In [3]:

```
import torch
from torch import nn
import numpy as np
from torch.autograd import Variable
import matplotlib.pyplot as plt

from Forward_with_Layer_Setting import Net
from InitialConditionLoss import InitialCondition_rho

##########################################

time_plotted = np.array([.1, .2, .3, .4, .5, .6, .7, .8, .9, 1])

##########################################

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

net = Net().to(device)

net.load_state_dict(torch.load("ES_NSloss5th_lr0.005_t1.0_8_11_18h.pt", map_location=torch.device('cpu')))
'''
Iteration: 6500         Total Loss: tensor(9.9080)
IC Loss:  tensor(0.0050)        BC Loss:  tensor(1.5626e-06)    NS PDE Loss:  tensor(2.4086e-05)        NS
  *Saved ; Early Stopping for the latest NS PDE Loss of 5th decimal place

  *Saved ; Early Stopping for the latest IC Loss of 3rd decimal place

  *Saved ; Early Stopping for the latest IC/NS Loss of 3rd decimal place and SurfaceTension Loss of 4th dec

  *(Regularize Losses)Small NS loss and big IC loss ->> IC Coefficients are changed; mse_BC + 100 * mse_MvI
'''
#Graph at various time slices

spatial_discretization = 1000

#Define numpy arrays for inputs
x1 = np.linspace(net.x1_l,net.x1_u,spatial_discretization).reshape(spatial_discretization)
x2 = np.linspace(net.x2_l,net.x2_u,spatial_discretization).reshape(spatial_discretization)
x1x2 = np.array(np.meshgrid(x1, x2)).reshape(2,spatial_discretization**2)
```

```
t = time_plotted[0]*np.ones((spatial_discretization**2,1))

x1_input = x1x2[0].reshape(spatial_discretization**2, 1)
x2_input = x1x2[1].reshape(spatial_discretization**2, 1)

x1x2 = [x1_input, x2_input]

#convert to pytorch tensors
pt_x1 = Variable(torch.from_numpy(x1_input).float(), requires_grad=False).to(device)
pt_x2 = Variable(torch.from_numpy(x2_input).float(), requires_grad=False).to(device)
pt_t = Variable(torch.from_numpy(t).float(), requires_grad=False).to(device)

#get network outputs
pt_u1, pt_u2, pt_p, pt_rho = net(pt_x1, pt_x2, pt_t)

#Convert back to numpy
u1, u2, p, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.

#get actual initial condition
rho_exact = InitialCondition_rho(net, pt_x1, pt_x2)




X, Y = np.meshgrid(x1, x2)



fig, axs = plt.subplots(3,4, figsize=(10,5))
#fig.suptitle(f'Time = {time_plotted}')
fig.tight_layout()
axs[0,0].set_title("Predicted density shape at t = %.1f" % time_plotted[0])
axs[0,1].set_title("Predicted density shape at t = %.1f" % time_plotted[1])
axs[0,2].set_title("Predicted density shape at t = %.1f" % time_plotted[2])
axs[0,3].set_title("Predicted density shape at t = %.1f" % time_plotted[3])
axs[1,0].set_title("Predicted density shape at t = %.1f" % time_plotted[4])
axs[1,1].set_title("Predicted density shape at t = %.1f" % time_plotted[5])
axs[1,2].set_title("Predicted density shape at t = %.1f" % time_plotted[6])
axs[1,3].set_title("Predicted density shape at t = %.1f" % time_plotted[7])
axs[2,0].set_title("Predicted density shape at t = %.1f" % time_plotted[8])
axs[2,1].set_title("Predicted density shape at t = %.1f" % time_plotted[9])
```

```
axs[2,2].set_title('Actual Initial Density where t=0 for comparing movement')
axs[2,3].set_title('Difference of Density Between t=0 and t=1')


# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[0,0].imshow(rho.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                              interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[0,0].invert_yaxis(), extend='both')
cbar.minorticks_on()
##############################################################################################

#Next final_time

t = time_plotted[1]*np.ones((spatial_discretization**2,1))
#convert to pytorch tensors
pt_t = Variable(torch.from_numpy(t).float(), requires_grad=False).to(device)
#get network outputs
_, _, _, pt_rho = net(pt_x1, pt_x2, pt_t)

#Convert back to numpy
_, _, _, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.cpu


# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[0,1].imshow(rho.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                              interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[0,1].invert_yaxis(), extend='both')
cbar.minorticks_on()
##############################################################################################

#Next final_time

t = time_plotted[2]*np.ones((spatial_discretization**2,1))
#convert to pytorch tensors
pt_t = Variable(torch.from_numpy(t).float(), requires_grad=False).to(device)
#get network outputs
_, _, _, pt_rho = net(pt_x1, pt_x2, pt_t)
```

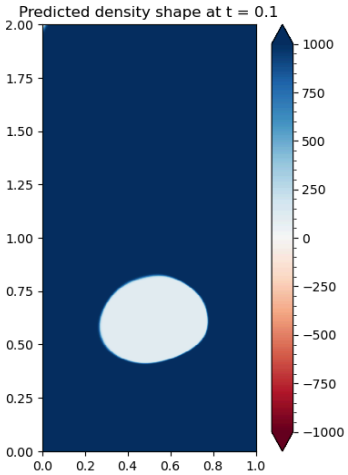
```
#Convert back to numpy
_, _, _, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.cpu


# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[0,2].imshow(rho.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                              interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[0,2].invert_yaxis(), extend='both')
cbar.minorticks_on()
###############################################################################################


#Next final_time

t = time_plotted[3]*np.ones((spatial_discretization**2,1))
#convert to pytorch tensors
pt_t = Variable(torch.from_numpy(t).float(), requires_grad=False).to(device)
#get network outputs
_, _, _, pt_rho = net(pt_x1, pt_x2, pt_t)

#Convert back to numpy
_, _, _, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.cpu


# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[0,3].imshow(rho.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                              interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[0,3].invert_yaxis(), extend='both')
cbar.minorticks_on()
###############################################################################################


#Next final_time

t = time_plotted[4]*np.ones((spatial_discretization**2,1))
#convert to pytorch tensors
pt_t = Variable(torch.from_numpy(t).float(), requires_grad=False).to(device)
```

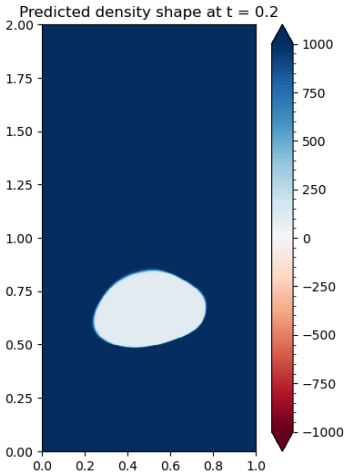
```
#get network outputs
_, _, _, pt_rho = net(pt_x1, pt_x2, pt_t)

#Convert back to numpy
_, _, _, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.cpu


# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[1, 0].imshow(rho.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                              interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[1,0].invert_yaxis(), extend='both')
cbar.minorticks_on()
#########################################################################################


#Next final_time

t = time_plotted[5]*np.ones((spatial_discretization**2,1))
#convert to pytorch tensors
pt_t = Variable(torch.from_numpy(t).float(), requires_grad=False).to(device)
#get network outputs
_, _, _, pt_rho = net(pt_x1, pt_x2, pt_t)

#Convert back to numpy
_, _, _, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.cpu


# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[1,1].imshow(rho.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                              interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[1,1].invert_yaxis(), extend='both')
cbar.minorticks_on()
#########################################################################################


#Next final_time

t = time_plotted[6]*np.ones((spatial_discretization**2,1))
```

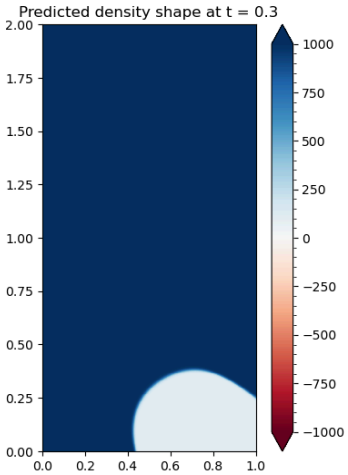
```
#convert to pytorch tensors
pt_t = Variable(torch.from_numpy(t).float(), requires_grad=False).to(device)
#get network outputs
_, _, _, pt_rho = net(pt_x1, pt_x2, pt_t)

#Convert back to numpy
_, _, _, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.cpu


# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[1,2].imshow(rho.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                                  interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[1,2].invert_yaxis(), extend='both')
cbar.minorticks_on()
#########################################################################################################


#Next final_time

t = time_plotted[7]*np.ones((spatial_discretization**2,1))
#convert to pytorch tensors
pt_t = Variable(torch.from_numpy(t).float(), requires_grad=False).to(device)
#get network outputs
_, _, _, pt_rho = net(pt_x1, pt_x2, pt_t)

#Convert back to numpy
_, _, _, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.cpu


# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[1,3].imshow(rho.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                                  interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[1,3].invert_yaxis(), extend='both')
cbar.minorticks_on()
#########################################################################################################


#Next final_time
```

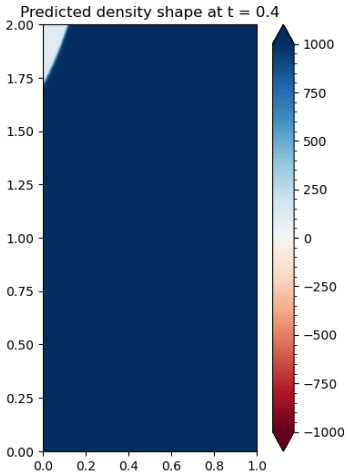
```
t = time_plotted[8]*np.ones((spatial_discretization**2,1))
#convert to pytorch tensors
pt_t = Variable(torch.from_numpy(t).float(), requires_grad=False).to(device)
#get network outputs
_, _, _, pt_rho = net(pt_x1, pt_x2, pt_t)

#Convert back to numpy
_, _, _, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.cpu


# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[2,0].imshow(rho.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                              interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[2,0].invert_yaxis(), extend='both')
cbar.minorticks_on()
#########################################################################################################


#Next final_time

t = time_plotted[9]*np.ones((spatial_discretization**2,1))
#convert to pytorch tensors
pt_t = Variable(torch.from_numpy(t).float(), requires_grad=False).to(device)
#get network outputs
_, _, _, pt_rho = net(pt_x1, pt_x2, pt_t)

#Convert back to numpy
_, _, _, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.cpu


# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[2,1].imshow(rho.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                              interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[2,1].invert_yaxis(), extend='both')
cbar.minorticks_on()

#########################################################################################################
```

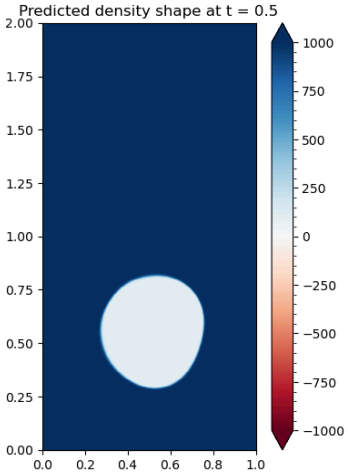
```
#Graph of IC at t=0

#Convert back to numpy
u1, u2, p, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.c
rho_exact = rho_exact.data.cpu().numpy()
error = rho-rho_exact

# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[2,2].imshow(rho_exact.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                                  interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[2,2].invert_yaxis(), extend='both')
cbar.minorticks_on()
##############################################################################################

#Comparing t=0 and t=1



# Plot both positive and negative values between +/- 1000
pos_neg_clipped = axs[2,3].imshow(error.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                                  interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[2,3].invert_yaxis(), extend='both')
cbar.minorticks_on()

#plt.xticks(np.arange(0, 1, step=.1))
#plt.yticks(np.arange(0, 2, step=.1))

#axs[2].legend()
fig.set_figheight(20)
fig.set_figwidth(20)

plt.show()

#Printing on Wide Screen
from IPython.display import display, HTML
display(HTML("<style>.container { width:100% !important; }</style>"))
```

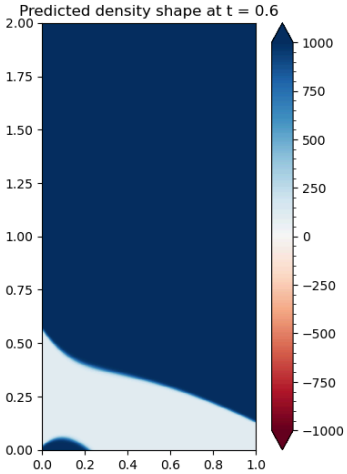

Predicted density shape at t = 0.1

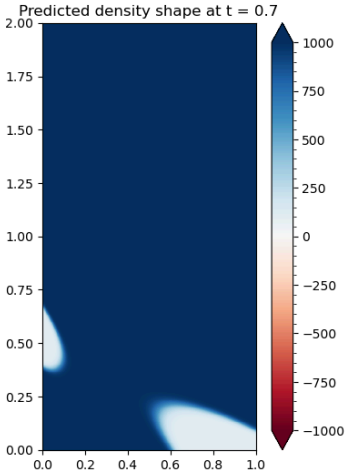

Predicted density shape at t = 0.2

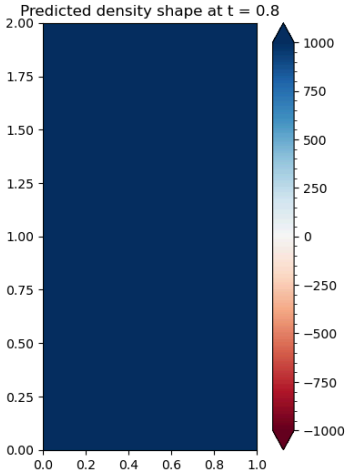

Predicted density shape at t = 0.3

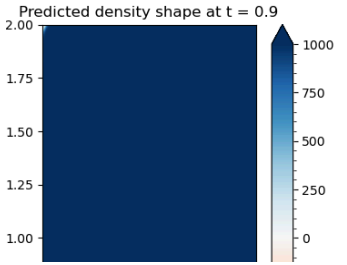

Predicted density shape at t = 0.4

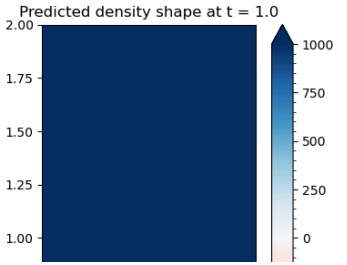

Predicted density shape at t = 0.5

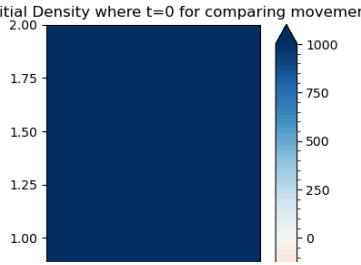

Predicted density shape at t = 0.6

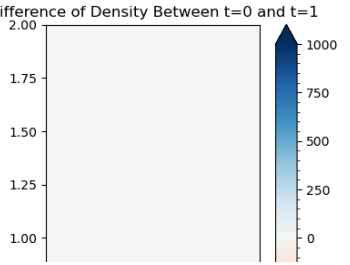

Predicted density shape at t = 0.7

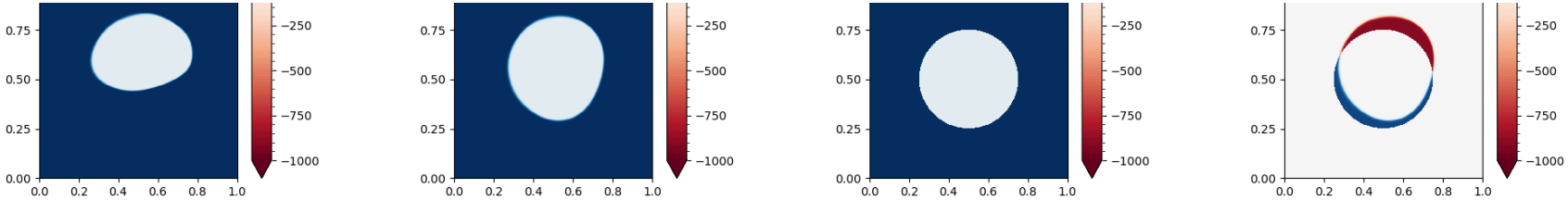

Predicted density shape at t = 0.8


Predicted density shape at t = 0.9


Predicted density shape at t = 1.0


Actual Initial Density where t=0 for comparing movement


Difference of Density Between t=0 and t=1



In [4]:

```
import torch
from torch import nn
import numpy as np
from torch.autograd import Variable
import matplotlib.pyplot as plt

from Forward_with_Layer_Setting import Net
from InitialConditionLoss import InitialCondition_rho

########################################

time_plotted = np.array([.1, .2, .3, .4, .5, .6, .7, .8, .9, 1])

########################################

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

net = Net().to(device)

net.load_state_dict(torch.load("ES_NSloss6th_lr0.005_t1.0_8_11_18h.pt", map_location=torch.device('cpu')))

'''
Iteration: 7500         Total Loss: tensor(63.5094)
IC Loss:  tensor(0.0064)        BC Loss:  tensor(2.0507e-06)    NS PDE Loss:  tensor(4.8408e-06)        NS
  *Saved ; Early Stopping for the latest NS PDE Loss of 6th decimal place

  *Saved ; Early Stopping for the latest IC Loss of 3rd decimal place

'''

#Graph at various time slices

spatial_discretization = 1000
```

```
#Define numpy arrays for inputs
x1 = np.linspace(net.x1_l,net.x1_u,spatial_discretization).reshape(spatial_discretization)
x2 = np.linspace(net.x2_l,net.x2_u,spatial_discretization).reshape(spatial_discretization)
x1x2 = np.array(np.meshgrid(x1, x2)).reshape(2,spatial_discretization**2)

t = time_plotted[0]*np.ones((spatial_discretization**2,1))

x1_input = x1x2[0].reshape(spatial_discretization**2, 1)
x2_input = x1x2[1].reshape(spatial_discretization**2, 1)

x1x2 = [x1_input, x2_input]

#convert to pytorch tensors
pt_x1 = Variable(torch.from_numpy(x1_input).float(), requires_grad=False).to(device)
pt_x2 = Variable(torch.from_numpy(x2_input).float(), requires_grad=False).to(device)
pt_t = Variable(torch.from_numpy(t).float(), requires_grad=False).to(device)

#get network outputs
pt_u1, pt_u2, pt_p, pt_rho = net(pt_x1, pt_x2, pt_t)

#Convert back to numpy
u1, u2, p, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.

#get actual initial condition
rho_exact = InitialCondition_rho(net, pt_x1, pt_x2)



X, Y = np.meshgrid(x1, x2)



fig, axs = plt.subplots(3,4, figsize=(10,5))
#fig.suptitle(f'Time = {time_plotted}')
fig.tight_layout()
axs[0,0].set_title("Predicted density shape at t = %.1f" % time_plotted[0])
axs[0,1].set_title("Predicted density shape at t = %.1f" % time_plotted[1])
axs[0,2].set_title("Predicted density shape at t = %.1f" % time_plotted[2])
axs[0,3].set_title("Predicted density shape at t = %.1f" % time_plotted[3])
axs[1,0].set_title("Predicted density shape at t = %.1f" % time_plotted[4])
```

```
axs[1,1].set_title("Predicted density shape at t = %.1f" % time_plotted[5])
axs[1,2].set_title("Predicted density shape at t = %.1f" % time_plotted[6])
axs[1,3].set_title("Predicted density shape at t = %.1f" % time_plotted[7])
axs[2,0].set_title("Predicted density shape at t = %.1f" % time_plotted[8])
axs[2,1].set_title("Predicted density shape at t = %.1f" % time_plotted[9])

axs[2,2].set_title('Actual Initial Density where t=0 for comparing movement')
axs[2,3].set_title('Difference of Density Between t=0 and t=1')


# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[0,0].imshow(rho.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                              interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[0,0].invert_yaxis(), extend='both')
cbar.minorticks_on()
#################################################################################################

#Next final_time

t = time_plotted[1]*np.ones((spatial_discretization**2,1))
#convert to pytorch tensors
pt_t = Variable(torch.from_numpy(t).float(), requires_grad=False).to(device)
#get network outputs
_, _, _, pt_rho = net(pt_x1, pt_x2, pt_t)

#Convert back to numpy
_, _, _, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.cpu


# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[0,1].imshow(rho.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                              interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[0,1].invert_yaxis(), extend='both')
cbar.minorticks_on()
#################################################################################################

#Next final_time
```

```
t = time_plotted[2]*np.ones((spatial_discretization**2,1))
#convert to pytorch tensors
pt_t = Variable(torch.from_numpy(t).float(), requires_grad=False).to(device)
#get network outputs
_, _, _, pt_rho = net(pt_x1, pt_x2, pt_t)

#Convert back to numpy
_, _, _, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.cpu


# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[0,2].imshow(rho.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                              interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[0,2].invert_yaxis(), extend='both')
cbar.minorticks_on()
#################################################################################################


#Next final_time

t = time_plotted[3]*np.ones((spatial_discretization**2,1))
#convert to pytorch tensors
pt_t = Variable(torch.from_numpy(t).float(), requires_grad=False).to(device)
#get network outputs
_, _, _, pt_rho = net(pt_x1, pt_x2, pt_t)

#Convert back to numpy
_, _, _, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.cpu


# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[0,3].imshow(rho.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                              interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[0,3].invert_yaxis(), extend='both')
cbar.minorticks_on()
#################################################################################################
```

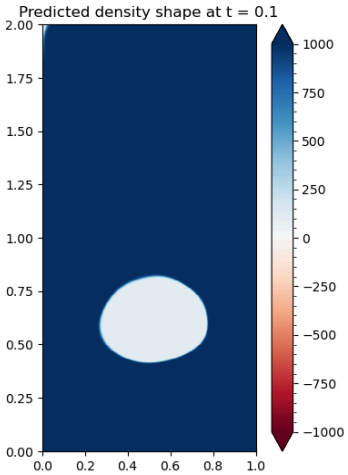
```
#Next final_time

t = time_plotted[4]*np.ones((spatial_discretization**2,1))
#convert to pytorch tensors
pt_t = Variable(torch.from_numpy(t).float(), requires_grad=False).to(device)
#get network outputs
_, _, _, pt_rho = net(pt_x1, pt_x2, pt_t)

#Convert back to numpy
_, _, _, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.cpu


# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[1, 0].imshow(rho.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                           interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[1,0].invert_yaxis(), extend='both')
cbar.minorticks_on()
#########################################################################################


#Next final_time

t = time_plotted[5]*np.ones((spatial_discretization**2,1))
#convert to pytorch tensors
pt_t = Variable(torch.from_numpy(t).float(), requires_grad=False).to(device)
#get network outputs
_, _, _, pt_rho = net(pt_x1, pt_x2, pt_t)

#Convert back to numpy
_, _, _, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.cpu


# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[1,1].imshow(rho.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                           interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[1,1].invert_yaxis(), extend='both')
cbar.minorticks_on()
#########################################################################################
```

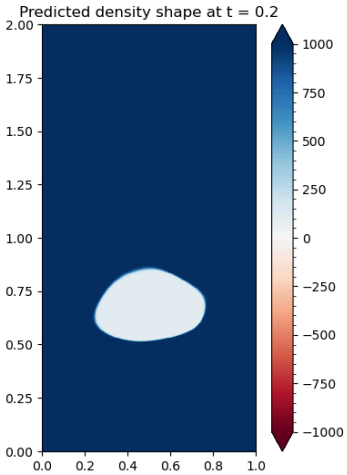
```
#Next final_time

t = time_plotted[6]*np.ones((spatial_discretization**2,1))
#convert to pytorch tensors
pt_t = Variable(torch.from_numpy(t).float(), requires_grad=False).to(device)
#get network outputs
_, _, _, pt_rho = net(pt_x1, pt_x2, pt_t)

#Convert back to numpy
_, _, _, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.cpu


# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[1,2].imshow(rho.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                              interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[1,2].invert_yaxis(), extend='both')
cbar.minorticks_on()
#########################################################################################################


#Next final_time

t = time_plotted[7]*np.ones((spatial_discretization**2,1))
#convert to pytorch tensors
pt_t = Variable(torch.from_numpy(t).float(), requires_grad=False).to(device)
#get network outputs
_, _, _, pt_rho = net(pt_x1, pt_x2, pt_t)

#Convert back to numpy
_, _, _, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.cpu


# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[1,3].imshow(rho.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                              interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[1,3].invert_yaxis(), extend='both')
```

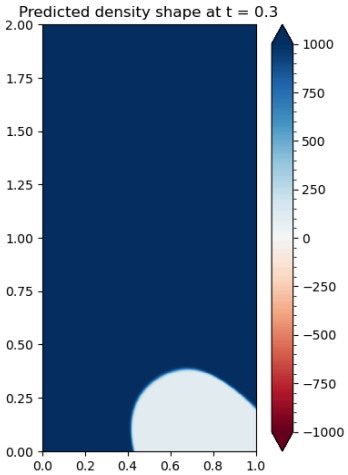
```
cbar.minorticks_on()
#########################################################################################################


#Next final_time

t = time_plotted[8]*np.ones((spatial_discretization**2,1))
#convert to pytorch tensors
pt_t = Variable(torch.from_numpy(t).float(), requires_grad=False).to(device)
#get network outputs
_, _, _, pt_rho = net(pt_x1, pt_x2, pt_t)

#Convert back to numpy
_, _, _, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.cpu


# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[2,0].imshow(rho.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                                  interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[2,0].invert_yaxis(), extend='both')
cbar.minorticks_on()
#########################################################################################################


#Next final_time

t = time_plotted[9]*np.ones((spatial_discretization**2,1))
#convert to pytorch tensors
pt_t = Variable(torch.from_numpy(t).float(), requires_grad=False).to(device)
#get network outputs
_, _, _, pt_rho = net(pt_x1, pt_x2, pt_t)

#Convert back to numpy
_, _, _, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.cpu


# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[2,1].imshow(rho.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                                  interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
```

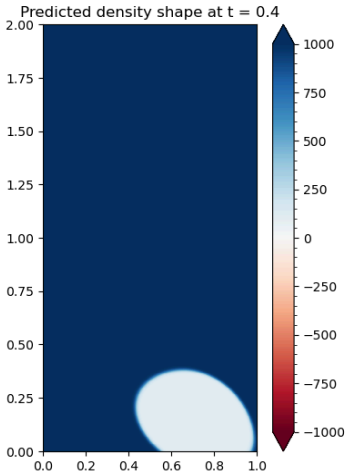
```
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[2,1].invert_yaxis(), extend='both')
cbar.minorticks_on()

#########################################################################################################

#Graph of IC at t=0

#Convert back to numpy
u1, u2, p, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.
rho_exact = rho_exact.data.cpu().numpy()
error = rho-rho_exact

# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[2,2].imshow(rho_exact.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                              interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[2,2].invert_yaxis(), extend='both')
cbar.minorticks_on()
#########################################################################################################

#Comparing t=0 and t=1



# Plot both positive and negative values between +/- 1000
pos_neg_clipped = axs[2,3].imshow(error.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                              interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[2,3].invert_yaxis(), extend='both')
cbar.minorticks_on()

#plt.xticks(np.arange(0, 1, step=.1))
#plt.yticks(np.arange(0, 2, step=.1))

#axs[2].legend()
fig.set_figheight(20)
fig.set_figwidth(20)

plt.show()
```

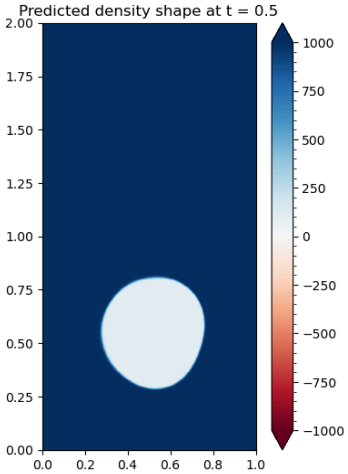
```
#Printing on Wide Screen
from IPython.display import display, HTML
display(HTML("<style>.container { width:100% !important; }</style>"))
```

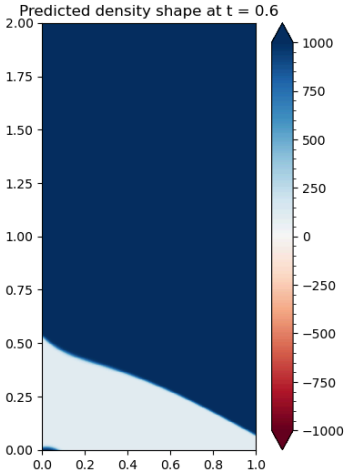

Predicted density shape at t = 0.1

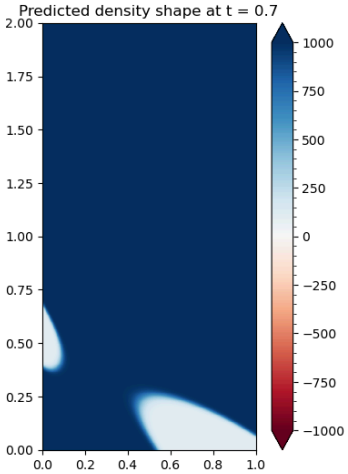

Predicted density shape at t = 0.2

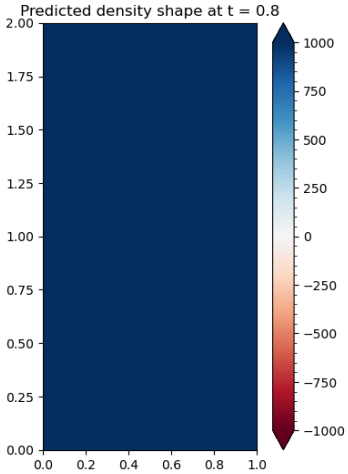

Predicted density shape at t = 0.3

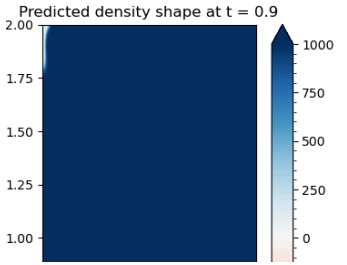

Predicted density shape at t = 0.4

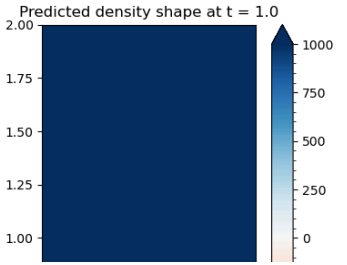

Predicted density shape at t = 0.5

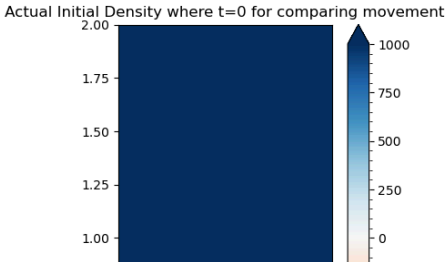

Predicted density shape at t = 0.6

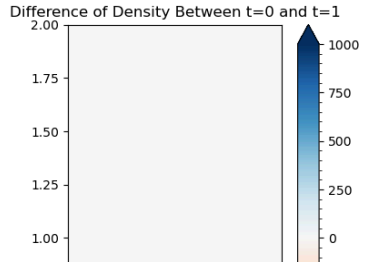

Predicted density shape at t = 0.7

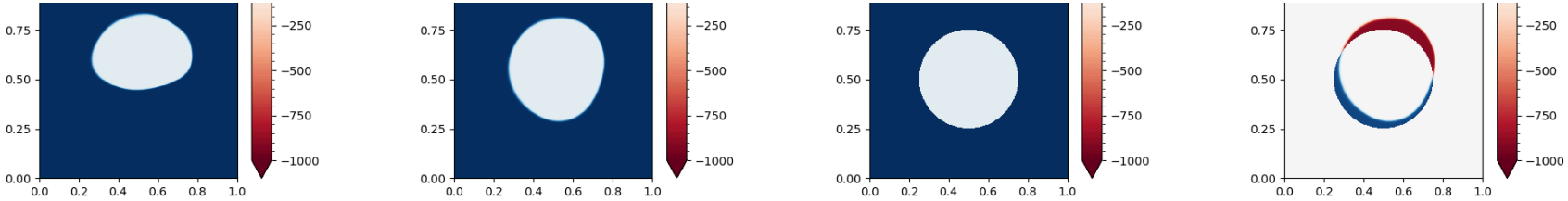

Predicted density shape at t = 0.8


Predicted density shape at t = 0.9


Predicted density shape at t = 1.0


Actual Initial Density where t=0 for comparing movement


Difference of Density Between t=0 and t=1



In [6]:
```
import torch
from torch import nn
import numpy as np
from torch.autograd import Variable
import matplotlib.pyplot as plt

from Forward_with_Layer_Setting import Net
from InitialConditionLoss import InitialCondition_rho

########################################

time_plotted = np.array([.1, .2, .3, .4, .5, .6, .7, .8, .9, 1])

########################################

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

net = Net().to(device)

net.load_state_dict(torch.load("ES_NSloss6th_lr0.005_t1.0_8_11_18h.pt", map_location=torch.device('cpu')))

'''
Iteration: 7000         Total Loss: tensor(50.8040)
IC Loss:  tensor(0.0051)        BC Loss:  tensor(5.7619e-07)    NS PDE Loss:  tensor(9.5337e-06)        NS
  *Saved ; Early Stopping for the latest NS PDE Loss of 6th decimal place

  *Saved ; Early Stopping for the latest IC Loss of 3rd decimal place

  *(Regularize Losses)Small NS loss and big IC loss ->> IC Coefficients are changed; mse_BC + 100 * mse_MvI

'''

#Graph at various time slices
```

```
spatial_discretization = 1000

#Define numpy arrays for inputs
x1 = np.linspace(net.x1_l,net.x1_u,spatial_discretization).reshape(spatial_discretization)
x2 = np.linspace(net.x2_l,net.x2_u,spatial_discretization).reshape(spatial_discretization)
x1x2 = np.array(np.meshgrid(x1, x2)).reshape(2,spatial_discretization**2)

t = time_plotted[0]*np.ones((spatial_discretization**2,1))

x1_input = x1x2[0].reshape(spatial_discretization**2, 1)
x2_input = x1x2[1].reshape(spatial_discretization**2, 1)

x1x2 = [x1_input, x2_input]

#convert to pytorch tensors
pt_x1 = Variable(torch.from_numpy(x1_input).float(), requires_grad=False).to(device)
pt_x2 = Variable(torch.from_numpy(x2_input).float(), requires_grad=False).to(device)
pt_t = Variable(torch.from_numpy(t).float(), requires_grad=False).to(device)

#get network outputs
pt_u1, pt_u2, pt_p, pt_rho = net(pt_x1, pt_x2, pt_t)

#Convert back to numpy
u1, u2, p, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.

#get actual initial condition
rho_exact = InitialCondition_rho(net, pt_x1, pt_x2)




X, Y = np.meshgrid(x1, x2)



fig, axs = plt.subplots(3,4, figsize=(10,5))
#fig.suptitle(f'Time = {time_plotted}')
fig.tight_layout()
axs[0,0].set_title("Predicted density shape at t = %.1f" % time_plotted[0])
axs[0,1].set_title("Predicted density shape at t = %.1f" % time_plotted[1])
axs[0,2].set_title("Predicted density shape at t = %.1f" % time_plotted[2])
```

```
axs[0,3].set_title("Predicted density shape at t = %.1f" % time_plotted[3])
axs[1,0].set_title("Predicted density shape at t = %.1f" % time_plotted[4])
axs[1,1].set_title("Predicted density shape at t = %.1f" % time_plotted[5])
axs[1,2].set_title("Predicted density shape at t = %.1f" % time_plotted[6])
axs[1,3].set_title("Predicted density shape at t = %.1f" % time_plotted[7])
axs[2,0].set_title("Predicted density shape at t = %.1f" % time_plotted[8])
axs[2,1].set_title("Predicted density shape at t = %.1f" % time_plotted[9])

axs[2,2].set_title('Actual Initial Density where t=0 for comparing movement')
axs[2,3].set_title('Difference of Density Between t=0 and t=1')


# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[0,0].imshow(rho.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                              interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[0,0].invert_yaxis(), extend='both')
cbar.minorticks_on()
#################################################################################################

#Next final_time

t = time_plotted[1]*np.ones((spatial_discretization**2,1))
#convert to pytorch tensors
pt_t = Variable(torch.from_numpy(t).float(), requires_grad=False).to(device)
#get network outputs
_, _, _, pt_rho = net(pt_x1, pt_x2, pt_t)

#Convert back to numpy
_, _, _, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.cpu


# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[0,1].imshow(rho.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                              interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[0,1].invert_yaxis(), extend='both')
cbar.minorticks_on()
#################################################################################################
```

```
#Next final_time

t = time_plotted[2]*np.ones((spatial_discretization**2,1))
#convert to pytorch tensors
pt_t = Variable(torch.from_numpy(t).float(), requires_grad=False).to(device)
#get network outputs
_, _, _, pt_rho = net(pt_x1, pt_x2, pt_t)

#Convert back to numpy
_, _, _, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.cpu


# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[0,2].imshow(rho.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                            interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[0,2].invert_yaxis(), extend='both')
cbar.minorticks_on()
#########################################################################################


#Next final_time

t = time_plotted[3]*np.ones((spatial_discretization**2,1))
#convert to pytorch tensors
pt_t = Variable(torch.from_numpy(t).float(), requires_grad=False).to(device)
#get network outputs
_, _, _, pt_rho = net(pt_x1, pt_x2, pt_t)

#Convert back to numpy
_, _, _, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.cpu


# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[0,3].imshow(rho.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                            interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[0,3].invert_yaxis(), extend='both')
cbar.minorticks_on()
#########################################################################################
```

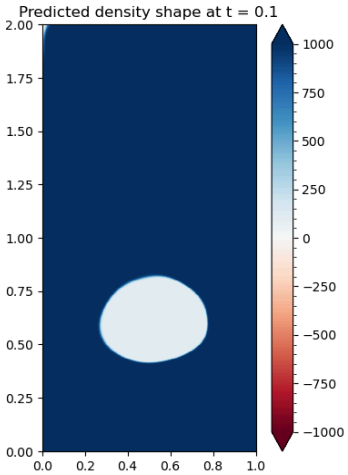
```
#Next final_time

t = time_plotted[4]*np.ones((spatial_discretization**2,1))
#convert to pytorch tensors
pt_t = Variable(torch.from_numpy(t).float(), requires_grad=False).to(device)
#get network outputs
_, _, _, pt_rho = net(pt_x1, pt_x2, pt_t)

#Convert back to numpy
_, _, _, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.cpu


# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[1, 0].imshow(rho.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                              interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[1,0].invert_yaxis(), extend='both')
cbar.minorticks_on()
###############################################################################################


#Next final_time

t = time_plotted[5]*np.ones((spatial_discretization**2,1))
#convert to pytorch tensors
pt_t = Variable(torch.from_numpy(t).float(), requires_grad=False).to(device)
#get network outputs
_, _, _, pt_rho = net(pt_x1, pt_x2, pt_t)

#Convert back to numpy
_, _, _, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.cpu


# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[1,1].imshow(rho.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                              interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[1,1].invert_yaxis(), extend='both')
```

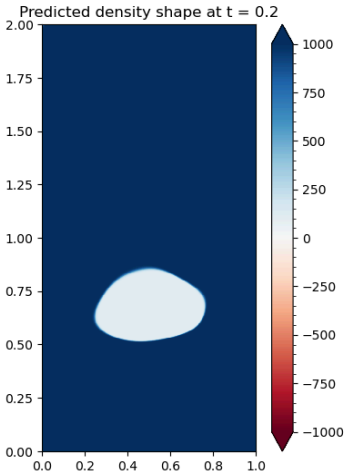
```
cbar.minorticks_on()
#########################################################################################################


#Next final_time

t = time_plotted[6]*np.ones((spatial_discretization**2,1))
#convert to pytorch tensors
pt_t = Variable(torch.from_numpy(t).float(), requires_grad=False).to(device)
#get network outputs
_, _, _, pt_rho = net(pt_x1, pt_x2, pt_t)

#Convert back to numpy
_, _, _, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.cpu


# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[1,2].imshow(rho.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                              interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[1,2].invert_yaxis(), extend='both')
cbar.minorticks_on()
#########################################################################################################


#Next final_time

t = time_plotted[7]*np.ones((spatial_discretization**2,1))
#convert to pytorch tensors
pt_t = Variable(torch.from_numpy(t).float(), requires_grad=False).to(device)
#get network outputs
_, _, _, pt_rho = net(pt_x1, pt_x2, pt_t)

#Convert back to numpy
_, _, _, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.cpu


# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[1,3].imshow(rho.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                              interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
```

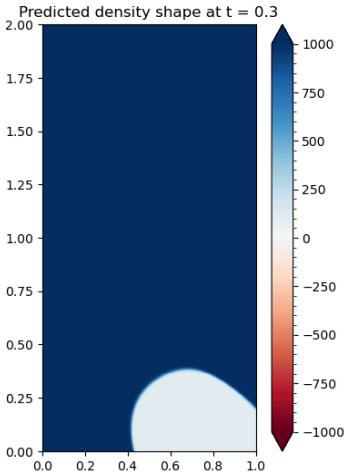
```
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[1,3].invert_yaxis(), extend='both')
cbar.minorticks_on()
###########################################################################################


#Next final_time

t = time_plotted[8]*np.ones((spatial_discretization**2,1))
#convert to pytorch tensors
pt_t = Variable(torch.from_numpy(t).float(), requires_grad=False).to(device)
#get network outputs
_, _, _, pt_rho = net(pt_x1, pt_x2, pt_t)

#Convert back to numpy
_, _, _, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.cpu


# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[2,0].imshow(rho.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                              interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[2,0].invert_yaxis(), extend='both')
cbar.minorticks_on()
###########################################################################################


#Next final_time

t = time_plotted[9]*np.ones((spatial_discretization**2,1))
#convert to pytorch tensors
pt_t = Variable(torch.from_numpy(t).float(), requires_grad=False).to(device)
#get network outputs
_, _, _, pt_rho = net(pt_x1, pt_x2, pt_t)

#Convert back to numpy
_, _, _, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.cpu


# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[2,1].imshow(rho.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
```

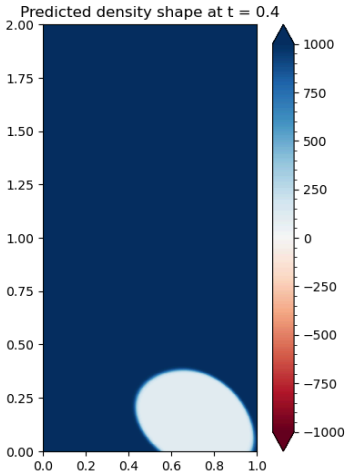
```
                                interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[2,1].invert_yaxis(), extend='both')
cbar.minorticks_on()

#################################################################################################

#Graph of IC at t=0

#Convert back to numpy
u1, u2, p, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.
rho_exact = rho_exact.data.cpu().numpy()
error = rho-rho_exact

# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[2,2].imshow(rho_exact.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                                interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[2,2].invert_yaxis(), extend='both')
cbar.minorticks_on()
#################################################################################################

#Comparing t=0 and t=1



# Plot both positive and negative values between +/- 1000
pos_neg_clipped = axs[2,3].imshow(error.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                                interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[2,3].invert_yaxis(), extend='both')
cbar.minorticks_on()

#plt.xticks(np.arange(0, 1, step=.1))
#plt.yticks(np.arange(0, 2, step=.1))

#axs[2].legend()
fig.set_figheight(20)
fig.set_figwidth(20)
```

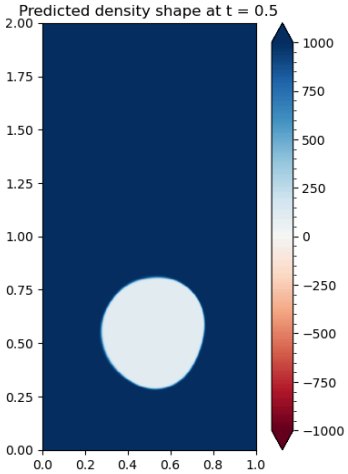
```
plt.show()

#Printing on Wide Screen
from IPython.display import display, HTML
display(HTML("<style>.container { width:100% !important; }</style>"))
```

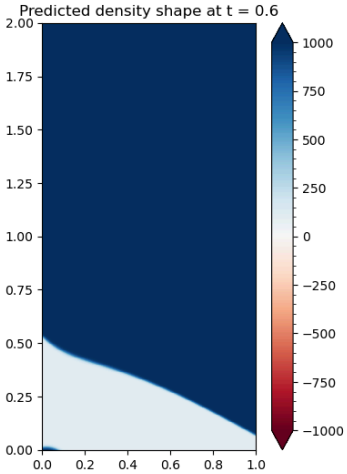

Predicted density shape at t = 0.1

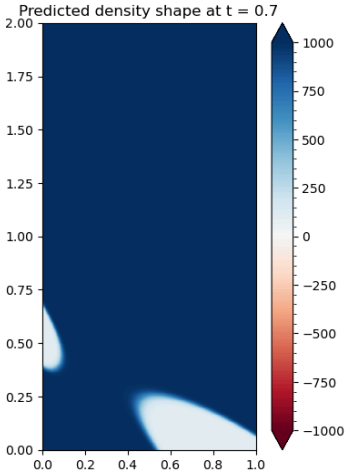

Predicted density shape at t = 0.2

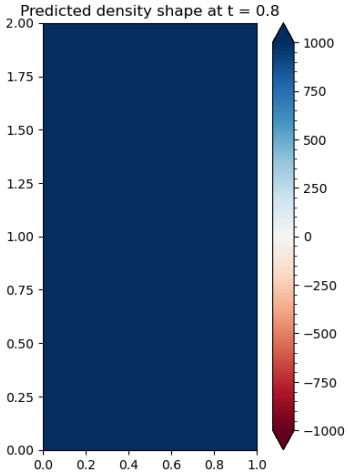

Predicted density shape at t = 0.3

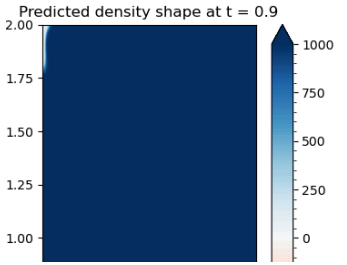

Predicted density shape at t = 0.4

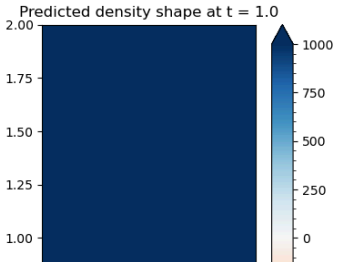

Predicted density shape at t = 0.5

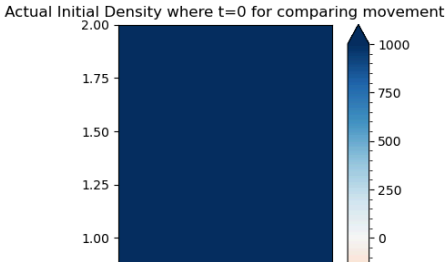

Predicted density shape at t = 0.6

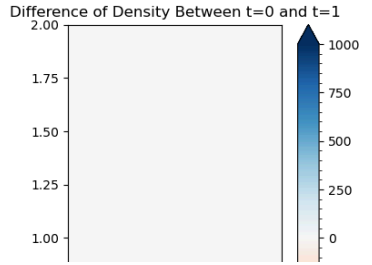

Predicted density shape at t = 0.7

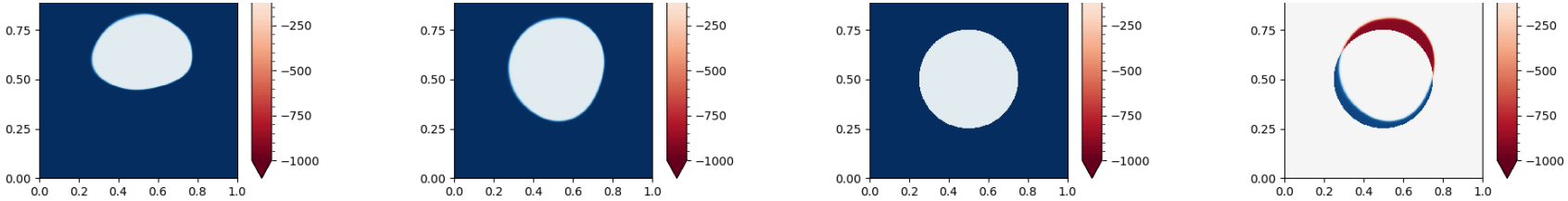

Predicted density shape at t = 0.8


Predicted density shape at t = 0.9


Predicted density shape at t = 1.0


Actual Initial Density where t=0 for comparing movement


Difference of Density Between t=0 and t=1



In [7]:
```
import torch
from torch import nn
import numpy as np
from torch.autograd import Variable
import matplotlib.pyplot as plt

from Forward_with_Layer_Setting import Net
from InitialConditionLoss import InitialCondition_rho

########################################

time_plotted = np.array([.1, .2, .3, .4, .5, .6, .7, .8, .9, 1])

########################################

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

net = Net().to(device)

net.load_state_dict(torch.load("ES_NSloss4th_lr0.005_t1.0_8_11_18h.pt", map_location=torch.device('cpu')))

'''
Iteration: 6000         Total Loss: tensor(10.2988)
IC Loss:  tensor(0.0051)        BC Loss:  tensor(1.6402e-06)    NS PDE Loss:  tensor(0.0009)    NS Div Free
  *Saved ; Early Stopping for the latest NS PDE Loss of 4th decimal place

  *Saved ; Early Stopping for the latest IC Loss of 3rd decimal place


'''

#Graph at various time slices

spatial_discretization = 1000
```

```
#Define numpy arrays for inputs
x1 = np.linspace(net.x1_l,net.x1_u,spatial_discretization).reshape(spatial_discretization)
x2 = np.linspace(net.x2_l,net.x2_u,spatial_discretization).reshape(spatial_discretization)
x1x2 = np.array(np.meshgrid(x1, x2)).reshape(2,spatial_discretization**2)

t = time_plotted[0]*np.ones((spatial_discretization**2,1))

x1_input = x1x2[0].reshape(spatial_discretization**2, 1)
x2_input = x1x2[1].reshape(spatial_discretization**2, 1)

x1x2 = [x1_input, x2_input]

#convert to pytorch tensors
pt_x1 = Variable(torch.from_numpy(x1_input).float(), requires_grad=False).to(device)
pt_x2 = Variable(torch.from_numpy(x2_input).float(), requires_grad=False).to(device)
pt_t = Variable(torch.from_numpy(t).float(), requires_grad=False).to(device)

#get network outputs
pt_u1, pt_u2, pt_p, pt_rho = net(pt_x1, pt_x2, pt_t)

#Convert back to numpy
u1, u2, p, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.

#get actual initial condition
rho_exact = InitialCondition_rho(net, pt_x1, pt_x2)




X, Y = np.meshgrid(x1, x2)



fig, axs = plt.subplots(3,4, figsize=(10,5))
#fig.suptitle(f'Time = {time_plotted}')
fig.tight_layout()
axs[0,0].set_title("Predicted density shape at t = %.1f" % time_plotted[0])
axs[0,1].set_title("Predicted density shape at t = %.1f" % time_plotted[1])
axs[0,2].set_title("Predicted density shape at t = %.1f" % time_plotted[2])
axs[0,3].set_title("Predicted density shape at t = %.1f" % time_plotted[3])
```

```
axs[1,0].set_title("Predicted density shape at t = %.1f" % time_plotted[4])
axs[1,1].set_title("Predicted density shape at t = %.1f" % time_plotted[5])
axs[1,2].set_title("Predicted density shape at t = %.1f" % time_plotted[6])
axs[1,3].set_title("Predicted density shape at t = %.1f" % time_plotted[7])
axs[2,0].set_title("Predicted density shape at t = %.1f" % time_plotted[8])
axs[2,1].set_title("Predicted density shape at t = %.1f" % time_plotted[9])

axs[2,2].set_title('Actual Initial Density where t=0 for comparing movement')
axs[2,3].set_title('Difference of Density Between t=0 and t=1')


# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[0,0].imshow(rho.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                                  interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[0,0].invert_yaxis(), extend='both')
cbar.minorticks_on()
##########################################################################################################

#Next final_time

t = time_plotted[1]*np.ones((spatial_discretization**2,1))
#convert to pytorch tensors
pt_t = Variable(torch.from_numpy(t).float(), requires_grad=False).to(device)
#get network outputs
_, _, _, pt_rho = net(pt_x1, pt_x2, pt_t)

#Convert back to numpy
_, _, _, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.cpu


# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[0,1].imshow(rho.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                                  interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[0,1].invert_yaxis(), extend='both')
cbar.minorticks_on()
##########################################################################################################

#Next final_time
```

```
t = time_plotted[2]*np.ones((spatial_discretization**2,1))
#convert to pytorch tensors
pt_t = Variable(torch.from_numpy(t).float(), requires_grad=False).to(device)
#get network outputs
_, _, _, pt_rho = net(pt_x1, pt_x2, pt_t)

#Convert back to numpy
_, _, _, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.cpu


# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[0,2].imshow(rho.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                              interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[0,2].invert_yaxis(), extend='both')
cbar.minorticks_on()
#########################################################################################################


#Next final_time

t = time_plotted[3]*np.ones((spatial_discretization**2,1))
#convert to pytorch tensors
pt_t = Variable(torch.from_numpy(t).float(), requires_grad=False).to(device)
#get network outputs
_, _, _, pt_rho = net(pt_x1, pt_x2, pt_t)

#Convert back to numpy
_, _, _, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.cpu


# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[0,3].imshow(rho.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                              interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[0,3].invert_yaxis(), extend='both')
cbar.minorticks_on()
#########################################################################################################
```

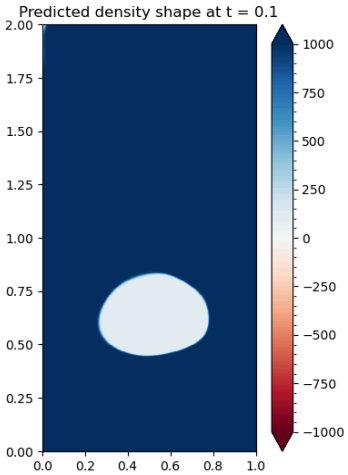
```
#Next final_time

t = time_plotted[4]*np.ones((spatial_discretization**2,1))
#convert to pytorch tensors
pt_t = Variable(torch.from_numpy(t).float(), requires_grad=False).to(device)
#get network outputs
_, _, _, pt_rho = net(pt_x1, pt_x2, pt_t)

#Convert back to numpy
_, _, _, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.cpu


# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[1, 0].imshow(rho.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                           interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[1,0].invert_yaxis(), extend='both')
cbar.minorticks_on()
#########################################################################################################


#Next final_time

t = time_plotted[5]*np.ones((spatial_discretization**2,1))
#convert to pytorch tensors
pt_t = Variable(torch.from_numpy(t).float(), requires_grad=False).to(device)
#get network outputs
_, _, _, pt_rho = net(pt_x1, pt_x2, pt_t)

#Convert back to numpy
_, _, _, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.cpu


# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[1,1].imshow(rho.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                           interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[1,1].invert_yaxis(), extend='both')
cbar.minorticks_on()
```

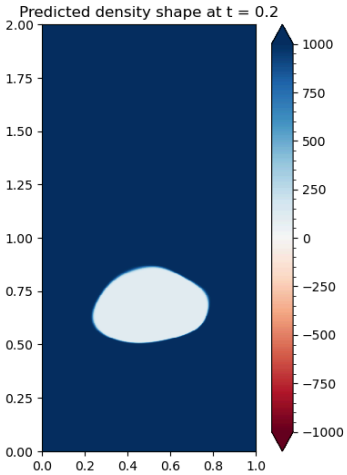
```
#########################################################################################################


#Next final_time

t = time_plotted[6]*np.ones((spatial_discretization**2,1))
#convert to pytorch tensors
pt_t = Variable(torch.from_numpy(t).float(), requires_grad=False).to(device)
#get network outputs
_, _, _, pt_rho = net(pt_x1, pt_x2, pt_t)

#Convert back to numpy
_, _, _, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.cpu


# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[1,2].imshow(rho.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                              interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[1,2].invert_yaxis(), extend='both')
cbar.minorticks_on()
#########################################################################################################


#Next final_time

t = time_plotted[7]*np.ones((spatial_discretization**2,1))
#convert to pytorch tensors
pt_t = Variable(torch.from_numpy(t).float(), requires_grad=False).to(device)
#get network outputs
_, _, _, pt_rho = net(pt_x1, pt_x2, pt_t)

#Convert back to numpy
_, _, _, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.cpu


# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[1,3].imshow(rho.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                              interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
```

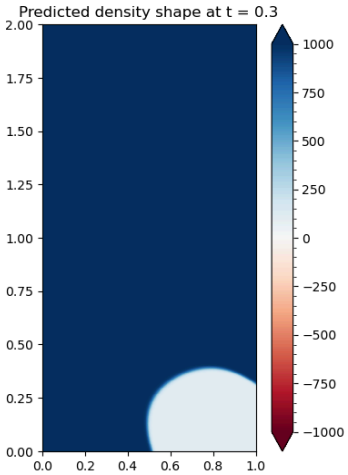
```
cbar = fig.colorbar(pos_neg_clipped, ax=axs[1,3].invert_yaxis(), extend='both')
cbar.minorticks_on()
#########################################################################################################


#Next final_time

t = time_plotted[8]*np.ones((spatial_discretization**2,1))
#convert to pytorch tensors
pt_t = Variable(torch.from_numpy(t).float(), requires_grad=False).to(device)
#get network outputs
_, _, _, pt_rho = net(pt_x1, pt_x2, pt_t)

#Convert back to numpy
_, _, _, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.cpu


# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[2,0].imshow(rho.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                                  interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[2,0].invert_yaxis(), extend='both')
cbar.minorticks_on()
#########################################################################################################


#Next final_time

t = time_plotted[9]*np.ones((spatial_discretization**2,1))
#convert to pytorch tensors
pt_t = Variable(torch.from_numpy(t).float(), requires_grad=False).to(device)
#get network outputs
_, _, _, pt_rho = net(pt_x1, pt_x2, pt_t)

#Convert back to numpy
_, _, _, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.cpu


# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[2,1].imshow(rho.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                                  interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
```

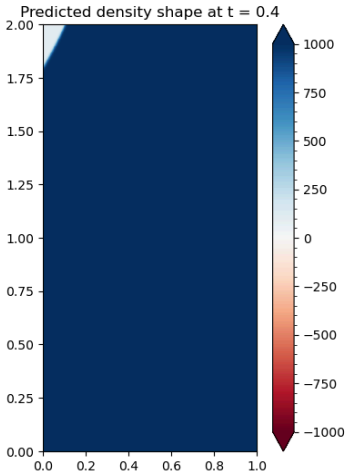
```
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[2,1].invert_yaxis(), extend='both')
cbar.minorticks_on()

#########################################################################################

#Graph of IC at t=0

#Convert back to numpy
u1, u2, p, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.
rho_exact = rho_exact.data.cpu().numpy()
error = rho-rho_exact

# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[2,2].imshow(rho_exact.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                              interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[2,2].invert_yaxis(), extend='both')
cbar.minorticks_on()
#########################################################################################

#Comparing t=0 and t=1



# Plot both positive and negative values between +/- 1000
pos_neg_clipped = axs[2,3].imshow(error.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                              interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[2,3].invert_yaxis(), extend='both')
cbar.minorticks_on()

#plt.xticks(np.arange(0, 1, step=.1))
#plt.yticks(np.arange(0, 2, step=.1))

#axs[2].legend()
fig.set_figheight(20)
fig.set_figwidth(20)
```

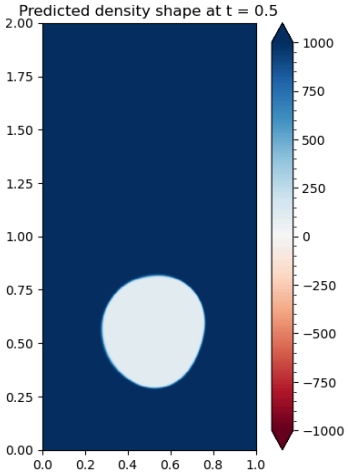
```
plt.show()

#Printing on Wide Screen
from IPython.display import display, HTML
display(HTML("<style>.container { width:100% !important; }</style>"))
```

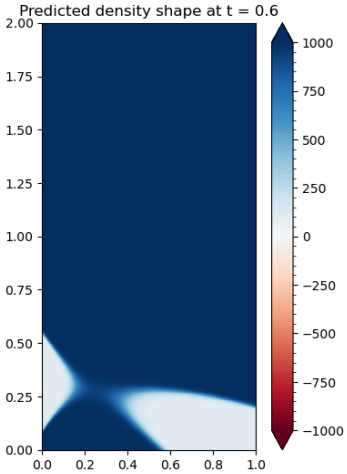

Predicted density shape at t = 0.1

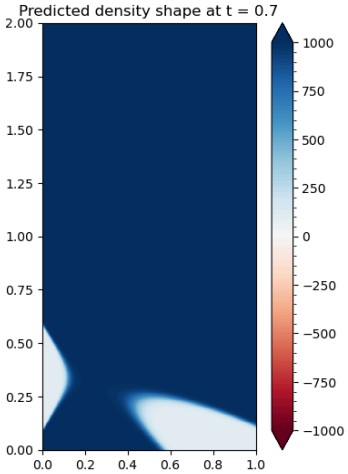

Predicted density shape at t = 0.2

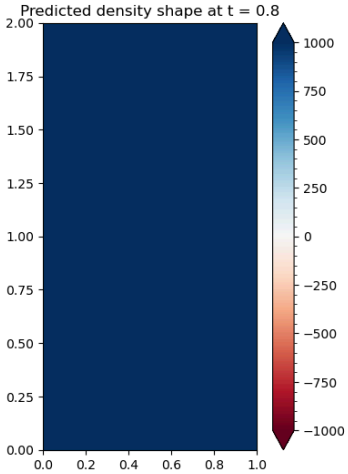

Predicted density shape at t = 0.3

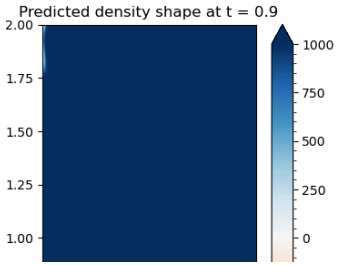

Predicted density shape at t = 0.4

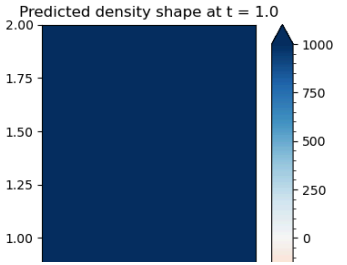

Predicted density shape at t = 0.5

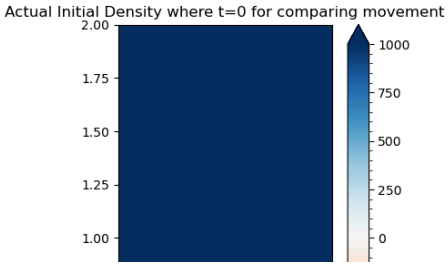

Predicted density shape at t = 0.6

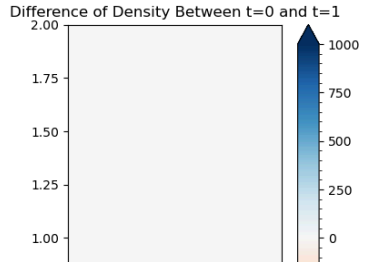

Predicted density shape at t = 0.7

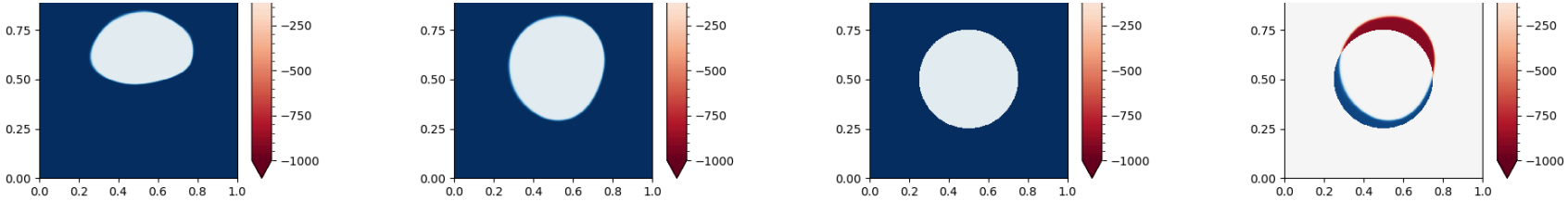

Predicted density shape at t = 0.8


Predicted density shape at t = 0.9


Predicted density shape at t = 1.0


Actual Initial Density where t=0 for comparing movement


Difference of Density Between t=0 and t=1



In [8]:
```
import torch
from torch import nn
import numpy as np
from torch.autograd import Variable
import matplotlib.pyplot as plt

from Forward_with_Layer_Setting import Net
from InitialConditionLoss import InitialCondition_rho

##########################################

time_plotted = np.array([.1, .2, .3, .4, .5, .6, .7, .8, .9, 1])

##########################################

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

net = Net().to(device)

net.load_state_dict(torch.load("ES_NSloss3rd_lr0.005_t1.0_8_11_18h.pt", map_location=torch.device('cpu')))

'''
Iteration: 4500         Total Loss: tensor(10.0714)
IC Loss:  tensor(0.0050)        BC Loss:  tensor(1.1285e-05)    NS PDE Loss:  tensor(0.0019)    NS Div Free
  *Saved ; Early Stopping for the latest NS PDE Loss of 3rd decimal place

  *Saved ; Early Stopping for the latest IC Loss of 3rd decimal place



'''

#Graph at various time slices
```

```
spatial_discretization = 1000

#Define numpy arrays for inputs
x1 = np.linspace(net.x1_l,net.x1_u,spatial_discretization).reshape(spatial_discretization)
x2 = np.linspace(net.x2_l,net.x2_u,spatial_discretization).reshape(spatial_discretization)
x1x2 = np.array(np.meshgrid(x1, x2)).reshape(2,spatial_discretization**2)

t = time_plotted[0]*np.ones((spatial_discretization**2,1))

x1_input = x1x2[0].reshape(spatial_discretization**2, 1)
x2_input = x1x2[1].reshape(spatial_discretization**2, 1)

x1x2 = [x1_input, x2_input]

#convert to pytorch tensors
pt_x1 = Variable(torch.from_numpy(x1_input).float(), requires_grad=False).to(device)
pt_x2 = Variable(torch.from_numpy(x2_input).float(), requires_grad=False).to(device)
pt_t = Variable(torch.from_numpy(t).float(), requires_grad=False).to(device)

#get network outputs
pt_u1, pt_u2, pt_p, pt_rho = net(pt_x1, pt_x2, pt_t)

#Convert back to numpy
u1, u2, p, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.

#get actual initial condition
rho_exact = InitialCondition_rho(net, pt_x1, pt_x2)




X, Y = np.meshgrid(x1, x2)



fig, axs = plt.subplots(3,4, figsize=(10,5))
#fig.suptitle(f'Time = {time_plotted}')
fig.tight_layout()
axs[0,0].set_title("Predicted density shape at t = %.1f" % time_plotted[0])
axs[0,1].set_title("Predicted density shape at t = %.1f" % time_plotted[1])
axs[0,2].set_title("Predicted density shape at t = %.1f" % time_plotted[2])
```

```
axs[0,3].set_title("Predicted density shape at t = %.1f" % time_plotted[3])
axs[1,0].set_title("Predicted density shape at t = %.1f" % time_plotted[4])
axs[1,1].set_title("Predicted density shape at t = %.1f" % time_plotted[5])
axs[1,2].set_title("Predicted density shape at t = %.1f" % time_plotted[6])
axs[1,3].set_title("Predicted density shape at t = %.1f" % time_plotted[7])
axs[2,0].set_title("Predicted density shape at t = %.1f" % time_plotted[8])
axs[2,1].set_title("Predicted density shape at t = %.1f" % time_plotted[9])

axs[2,2].set_title('Actual Initial Density where t=0 for comparing movement')
axs[2,3].set_title('Difference of Density Between t=0 and t=1')


# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[0,0].imshow(rho.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                              interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[0,0].invert_yaxis(), extend='both')
cbar.minorticks_on()
#################################################################################################

#Next final_time

t = time_plotted[1]*np.ones((spatial_discretization**2,1))
#convert to pytorch tensors
pt_t = Variable(torch.from_numpy(t).float(), requires_grad=False).to(device)
#get network outputs
_, _, _, pt_rho = net(pt_x1, pt_x2, pt_t)

#Convert back to numpy
_, _, _, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.cpu


# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[0,1].imshow(rho.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                              interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[0,1].invert_yaxis(), extend='both')
cbar.minorticks_on()
#################################################################################################
```

```
#Next final_time

t = time_plotted[2]*np.ones((spatial_discretization**2,1))
#convert to pytorch tensors
pt_t = Variable(torch.from_numpy(t).float(), requires_grad=False).to(device)
#get network outputs
_, _, _, pt_rho = net(pt_x1, pt_x2, pt_t)

#Convert back to numpy
_, _, _, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.cpu


# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[0,2].imshow(rho.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                            interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[0,2].invert_yaxis(), extend='both')
cbar.minorticks_on()
#########################################################################################


#Next final_time

t = time_plotted[3]*np.ones((spatial_discretization**2,1))
#convert to pytorch tensors
pt_t = Variable(torch.from_numpy(t).float(), requires_grad=False).to(device)
#get network outputs
_, _, _, pt_rho = net(pt_x1, pt_x2, pt_t)

#Convert back to numpy
_, _, _, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.cpu


# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[0,3].imshow(rho.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                            interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[0,3].invert_yaxis(), extend='both')
cbar.minorticks_on()
#########################################################################################
```

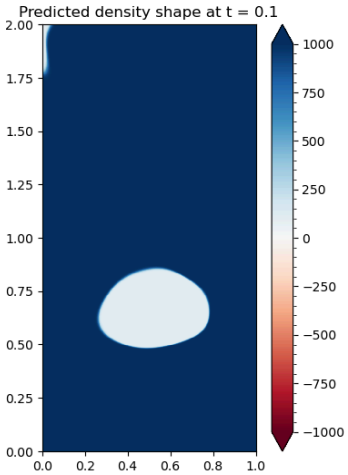
```
#Next final_time

t = time_plotted[4]*np.ones((spatial_discretization**2,1))
#convert to pytorch tensors
pt_t = Variable(torch.from_numpy(t).float(), requires_grad=False).to(device)
#get network outputs
_, _, _, pt_rho = net(pt_x1, pt_x2, pt_t)

#Convert back to numpy
_, _, _, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.cpu


# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[1, 0].imshow(rho.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                              interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[1,0].invert_yaxis(), extend='both')
cbar.minorticks_on()
###############################################################################################


#Next final_time

t = time_plotted[5]*np.ones((spatial_discretization**2,1))
#convert to pytorch tensors
pt_t = Variable(torch.from_numpy(t).float(), requires_grad=False).to(device)
#get network outputs
_, _, _, pt_rho = net(pt_x1, pt_x2, pt_t)

#Convert back to numpy
_, _, _, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.cpu


# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[1,1].imshow(rho.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                              interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[1,1].invert_yaxis(), extend='both')
```

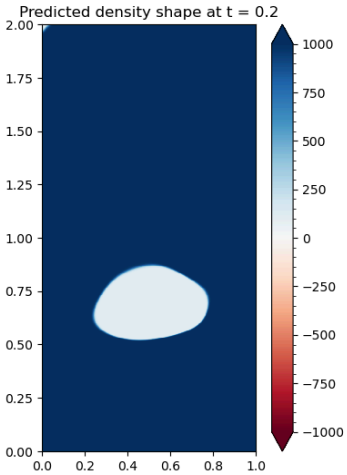
```
cbar.minorticks_on()
#########################################################################################################


#Next final_time

t = time_plotted[6]*np.ones((spatial_discretization**2,1))
#convert to pytorch tensors
pt_t = Variable(torch.from_numpy(t).float(), requires_grad=False).to(device)
#get network outputs
_, _, _, pt_rho = net(pt_x1, pt_x2, pt_t)

#Convert back to numpy
_, _, _, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.cpu


# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[1,2].imshow(rho.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                              interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[1,2].invert_yaxis(), extend='both')
cbar.minorticks_on()
#########################################################################################################


#Next final_time

t = time_plotted[7]*np.ones((spatial_discretization**2,1))
#convert to pytorch tensors
pt_t = Variable(torch.from_numpy(t).float(), requires_grad=False).to(device)
#get network outputs
_, _, _, pt_rho = net(pt_x1, pt_x2, pt_t)

#Convert back to numpy
_, _, _, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.cpu


# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[1,3].imshow(rho.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                              interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
```

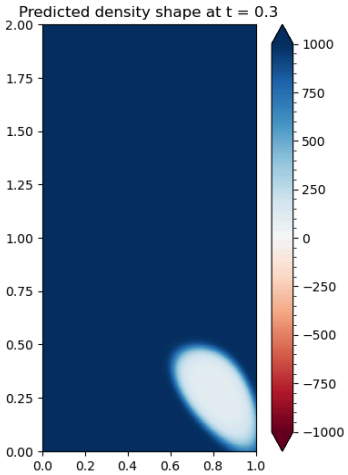
```
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[1,3].invert_yaxis(), extend='both')
cbar.minorticks_on()
###########################################################################################


#Next final_time

t = time_plotted[8]*np.ones((spatial_discretization**2,1))
#convert to pytorch tensors
pt_t = Variable(torch.from_numpy(t).float(), requires_grad=False).to(device)
#get network outputs
_, _, _, pt_rho = net(pt_x1, pt_x2, pt_t)

#Convert back to numpy
_, _, _, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.cpu


# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[2,0].imshow(rho.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                              interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[2,0].invert_yaxis(), extend='both')
cbar.minorticks_on()
###########################################################################################


#Next final_time

t = time_plotted[9]*np.ones((spatial_discretization**2,1))
#convert to pytorch tensors
pt_t = Variable(torch.from_numpy(t).float(), requires_grad=False).to(device)
#get network outputs
_, _, _, pt_rho = net(pt_x1, pt_x2, pt_t)

#Convert back to numpy
_, _, _, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.cpu


# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[2,1].imshow(rho.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
```

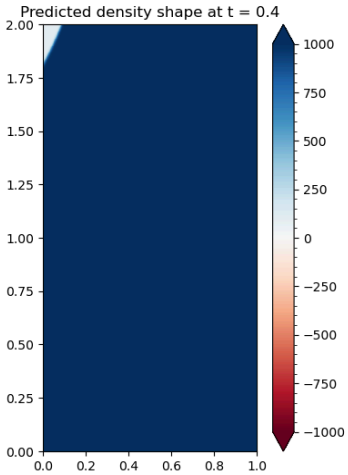
```
                                interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[2,1].invert_yaxis(), extend='both')
cbar.minorticks_on()

#################################################################################################

#Graph of IC at t=0

#Convert back to numpy
u1, u2, p, rho = pt_u1.data.cpu().numpy(), pt_u2.data.cpu().numpy(), pt_p.data.cpu().numpy(), pt_rho.data.
rho_exact = rho_exact.data.cpu().numpy()
error = rho-rho_exact

# Plot both positive and negative values between +/- 1.2
pos_neg_clipped = axs[2,2].imshow(rho_exact.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                                interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[2,2].invert_yaxis(), extend='both')
cbar.minorticks_on()
#################################################################################################

#Comparing t=0 and t=1



# Plot both positive and negative values between +/- 1000
pos_neg_clipped = axs[2,3].imshow(error.reshape(X.shape), cmap='RdBu', vmin=-1000, vmax=1000,
                                interpolation='none', extent = (net.x1_l,net.x1_u, net.x2_u, net.x2_l))
# Add minorticks on the colorbar to make it easy to read the
# values off the colorbar.
cbar = fig.colorbar(pos_neg_clipped, ax=axs[2,3].invert_yaxis(), extend='both')
cbar.minorticks_on()

#plt.xticks(np.arange(0, 1, step=.1))
#plt.yticks(np.arange(0, 2, step=.1))

#axs[2].legend()
fig.set_figheight(20)
fig.set_figwidth(20)
```

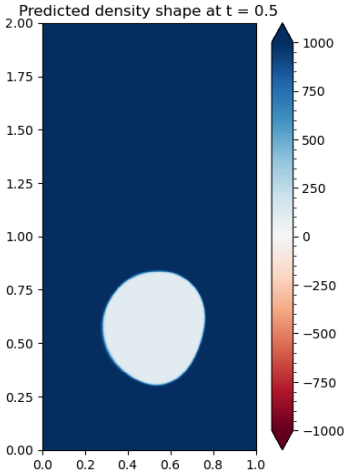
```
plt.show()

#Printing on Wide Screen
from IPython.display import display, HTML
display(HTML("<style>.container { width:100% !important; }</style>"))
```

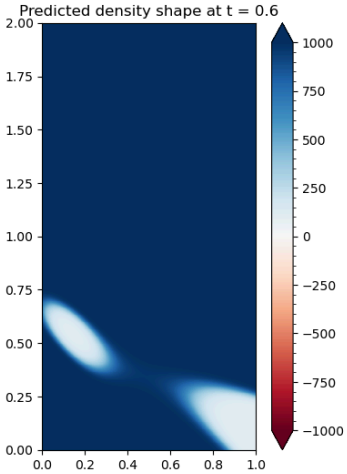

Predicted density shape at t = 0.1

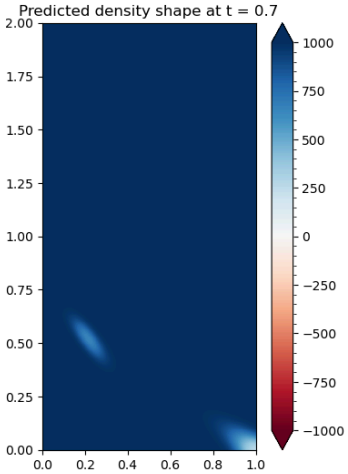

Predicted density shape at t = 0.2

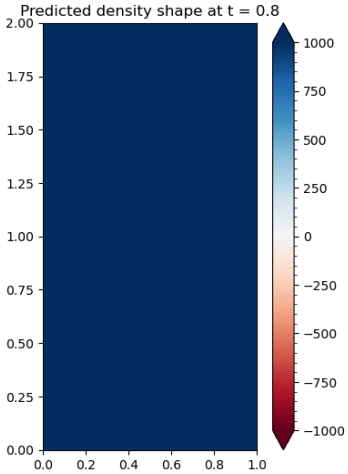

Predicted density shape at t = 0.3

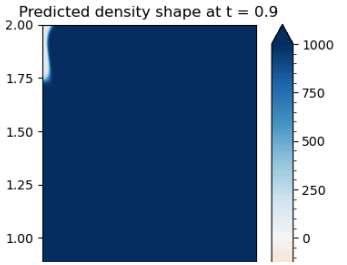

Predicted density shape at t = 0.4

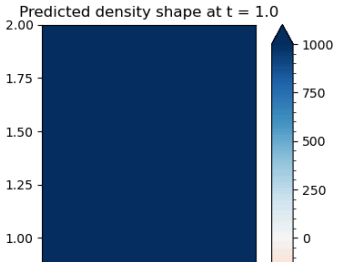

Predicted density shape at t = 0.5

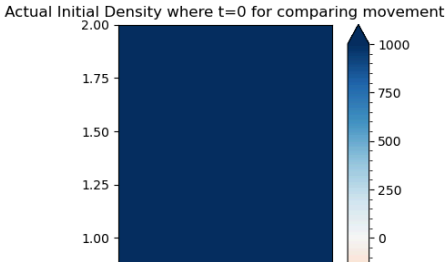

Predicted density shape at t = 0.6

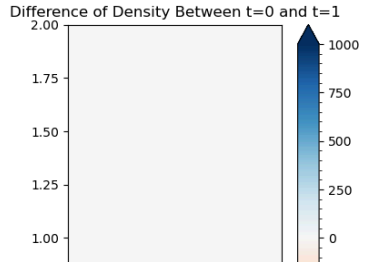

Predicted density shape at t = 0.7

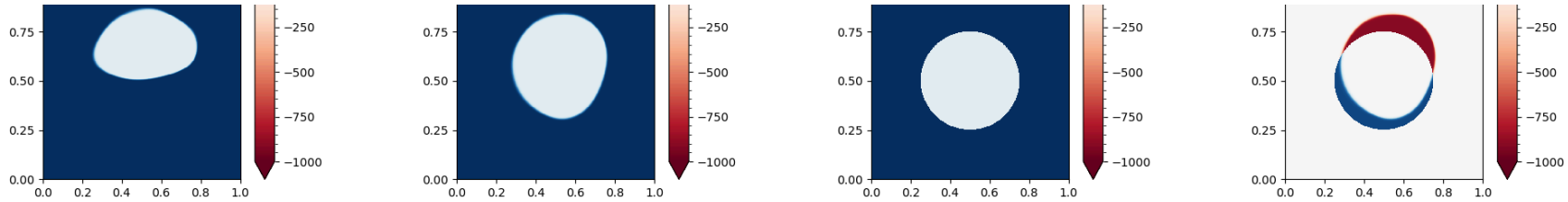

Predicted density shape at t = 0.8


Predicted density shape at t = 0.9


Predicted density shape at t = 1.0


Actual Initial Density where t=0 for comparing movement


Difference of Density Between t=0 and t=1



In [ ]: