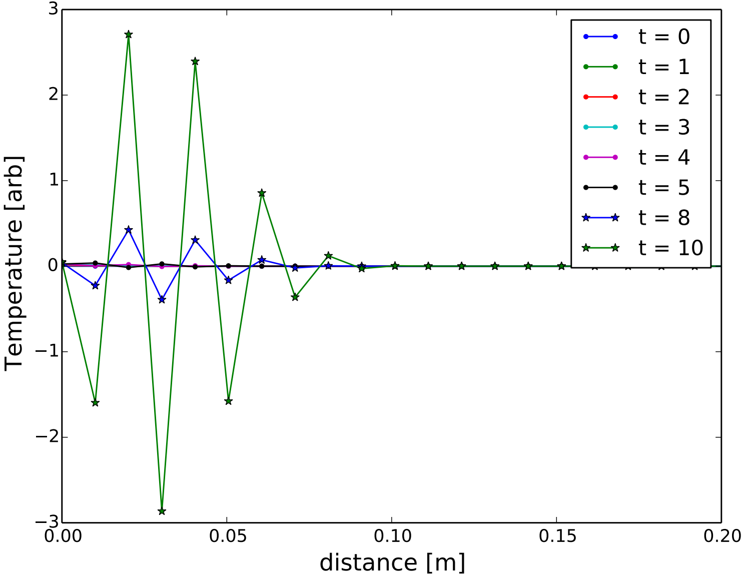
What happened?



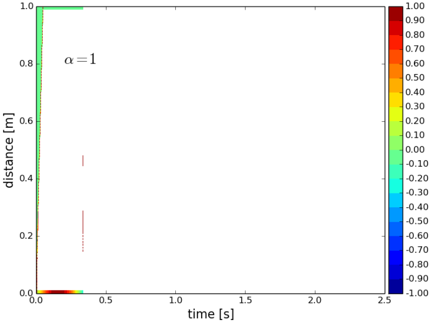
What happened?

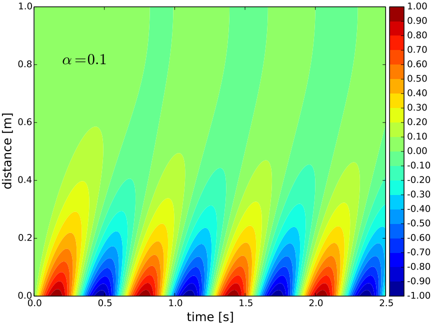


What happened?



Can we save it?





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

Homework 4

Consider a density column, $\rho(z,t)$, 10 meters high, which is described by the modified diffusion equation:

$$\frac{\partial \rho}{\partial t} = D\frac{\partial^2 \rho}{\partial z^2} + g\frac{\partial \rho}{\partial z}$$

with boundary conditions $\rho(0,t) = 1.0$ and $\rho(10,t) = 0.0$, and $D = 10$ and $g = 4$.

1. With the initial condition $\rho(z) = \cos(z\pi/20.0)$, and 100 points in z, evolve this system forward in time until a steady state is reached. Use an implicit method, with an appropriate timestep.
2. Plot snapshots of ρ versus z as the system approaches steady state. Note that the solution in steady state is *approximately* $\rho(z) = e^{-\frac{gz}{D}}$.
3. Submit your code, figure and 'hg log' output. Remember to comment your code.