NeuralField

January 29, 2020

[ ]:
```
import matplotlib.pyplot as plt
import plotly.graph_objects as go
import numpy as np
%matplotlib inline
```

[ ]:
```
class NeuralField:

    def __init__(self, tau, a, b, d, k, c=None, v=None):

        self.tau = tau
        self.a = a
        self.b = b
        self.d = d
        self.k = k

        if c and v:
            self.c = c
            self.v = v

    def kernel(self, x):
        '''Interaction kernel, Gabor filter'''
        return self.a * (np.exp(-(x**2)/(4*self.b**2))*np.cos(self.k*x))/(self.
↪b*np.sqrt(np.pi))

    def kernel2(self, x):
        return np.exp(-self.c*abs(x))*np.sign(x)

    def current(self, f, A, dx, add_noise=False):
        '''Input stimulus'''
        r = np.zeros(len(f))
        for k in range(len(f)):
            xk = A + k*dx
            r[k] = np.exp(-(xk**2)/(4*self.d**2))/(2*self.d*np.sqrt(np.pi))
        if add_noise:
            mu, sigma = 0, 0.01
            noise = np.random.normal(mu, sigma, len(f))
            r += noise
```

```
        return r

    def current2(self, f, A, dx, t, add_noise=False):
        r = np.zeros(len(f))
        for k in range(len(f)):
            xk = A + k*dx
            r[k] = self.c * np.exp(-((xk - self.v*t)**2)/(4*self.d**2))/(2*self.
↪d*np.sqrt(np.pi))
        if add_noise:
            mu, sigma = 0, 0.01
            noise = np.random.normal(mu, sigma, len(f))
            r += noise
        return r

    def convolve(self, f, A, dx):
        '''Linear 1D convolution'''
        r = np.zeros(len(f))
        for k in range(len(f)):
            xk = A + k*dx
            for i in range(len(f)):
                xi = A + i*dx
                r[k] += self.kernel(xk - xi) * f[i] * dx
        return r

    def simulate(self, A, B, dt, N):
        '''Simulate the Neural Field'''
        t = np.arange(0, 40, dt)
        dx = (B-A)/N
        du = np.zeros((len(t), N))

        for i in range(1, len(t)):
            this_convolution = self.convolve(du[(i-1), :], A, dx)
            this_current     = self.current(du[(i-1), :], A, dx,␣
↪add_noise=False)
            du[i, :] = du[(i-1), :] + (dt/self.tau)*(-du[(i-1), :] +␣
↪this_convolution + this_current)
        return du

class NonlinearNeuralField:

    def __init__(self, tau, a, b, d, h, A, B, C):

        self.tau = tau
        self.a = a
        self.b = b
        self.d = d
        self.h = h
```

```
        self.A = A
        self.B = B
        self.C = C

    def kernel(self, x):

        if abs(x) <= self.a:
            return self.A
        elif (abs(x) >= self.a) and (abs(x) <= self.b):
            return -self.B
        return 0

    def kernel2(self, x):
        if abs(x) <= self.b:
            return 0.8*x
        else:
            return 0

    def current(self, f, A, dx):
        '''Input stimulus'''
        r = np.zeros(len(f))
        for k in range(len(f)):
            xk = A + k*dx
            if abs(xk) <= self.d:
                r[k] = self.C*(1-(abs(xk)/self.d))
            else:
                r[k] = 0
        return r

    def convolve(self, f, A, dx):
        '''Linear 1D convolution'''
        r = np.zeros(len(f))
        for k in range(len(f)):
            xk = A + k*dx
            for i in range(len(f)):
                xi = A + i*dx

                r[k] += self.kernel(xk - xi) * (f[i]>0) * dx
        return r

    def simulate(self, A, B, dt, N):
        '''Simulate the Neural Field'''
        t = np.arange(A, B, dt)
        dx = (B-A)/N
        du = np.zeros((len(t), N))
        to_start = - self.h
        du[0, :] = to_start
```

```
        for i in range(1, len(t)):
            this_convolution = self.convolve(du[(i-1), :], A, dx)
            this_current     = self.current(du[(i-1), :], A, dx)
            du[i, :] = du[(i-1), :] + (dt/self.tau)*(-du[(i-1), :] +␣
↪this_convolution + this_current - self.h)
        return du
```

[ ]:
```
tau = 10
a = 1
b = 0.6
d = 2
k = 4
c = 1
v = 0.1
f = NeuralField(tau, a, b, d, k, c, v)
```

[ ]:
```
A = -10
B = 10
dt = 0.1
N = 200
du = f.simulate(A, B, dt, N)
```

[ ]:
```
vals = np.arange(0, 200, 20)
ticks = np.arange(-10, 10, 2)

layout = go.Layout(scene = dict(
                    xaxis = dict(
                        title = 'Space',
                        tickmode = 'array',
                        tickvals = vals,
                        ticktext = ticks),
                    yaxis_title='Time / ms',
                    zaxis_title='Firing rate'),
                    width=700,
                    margin=dict(r=20, b=10, l=10, t=10))

fig = go.Figure(data=go.Surface(z=du, contours = {
        "x": {"show": True, "start": 0, "end": 200, "size": 2, "color":
↪"black"}}), layout=layout)
fig.show()
```

[ ]:
```
tau = 10
a = 1
b = 3
d = 4
h = 1
```

```
A = 3
B = 2
C = 0.6
n = NonlinearNeuralField(tau, a, b, d, h, A, B, C)
```

[ ]:
```
A = -10
B = 10
dt = 0.1
N = 200
du = n.simulate(A, B, dt, N)
```

[ ]:
```
vals = np.arange(0, 200, 20)
ticks = np.arange(-10, 10, 2)

layout = go.Layout(scene = dict(
                    xaxis = dict(
                        title = 'Space',
                        tickmode = 'array',
                        tickvals = vals,
                        ticktext = ticks),
                    yaxis_title='Time / ms',
                    zaxis_title='Firing rate'),
                    width=700,
                    margin=dict(r=20, b=10, l=10, t=10))

fig = go.Figure(data=go.Surface(z=du, contours = {
        "x": {"show": True, "start": 0, "end": 200, "size": 2, "color":
 ↪"black"}}), layout=layout)
fig.show()
```

[ ]:
```

```