Spatial evolutionary game simulator

S. Kolotev, A. Malyutin, E. Burovski, S. Krashakov and L. Shchur, *Dynamic fractals in spatial evolutionary games*, Physica A **499**, 142 (2018)

Thanks to the HSE Academic Fund Grant No 18-05-0024 and the "5-100" program.

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
from matplotlib import animation

%matplotlib notebook
```

In [2]:

```
import sys
import cython

print("python ", sys.version)
print("numpy", np.__version__, ", matplotlib", mpl.__version__, ", and Cython",
cython.__version__)
```

```
python  3.5.2 (default, Nov 17 2016, 17:05:23)
[GCC 5.4.0 20160609]
numpy 1.13.1 , matplotlib 2.0.2 , and Cython 0.27.3
```

Main update function:

In [3]:

```
def evolve2(field, b, num_steps=1):
    L = field.shape[0]
    current = np.zeros((L, L), dtype=int)
    scores = np.zeros((L, L), dtype=float)

    for step in range(num_steps):
        current = field.copy()
        scores = np.zeros((L, L), dtype=float)

        for x in range(L):
            for y in range(L):
                for i in range(-1, 2):
                    for j in range(-1, 2):
                        ix = (x + i) % L
                        jy = (y + j) % L
                        scores[x, y] += (1 - field[ix, jy])

                if field[x, y] == 1:
                    scores[x, y] *= b

        for x in range(L):
            for y in range(L):
                bestX = x
                bestY = y
                for i in range(-1, 2):
                    for j in range(-1, 2):
                        ix = (x + i) % L
                        jy = (y + j) % L
                        if (scores[bestX, bestY] < scores[ix, jy]):
                            bestX = ix
                            bestY = jy

                field[x, y] = current[bestX, bestY]
    return field
```

Run the simulation and animate

In [4]:

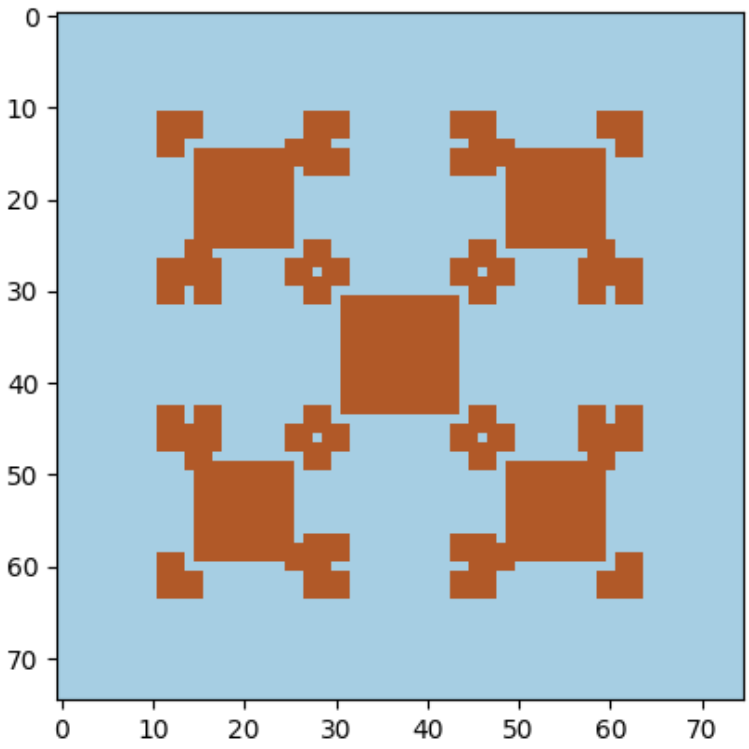
```
L = 75
field = np.zeros((L, L), dtype=int)
field[L//2, L//2] = 1

# draw the initial field
fig = plt.figure()
im = plt.imshow(field, animated=True, cmap=plt.get_cmap('Paired'))

# updater function
def updatefig(*args):
    global field
    field = evolve2(field, 1.81, 1)
    im.set_array(field)
    return im,

# animate!
anim = animation.FuncAnimation(fig, updatefig, frames=200, interval=50, blit=Tru
e)

plt.show()
```



Timings

In [5]:

```
L = 44
field = np.zeros((L, L), dtype=int)
field[L//2, L//2] = 1

%timeit evolve2(field, 1.81, 10)
```

```
1 loop, best of 3: 345 ms per loop
```

Enter Cython

In class, we started with the original code and did the conversion iteratively, removing yellow lines from the annotated output. Below is only the final result.

In [6]:

```
%load_ext cython
```

In [7]:

```
%%cython -a

import numpy as np

import cython

@cython.cdivision(True)
@cython.boundscheck(False)
def evolve2_1(long[:, ::1] field, double b, int num_steps=1):

    cdef int x, y, L, i, j, ix, jy, step

    L = field.shape[0]
    cdef double[:, ::1] scores = np.zeros((L, L), dtype=float)

    cdef double[:, ::1] _zeros = np.zeros((L, L), dtype=float)
    cdef long[:, ::1] current = field.copy()

    for step in range(num_steps):
        current = field.copy()
        scores[...] = _zeros

        for x in range(L):
            for y in range(L):
                for i in range(-1, 2):
                    for j in range(-1, 2):
                        ix = (x + i) % L
                        jy = (y + j) % L
                        scores[x, y] += (1 - field[ix, jy])

                if field[x, y] == 1:
                    scores[x, y] *= b

        for x in range(L):
            for y in range(L):
                bestX = x
                bestY = y
                for i in range(-1, 2):
                    for j in range(-1, 2):
                        ix = (x + i) % L
                        jy = (y + j) % L
                        if (scores[bestX, bestY] < scores[ix, jy]):
                            bestX = ix
                            bestY = jy

                field[x, y] = current[bestX, bestY]
    return field
```

Out[7]:

Generated by Cython 0.25.2

Yellow lines hint at Python interaction.
Click on a line that starts with a "+" to see the C code that Cython generated for it.

```
 01:
+02: import numpy as np
 03:
 04: import cython
 05:
 06: @cython.cdivision(True)
 07: @cython.boundscheck(False)
+08: def evolve2_1(long[:, ::1] field, double b, int num_steps=1):
 09:
 10:     cdef int x, y, L, i, j, ix, jy, step
 11:
+12:     L = field.shape[0]
+13:     cdef double[:, ::1] scores = np.zeros((L, L), dtype=float)
 14:
+15:     cdef double[:, ::1] _zeros = np.zeros((L, L), dtype=float)
+16:     cdef long[:, ::1] current = field.copy()
 17:
+18:     for step in range(num_steps):
+19:         current = field.copy()
+20:         scores[...] = _zeros
 21:
+22:         for x in range(L):
+23:             for y in range(L):
+24:                 for i in range(-1, 2):
+25:                     for j in range(-1, 2):
+26:                         ix = (x + i) % L
+27:                         jy = (y + j) % L
+28:                         scores[x, y] += (1 - field[ix, jy])
 29:
+30:                 if field[x, y] == 1:
+31:                     scores[x, y] *= b
 32:
+33:         for x in range(L):
+34:             for y in range(L):
+35:                 bestX = x
+36:                 bestY = y
+37:                 for i in range(-1, 2):
+38:                     for j in range(-1, 2):
+39:                         ix = (x + i) % L
+40:                         jy = (y + j) % L
+41:                         if (scores[bestX, bestY] < scores[ix,
jy]):
+42:                             bestX = ix
+43:                             bestY = jy
 44:
```

```
+45:                 field[x, y] = current[bestX, bestY]
+46:     return field
```

In [10]:

```
L = 44
field = np.zeros((L, L), dtype=int)
field[L//2, L//2] = 1

%timeit evolve2_1(field, 1.81, 10)
```

```
100 loops, best of 3: 1.98 ms per loop
```

In [11]:

```
345 / 1.98
```

Out[11]:

```
174.24242424242425
```

The recommended code structure is to have a python-facing `def` function which takes care of the input and output, allocated memory etc, and delegates all heavy lifting to an internal `cdef` function.

In [12]:

```
%%cython -a

import numpy as np
import cython

def evolve2_2(field, b, num_steps=1):
    # validate input here
    field = np.atleast_2d(field)
    # XXX check that field is integer dtype etc

    # allocate memory here
    L = field.shape[0]
    scores = np.zeros((L, L), dtype=float)
    current = field.copy()
    _zeros = np.zeros((L, L), dtype=float)

    cdef:
        int _num_steps = num_steps
        double _b = b

    # do the work (`field` array is updated in-place)
    _evolve2_2_impl(field, _b, _num_steps, scores, _zeros, current)

    # convert memoryviews to numpy arrays
    return np.asarray(field)


@cython.cdivision(True)
@cython.boundscheck(False)
cdef void _evolve2_2_impl(long[:, ::1] field, double b, int num_steps,
                                  double[:, ::1] scores, double[:, ::1] _zeros,
                                  long[:, ::1] current) nogil:
```

```
    cdef int x, y, L, i, j, ix, jy, step

    L = field.shape[0]

    for step in range(num_steps):
        current[...] = field
        scores[...] = _zeros

        for x in range(L):
            for y in range(L):
                for i in range(-1, 2):
                    for j in range(-1, 2):
                        ix = (x + i) % L
                        jy = (y + j) % L
                        scores[x, y] += (1 - field[ix, jy])

                if field[x, y] == 1:
                    scores[x, y] *= b

        for x in range(L):
            for y in range(L):
                bestX = x
                bestY = y
                for i in range(-1, 2):
                    for j in range(-1, 2):
                        ix = (x + i) % L
                        jy = (y + j) % L
                        if (scores[bestX, bestY] < scores[ix, jy]):
                            bestX = ix
                            bestY = jy

                field[x, y] = current[bestX, bestY]
```

Out[12]:

Generated by Cython 0.25.2

Yellow lines hint at Python interaction.
Click on a line that starts with a "+" to see the C code that Cython generated for it.

```
 01:
+02: import numpy as np
 03: import cython
 04:
+05: def evolve2_2(field, b, num_steps=1):
 06:     # validate input here
+07:     field = np.atleast_2d(field)
 08:     # XXX check that field is integer dtype etc
 09:
 10:     # allocate memory here
+11:     L = field.shape[0]
+12:     scores = np.zeros((L, L), dtype=float)
+13:     current = field.copy()
+14:     _zeros = np.zeros((L, L), dtype=float)
 15:
 16:     cdef:
+17:         int _num_steps = num_steps
+18:         double _b = b
 19:
 20:     # do the work (`field` array is updated in-place)
+21:     _evolve2_2_impl(field, _b, _num_steps, scores, _zeros, curren
t)
 22:
 23:     # convert memoryviews to numpy arrays
+24:     return np.asarray(field)
 25:
 26:
 27: @cython.cdivision(True)
 28: @cython.boundscheck(False)
+29: cdef void _evolve2_2_impl(long[:, ::1] field, double b, int num_s
teps,
 30:                                   double[:, ::1] scores,
double[:, ::1] _zeros,
 31:                                   long[:, ::1] current) nogil:
 32:
 33:     cdef int x, y, L, i, j, ix, jy, step
 34:
+35:     L = field.shape[0]
 36:
+37:     for step in range(num_steps):
+38:         current[...] = field
+39:         scores[...] = _zeros
 40:
+41:         for x in range(L):
+42:             for y in range(L):
```

```
+43:                 for i in range(-1, 2):
+44:                     for j in range(-1, 2):
+45:                         ix = (x + i) % L
+46:                         jy = (y + j) % L
+47:                         scores[x, y] += (1 - field[ix, jy])
 48:
+49:                 if field[x, y] == 1:
+50:                     scores[x, y] *= b
 51:
+52:         for x in range(L):
+53:             for y in range(L):
+54:                 bestX = x
+55:                 bestY = y
+56:                 for i in range(-1, 2):
+57:                     for j in range(-1, 2):
+58:                         ix = (x + i) % L
+59:                         jy = (y + j) % L
+60:                         if (scores[bestX, bestY] < scores[ix,
jy]):
+61:                             bestX = ix
+62:                             bestY = jy
 63:
+64:                 field[x, y] = current[bestX, bestY]
```

In [13]:

```
L = 44
field = np.zeros((L, L), dtype=int)
field[L//2, L//2] = 1

%timeit evolve2_2(field, 1.81, 10)
```

```
1000 loops, best of 3: 1.8 ms per loop
```

Now, for something completely different: numba just-in-time compilation

In [14]:

```
import numba
```

In [15]:

```
def evolve3(field, b, num_steps=1):
    L = field.shape[0]
    _zeros = np.zeros((L, L), dtype=float)
    _int_zeros = np.zeros((L, L), dtype=int)
    current = _int_zeros.copy()
    scores = _zeros.copy()
    evolve3_impl(field, b, num_steps, _zeros, _int_zeros)
    return field


# Note a single decorator: @jit


@numba.jit(nopython=True)
def evolve3_impl(field, b, num_steps, _zeros, _int_zeros):
    L = field.shape[0]
    current = _int_zeros.copy()

    for step in range(num_steps):
        current = field.copy()
        scores = _zeros.copy()

        for x in range(L):
            for y in range(L):
                for i in range(-1, 2):
                    for j in range(-1, 2):
                        ix = (x + i) % L
                        jy = (y + j) % L
                        scores[x, y] += (1 - field[ix, jy])

                if field[x, y] == 1:
                    scores[x, y] *= b

        for x in range(L):
            for y in range(L):
                bestX = x
                bestY = y
                for i in range(-1, 2):
                    for j in range(-1, 2):
                        ix = (x + i) % L
                        jy = (y + j) % L
                        if (scores[bestX, bestY] < scores[ix, jy]):
                            bestX = ix
                            bestY = jy

                field[x, y] = current[bestX, bestY]
    return field
```

In [16]:

```
L = 44
field = np.zeros((L, L), dtype=int)
field[L//2, L//2] = 1

%timeit evolve3(field, 1.81, 10)
```

```
The slowest run took 252.75 times longer than the fastest. This could
 mean that an intermediate result is being cached.
1 loop, best of 3: 2.14 ms per loop
```

In [ ]:

```

```