Problem1

March 23, 2021

1 Problem 1

[1]:
```
#Import libraries for simulation
import tensorflow.compat.v1 as tf
import numpy as np
import time
tf.disable_eager_execution()
#Imports for visualization
import PIL.Image
from io import BytesIO
from IPython.display import clear_output, Image, display


def DisplayArray(a, fmt='jpeg', rng=[0,1]):
    """Display an array as a picture."""
    a = (a - rng[0])/float(rng[1] - rng[0])*255
    a = np.uint8(np.clip(a, 0, 255))
    f = BytesIO()
    PIL.Image.fromarray(a).save(f, fmt)
    clear_output(wait = True)
    display(Image(data=f.getvalue()))
```

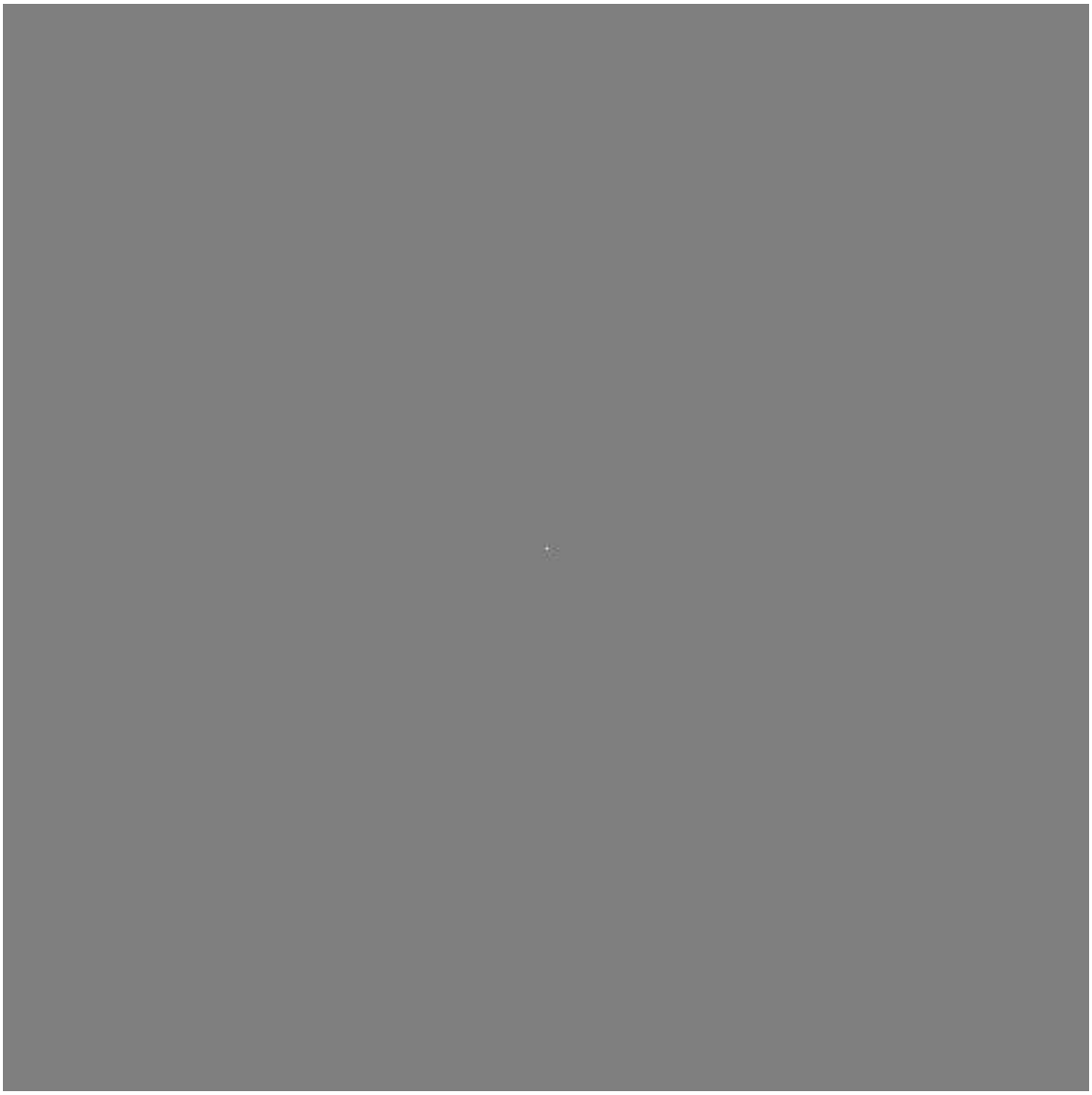
[2]:
```
def make_kernel(a):
    """Transform a 2D array into a convolution kernel"""
    a = np.asarray(a)
    a = a.reshape(list(a.shape) + [1,1])
    return tf.constant(a, dtype=1)

def simple_conv(x, k):
    """A simplified 2D convolution operation"""
    x = tf.expand_dims(tf.expand_dims(x, 0), -1)
    y = tf.nn.depthwise_conv2d(x, k, [1, 1, 1, 1], padding='SAME')
    return y[0, :, :, 0]
```

[3]:
```
def laplace_iso(x):
    """Compute the 2D laplacian of an array"""
    laplace_k = make_kernel([[0.25, 0.5, 0.25],
```

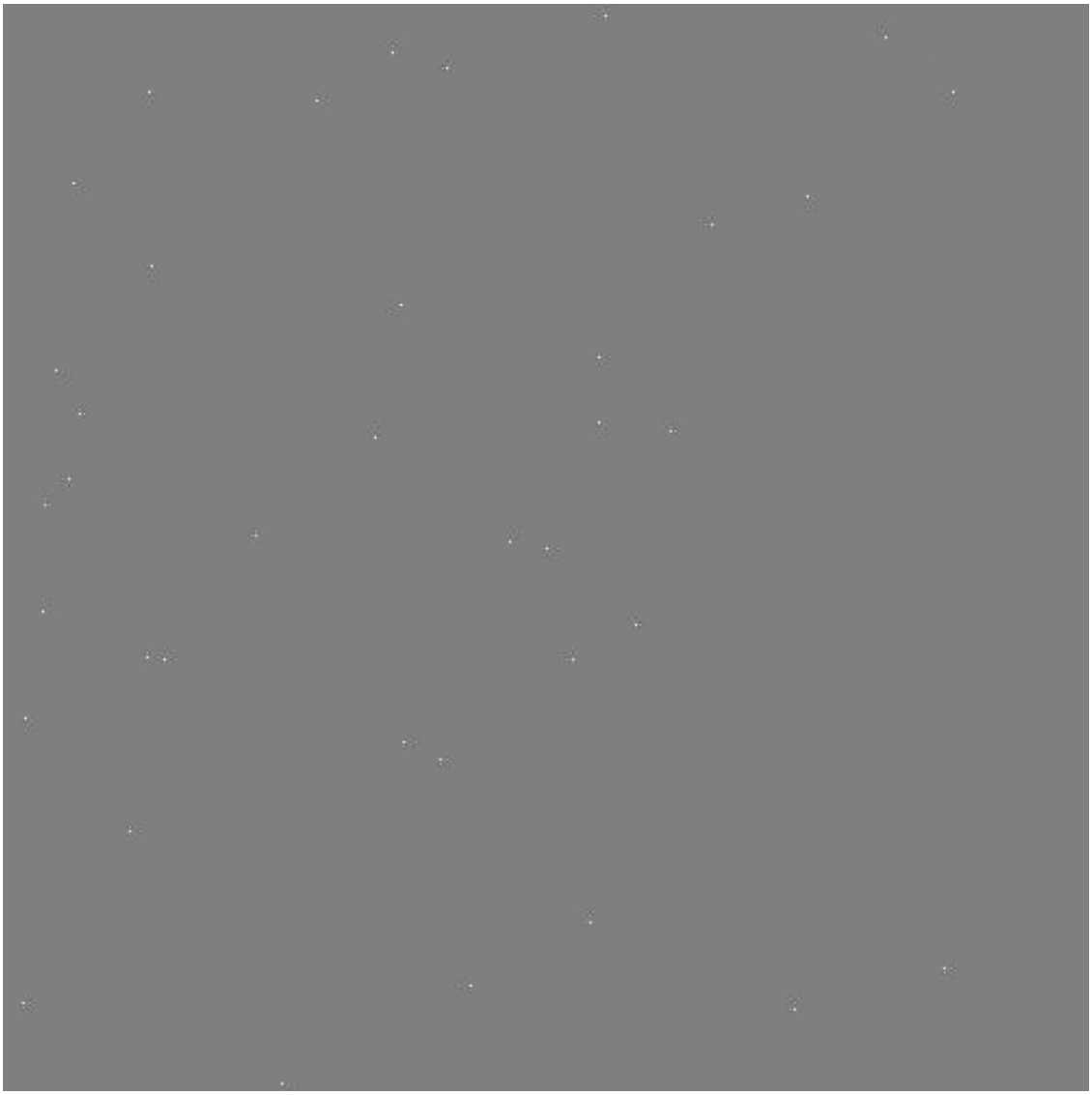
```
                            [0.5, -3., 0.5],
                            [0.25, 0.5, 0.25]])
    return simple_conv(x, laplace_k)
def laplace(x):
    """Compute the 2D laplacian of an array"""
    laplace_k = make_kernel([[0., 1., 0.],
                            [1., -4., 1.],
                            [0., 1., 0.]])
    return simple_conv(x, laplace_k)
```

2 N = 500 Steps

[4]:
```
N = 500

# Set everything to zero
u_init = np.zeros([N, N], dtype=np.float32)
ut_init = np.zeros([N, N], dtype=np.float32)

u_init[N//2,N//2] = 10.

DisplayArray(u_init, rng=[-0.1, 0.1])
```



[5]:
```
# more fun initial condition
for n in range(40):
    a,b = np.random.randint(0, N, 2)
    u_init[a,b] = np.random.uniform()

DisplayArray(u_init, rng=[-0.1, 0.1])
```

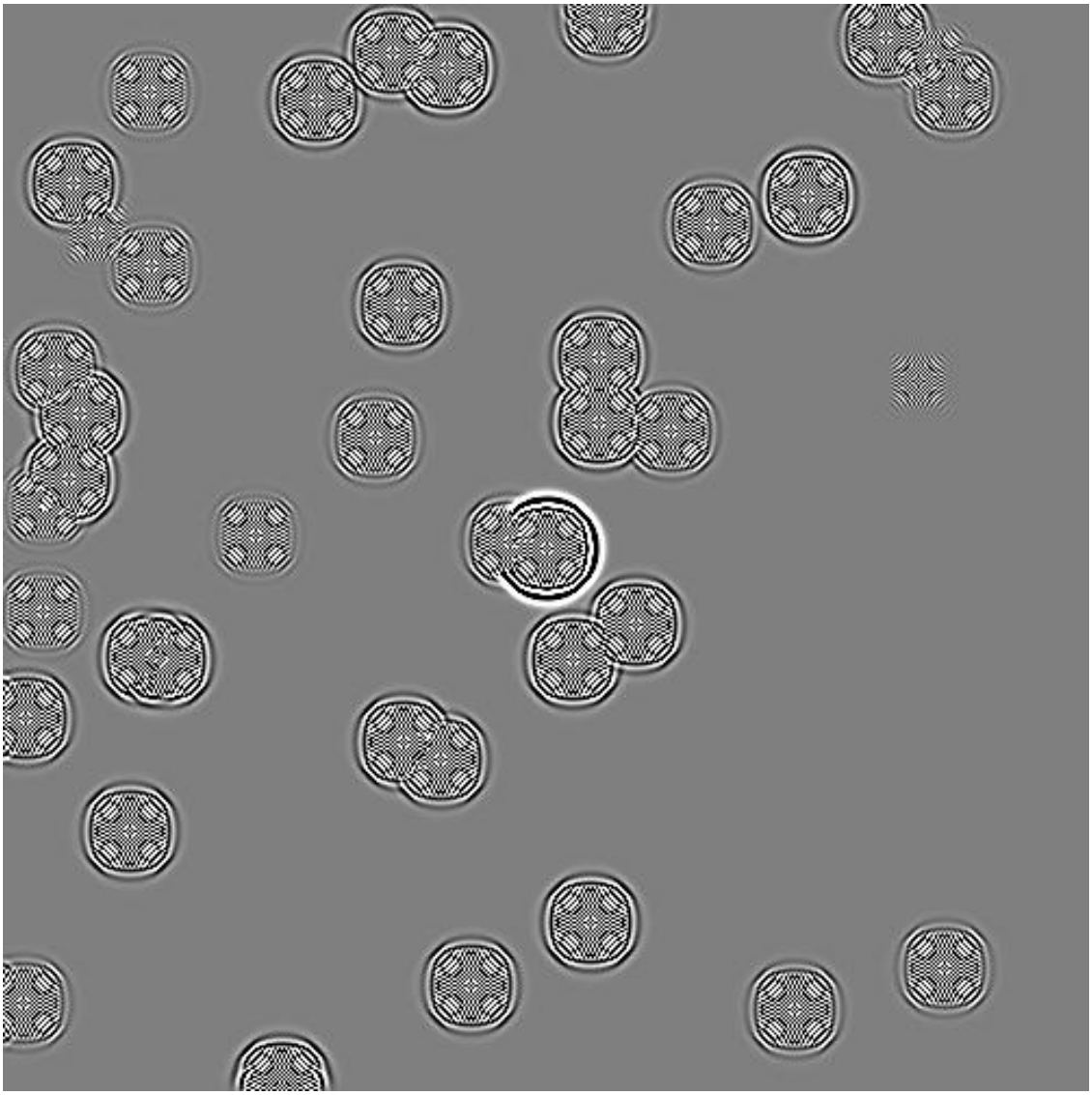


[6]:
```
sess = tf.InteractiveSession()
```

[7]:
```
# Parameters:
# eps -- time resolution
# damping -- wave damping
# c -- wave speed
eps = tf.placeholder(tf.float32, shape=())
damping = tf.placeholder(tf.float32, shape=())
c = tf.placeholder(tf.float32, shape=())

# Create variables for simulation state
U  = tf.Variable(u_init)
Ut = tf.Variable(ut_init)
```

```
# Discretized PDE update rules
U_ = U + eps * Ut
Ut_ = Ut + eps * ((c ** 2) * laplace(U) - damping * Ut)

# Operation to update the state
step = tf.group(
    U.assign(U_),
    Ut.assign(Ut_))

# Initialize state to initial conditions
tf.global_variables_initializer().run()
```

[8]:
```
# Run 1000 steps of PDE
toc = time.time()
for i in range(300):
    # Step simulation
    step.run({eps: 0.03, damping: 0.04, c: 3.0})
    DisplayArray(U.eval(), rng=[-0.1, 0.1])
tic = time.time()

sess.close()

TotalTime = tic-toc
print("Time for N = " + str(N) + " steps is " + str(TotalTime))
```



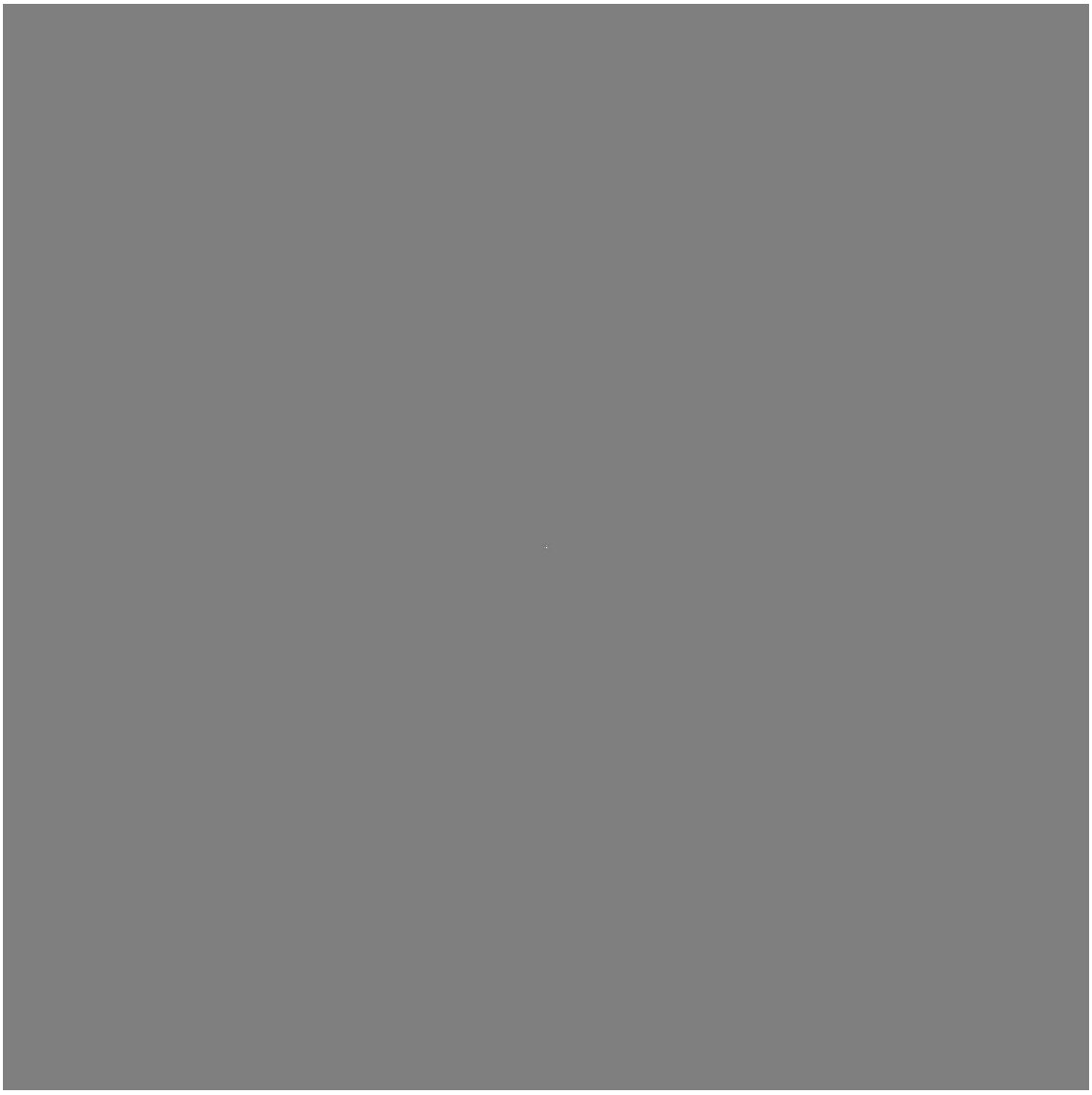
```
Time for N = 500 steps is 22.265596866607666
```

[9]:
```
sess.close()
```

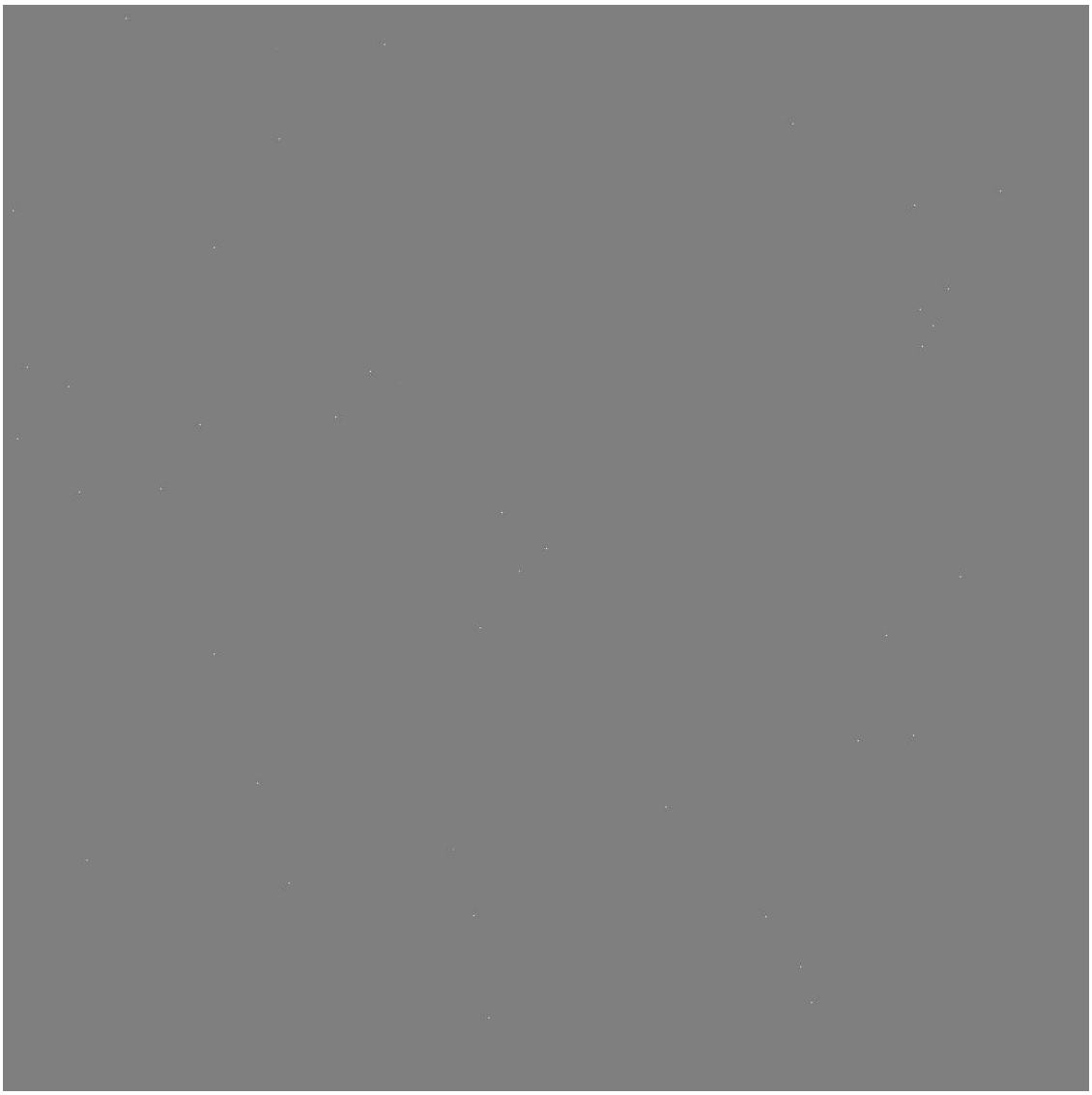
3 N = 1000 Steps

[10]:
```
N = 1000

# Set everything to zero
u_init = np.zeros([N, N], dtype=np.float32)
ut_init = np.zeros([N, N], dtype=np.float32)
```

```
u_init[N//2,N//2] = 10.

DisplayArray(u_init, rng=[-0.1, 0.1])
```



[11]:
```
# more fun initial condition
for n in range(40):
    a,b = np.random.randint(0, N, 2)
    u_init[a,b] = np.random.uniform()

DisplayArray(u_init, rng=[-0.1, 0.1])
```

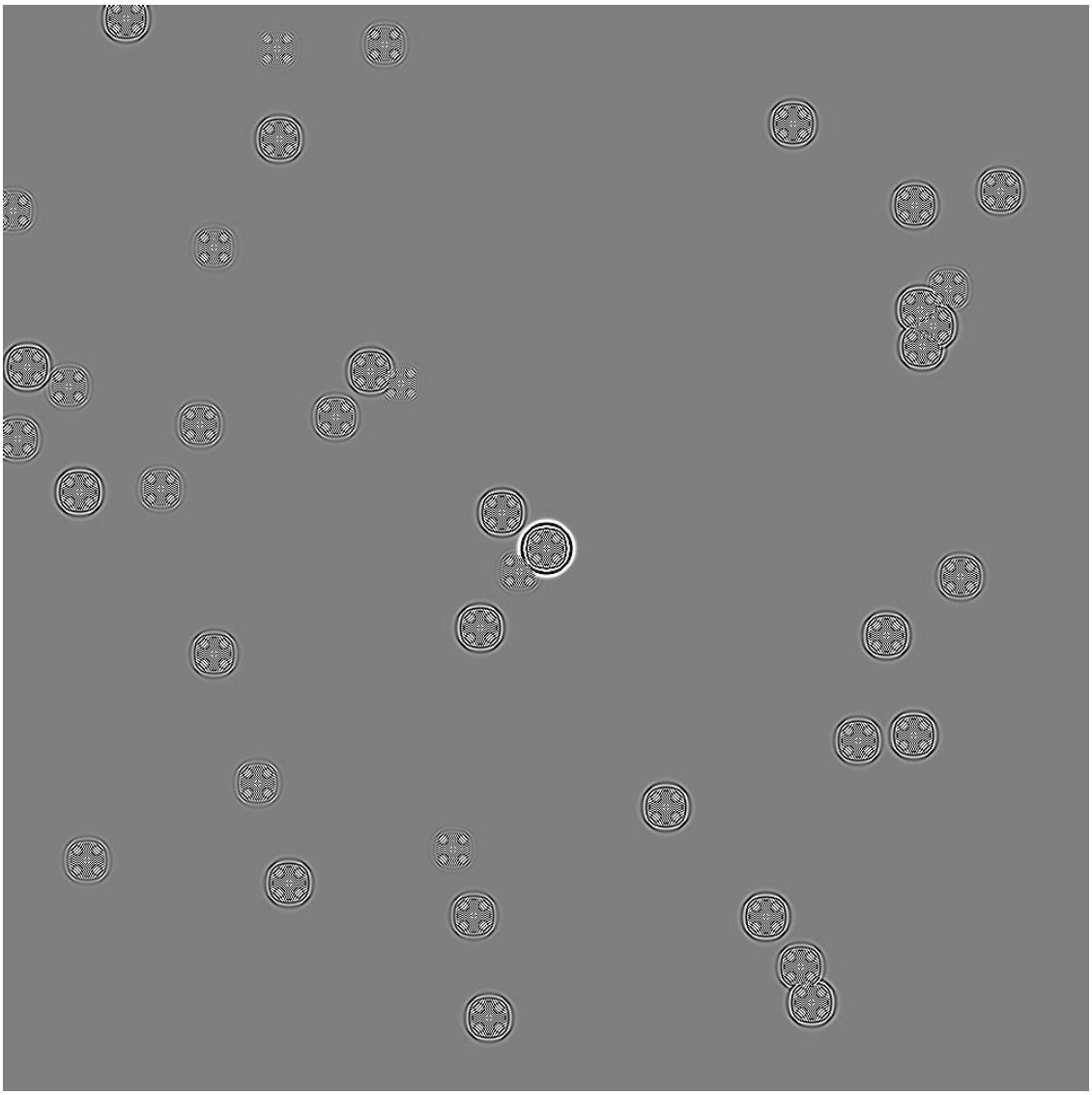


[12]:
```
sess = tf.InteractiveSession()
```

[13]:
```
# Parameters:
# eps -- time resolution
# damping -- wave damping
# c -- wave speed
eps = tf.placeholder(tf.float32, shape=())
damping = tf.placeholder(tf.float32, shape=())
c = tf.placeholder(tf.float32, shape=())

# Create variables for simulation state
U  = tf.Variable(u_init)
Ut = tf.Variable(ut_init)
```

```
# Discretized PDE update rules
U_ = U + eps * Ut
Ut_ = Ut + eps * ((c ** 2) * laplace(U) - damping * Ut)

# Operation to update the state
step = tf.group(
    U.assign(U_),
    Ut.assign(Ut_))

# Initialize state to initial conditions
tf.global_variables_initializer().run()
```

[14]:
```
# Run 1000 steps of PDE
toc = time.time()
for i in range(300):
    # Step simulation
    step.run({eps: 0.03, damping: 0.04, c: 3.0})
    DisplayArray(U.eval(), rng=[-0.1, 0.1])
tic = time.time()

sess.close()

TotalTime = tic-toc
print("Time for N = " + str(N) + " steps is " + str(TotalTime))
```



```
Time for N = 1000 steps is 60.673757791519165
```

[15]:
```
sess.close()
```