In [1]:
```
%autosave 0
```

```
Autosave disabled
```

Chapter 15-16: PDEs Solvers Using Neural Network Methods

PDE study considered from the viewpoint of neural networks. Many PDEs describe the evolution of a spatially distributed system over time. The state of such a system is defined by a value $v(x, t)$ that depends on a spatial variable x that is usually a vector and on time t. The value itself can be a vector as well.

"Learning samples" can consist of discrete values of the initial condition $v_{i,0}(x_{i,0})$ and values $v_{ij}(x_i, t_j)$ at later times. The initial values are presented to the network as inputs. The outputs of the (recurrent) network at later times t_j are then compared with the sample values to calculate the error.

Since PDEs are often spatially homogeneous, it makes sense to use recurrent convolutional networks here. This is the same type that is often used in image processing.

The size of the convolutional kernel depends on the degree of spatial derivatives in the PDE:

The network needs to have at least one layer for each component of the value v. Additional hidden layers might be required, especially if the PDE has higher temporal derivatives.

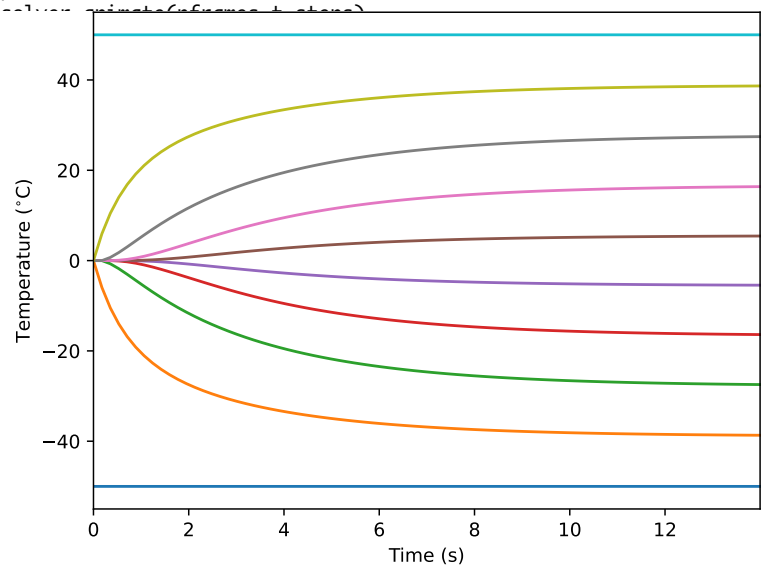
The example in the cell below was taken from the Github repository, <MatthewFilipovich / neural-network-pde-solver> . Besides this 1D heat equation example, examples for 2D heat and 1D wave are also contained there. Neural network PDE solving is a very active area, with lots of participants, and new solvers are coming out at a fast pace.

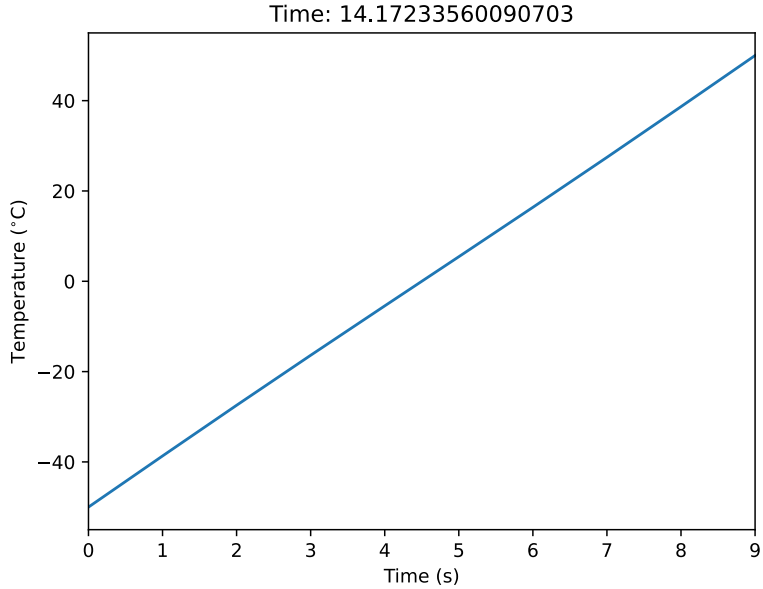
In this particular example, Mr. Filipovich has elected to show plots from new and older methods for comparison. (The 2nd plot shows a straight line, but if you look closely at the 5th plot, which is its companion, you will see that its line is not quite straight.)

The nengo_pde module is not hosted on Pypi, and cannot be installed by pip. But due to the presence of the *_init_*.py file in the module folder, placing the folder in the Jupyter home directory allows the nengo_pde module to be imported without difficulty into a Jupyter cell. (It is, however, necessary to install the nengo module from Pypi.)

In [14]:
```
"""Module solves heat equation in 1D using finite difference method and nengo_pde."""
from nengo_pde import Solver1D
import matplotlib.pyplot as plt
%config InlineBackend.figure_formats = ['svg']


def feedback_connection(u):
    return - (K/dx**2) * 2*u


def lateral_connection(u):
    return K/dx**2 * u


# Nengo simulation
t_steps = 80  # Number of time steps
x_steps = 8  # Number of x steps
neurons = 500  # Number of neurons
radius = 100  # Radius of neurons
boundaries = [-50, 50]  # Constant boundary conditions
solver = Solver1D(feedback_connection, lateral_connection)

# Grid properties
K = 4.2
x_len = 20  # mm
dx = x_len/x_steps
dt = dx**2/(2*K**2)  # dt chosen for stability

# Run finite difference method simulation
solver.run_FDM_order1(dt, t_steps, x_steps, boundaries)
fig, ax = solver.plot_population(dt, False)
```

```
ax.set_xlabel('Time (s)')
ax.set_ylabel('Temperature ($^{\circ}$C)')
plt.show()
fig, ax = solver.plot_grid(t_steps, False)
ax.set_xlabel('Time (s)')
ax.set_ylabel('Temperature ($^{\circ}$C)')
plt.show()
solver.animate(nframes=t_steps)

# Run nengo_pde simulation
solver.run_nengo_order1(dt, t_steps, x_steps, boundaries, neurons, radius)
fig, ax = solver.plot_population(0.001, False)
ax.set_xlabel('Time (s)')
ax.set_ylabel('Temperature ($^{\circ}$C)')
plt.show()
fig, ax = solver.plot_grid(t_steps, False)
ax.set_xlabel('Time (s)')
ax.set_ylabel('Temperature ($^{\circ}$C)')
plt.show()
```



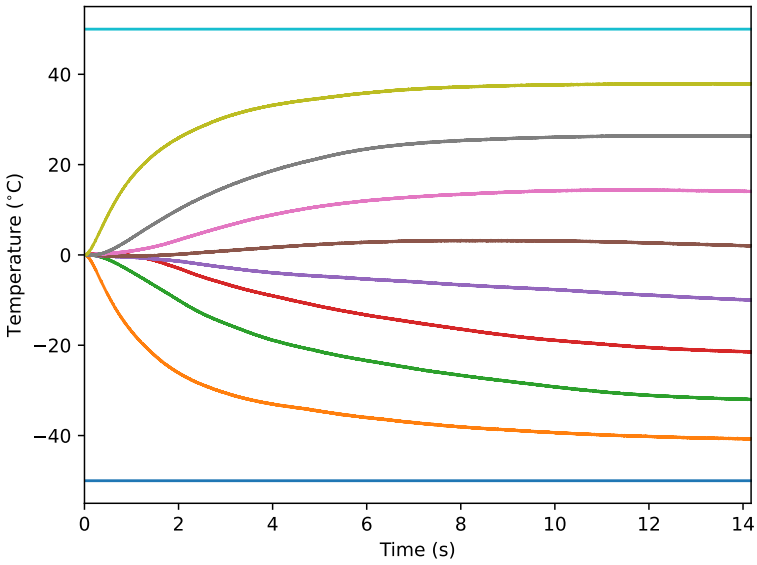


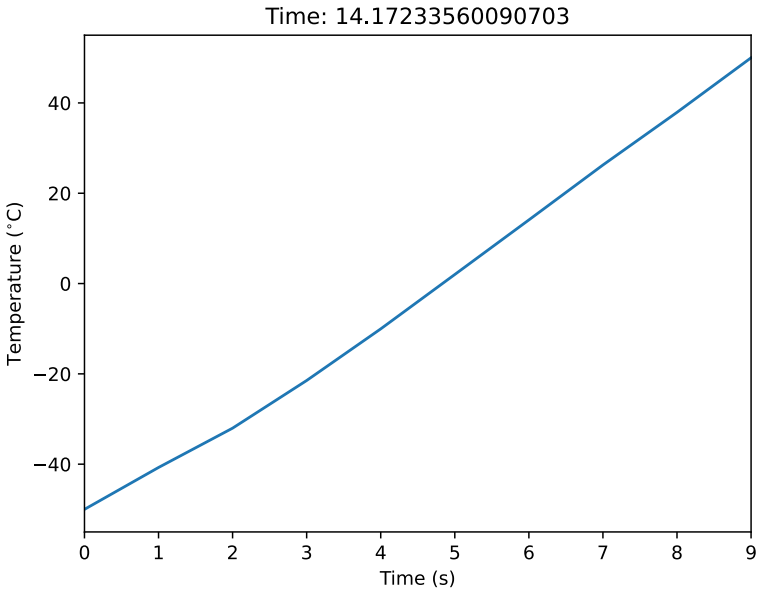
HMTL progress bar requires Jupyter Notebook >= 5.0 or Jupyter Lab. Alternatively, you can use TerminalProgressBar().

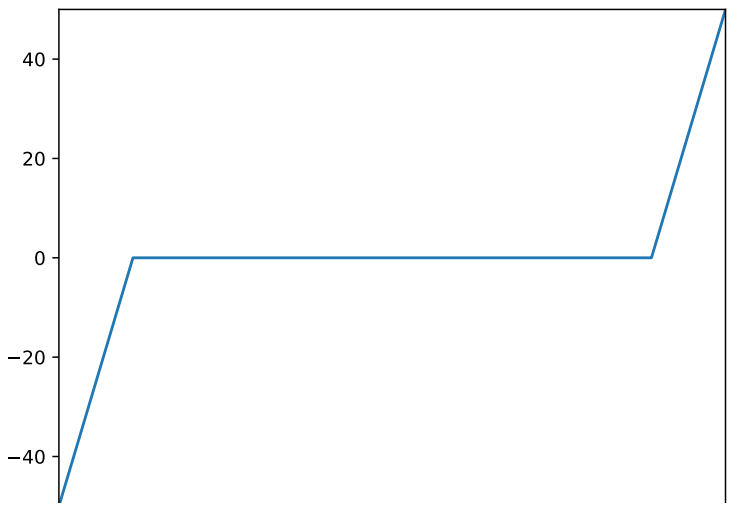
Build finished in 0:00:02.

HMTL progress bar requires Jupyter Notebook >= 5.0 or Jupyter Lab. Alternatively, you can use TerminalProgressBar().

Simulation finished in 0:00:06.







Out[14]: <matplotlib.animation.FuncAnimation at 0x25558739d60>

In [ ]:

0 1 2 3 4 5 6 7 8 9