Forest Fire Simulation
Mathematical Programming with Python

MATH 2604: Advanced Scientific Computing 4
Spring 2025
Monday/Wednesday/Friday, 1:00-1:50pm

https://people.sc.fsu.edu/~jburkardt/classes/python_2025/forest_fire/forest_fire.pdf

---

---

1 A Forest Fire

Let's consider a simple forest fire model.

We imagine a forest of trees laid out on a regular $m \times n$ grid. We imagine the trees have heights of 1, 2, 3 or 4 meters. The height of a tree will determine how long it can burn. To avoid having to worry about the boundaries of the region, we will assume the usual torus or video game or modular arithmetic conditions. So every tree has four neighbors, to the north, south, east and west.

We suppose that the fire is started by a single tree, and we will also assume this tree is tall (that is, has a heighf of 4 meters) so that it will burn for 4 time steps. Now we want to see if the fire spreads.

We will imagine that the probability that an unburnt tree catches fire is related to the number of burning neighbors. If there are 0 such neighbors, the tree will not ignite on this step. If just 1 neighbor is burning, there is a 25% chance that this tree will start to burn, and in general, if s neighbors are burning, there is a $s * 25\%$ chance. In particular, if all 4 neighbors are on fire, this tree must now ignite as well.

To keep track of the status of every tree, we create an array `forest[i,j]`. This array is randomly initialized with integer values between 1 and 4, representing the varying heights of the trees.

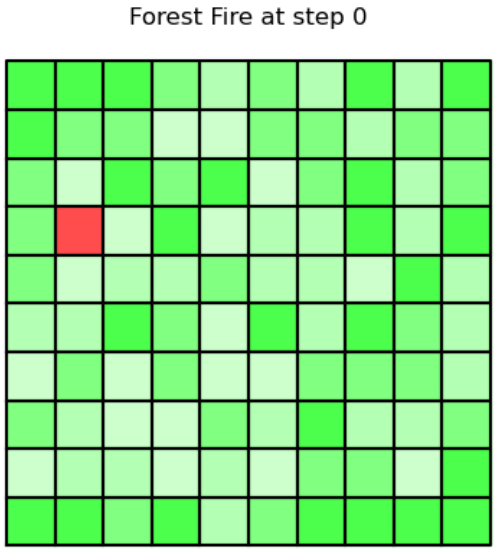
2 An outline of the simulation

To run the simulation, we start by picking at rando the location (i, j) where the fire will start. We set `forest[i,j]=-4`, that is, it's a tall tree (of height 4) and it's burning, because of the negative sign. We force it to be a tall tree so that it has 4 chances to catch its neighbors on fire.

Now we repeatedly advance one time step.

- Every tree that was burning now reduces its height by 1. In particular, the tree that started the fire goes down to -3, then -2, then -1, then 0. A tree that reaches height 0 is, of course, completely gone and no longer plays any role.
- Every tree that has neighbors that were burning now has to risk catching on fire, based on the number of such neighbors.
- Unburnt trees with no burning neighbors are left alone.

Here is a plot suggesting the initial situation in the forest. The darkest green spots are the tallest trees, and the red spot is where the fire starts, namely location [3,1].



3 Updating the status

Here is how we might implement the status of the forest over one step.

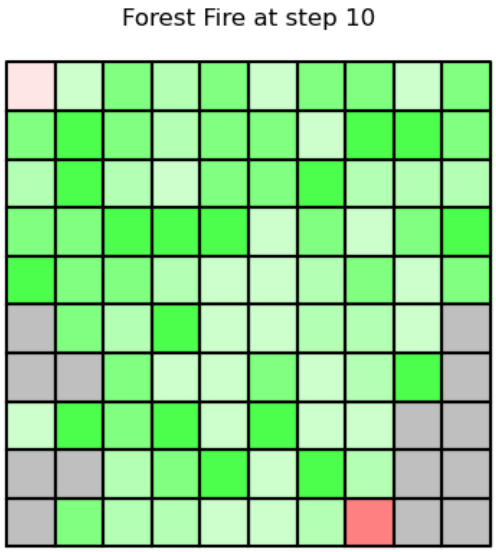
```
def forest_update ( forest ):

  import numpy as np

  m, n = forest.shape

  new_forest = forest.copy ( )

  for i in range ( 0, m ):
    for j in range ( 0, n ):
#
#  A negative value means the tree is burning.
#  Increase it by 1, so it burns towards 0.
#
      if ( forest[i,j] < 0 ):

        new_forest[i,j] = forest[i,j] + 1
#
```

```
#  A positive value means the tree is untouched.
#  Count the burning neighbors to see whether the tree should ignite.
#  This is done by making its index negative.
#
      elif ( 0 < forest[i,j] ):
#
#  Here, we locate the neighbors, but use modular arithmetic
#  so the grid wraps around at the edges.
#
        im1 = ( i - 1 ) % m
        ip1 = ( i + 1 ) % m
        jm1 = ( j - 1 ) % n
        jp1 = ( j + 1 ) % n
#
#  Count the number of neighboring trees that are burning.
#
        s = ( forest[im1,j] < 0 ) + ( forest[ip1,j] < 0 ) \
          + ( forest[i,jm1] < 0 ) + ( forest[i,jp1] < 0 )

        ignite = np.random.random ( )

        if ( ignite < 0.25 * s ):
          new_forest[i,j] = - new_forest[i,j]

  forest = new_forest.copy ( )

  return forest
```

If we keep following the updating rules, then the fire will eventually die out. We can ask questions such as how many trees are likely to survive, and what would happen if the direction of the wind influenced the fire spread.

4 Graphics considerations

Here is what the forest might look like after 10 steps of burning:



It's worth spending some time figuring out how to plot the forest fire.

Notice an interesting problem in geometry, the difference between array coordinates and Cartesian coordinates. We imagine the [0,0] entry of our array to be in the upper left; but if we are thinking in terms of (x, y) Cartesian coordinates, we would be tempted to imagine that tree is in the lower left corner instead. Either convention would work; we will choose to insist that the (0,0) tree is in the upper left, which means that when we do plotting, we will have to be careful to rearrange some data so that the plotting software gets things right!

Our plot can be thought of as an $m \times n$ visual array of boxes. Each box could be filled with a color to suggest the status of the tree there. Greens could be untouched trees, red shades suggest fire, and a gray box would represent a 0 value, where a tree has burnt to the ground. We can use a pair of `for()` loops to identify each box to be drawn. We can use a pair of `fill()` and `plot()` commands to fill a box with color, and then draw the outline of its boundary. For the `fill()` command, we want more choices than the simple one letter color codes. To get what we want, we will instead use the RGB color scheme.

```
    if ( forest[i,j] == 0 ):
      rgb = [0.75,0.75,0.75]      # light gray
    elif ( forest[i,j] == 1 ):
      rgb = [0.8,1.0,0.8]         # light green
    elif ( forest[i,j] == 2 ):
      rgb = [0.7,1.0,0.7]         # medium green
    elif ( forest[i,j] == 3 ):
      rgb = [0.5,1.0,0.5]         # strong green
    elif ( forest[i,j] == 4 ):
      rgb = [0.3,1.0,0.3]         # dark green
    elif ( forest[i,j] == -1 ):
      rgb = [1.0,0.9,0.9]         # light red
    elif ( forest[i,j] == -2 ):
      rgb = [1.0,0.7,0.7]         # medium red
    elif ( forest[i,j] == -3 ):
      rgb = [1.0,0.5,0.5]         # strong red
    elif ( forest[i,j] == -4 ):
      rgb = [1.0,0.3,0.3]         # dark red
```

Once we have chosen the color for our box, we get the coordinates of its corners, fill the interior with color, and draw the outline:

```
    x = [ j, j, j+1, j+1, j ]
    ir = m - 1 - i
    y = [ ir, ir+1, ir+1, ir, ir ]
    plt.fill ( x, y, color = rgb )
    plt.plot ( x, y, 'k-' )
```

After this, we can save the plot to a file, and display it using `plt.show()`.

5 Animation

Now let's consider the possibility of an animation. We already know that the plt.show() command will display the forest status at each time step. However, it would be much more dramatic to be able to watch the individual frames as a movie, without us having to close each frame in order to see the next one.

We may talk later about how to create an animation directly inside a Python program, using a peculiar feature known as `FuncAnimation()`. However, I find that feature awkward to use, and it involves some object-oriented coding I am uncomfortable with. As an alternative, since we have saved every frame as a `png` file, we can string them together as a movie, using one of several external programs.

On my system, I can use the ImageMagick `convert` function to take a collection of still frames and make a GIF animation. On my system, the movie can be created by

```
convert -delay 100 -loop 1 forest_fire*.png forest_fire.gif
```

where the `delay` option specifies the time delay between showing each frame, and the `loop` option specifies how many times the frames should be repeated.

The resulting GIF animation is available from the web page.