Exercise_7

June 16, 2025

#Advanced Deep Learning for Physics (IN2298) #Exercise 7: Optimal Path Simulator

[110]:
```
# Install PhyFlow and Import libraries
%%capture
!pip install --quiet phiflow;
!pip install nbconvert;
!apt-get install texlive texlive-xetex texlive-latex-extra pandoc;
from google.colab import drive
drive.mount("/content/drive");
from phi.torch.flow import *
```

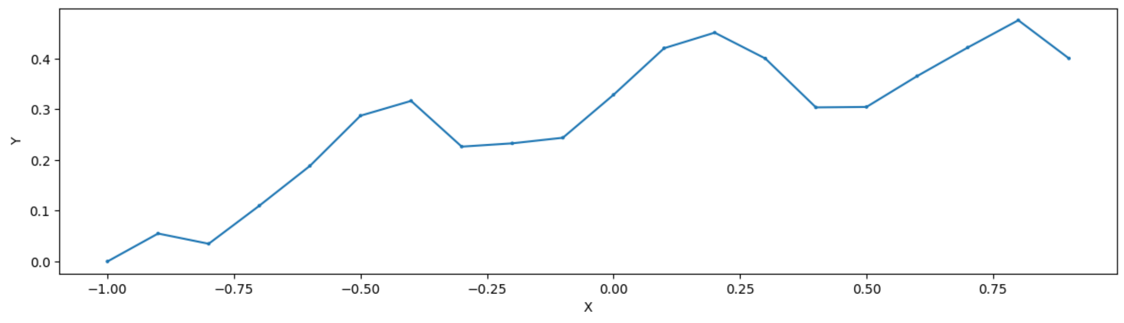
0.1 (1) Optimal Path: Physics-Based Optimization

[111]:
```
# Define system
def single_potential(x,y):
  return (x**2+y**2-1)**2

def step(x,y,dx,Fi):
  return x+dx,y+Fi

def potential(p: tensor):
  pot=0
  for i in range(N):
    pot += single_potential(p.time[i][0],p.time[i][1])
  return pot

def loss(F: tensor,x0:float,y0:float):
  x,y=x0,y0
  loss =0
  for i in range(N):
    x,y = step(x,y,dx,F.time[i])
    loss += single_potential(x,y)
    xi,yi = x,y
  return math.sum(loss)
```

[112]:
```
#Initial conditions
x0=-1
```

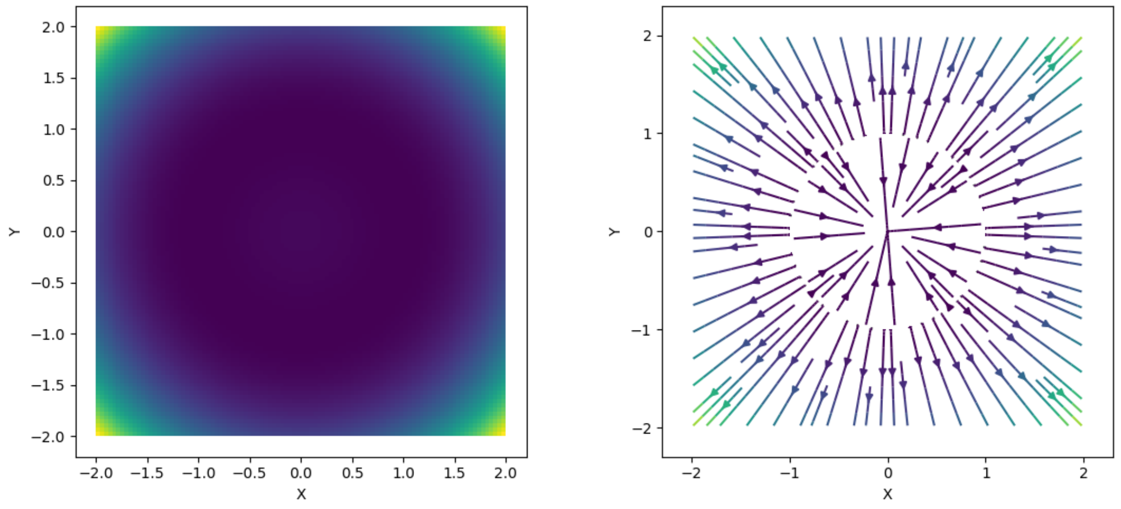
```
y0=0
N=20
dx=.1
v = math.zeros(channel(vector='x,y'),spatial(time=N))
# Hyperparameters
epochs = 500
lr =0.001
beta = 0.95
# Tensors
F = math.random_uniform(spatial(time=N),low=-.1,high=.1)
v = math.zeros(spatial(time=N))
pr = math.zeros(spatial(time=N-1),channel(vector='x,y'))
p0 = math.tensor([(-1,0)],spatial(time=1),channel(vector='x,y'))
p = math.concat([p0,pr],spatial('time'))
```

[113]:
```
def traj(x0: float,y0: float, F: tensor ,dx: float,N: int):
  p = []
  p.append(math.tensor([(x0,y0)],spatial(time=1),channel(vector='x,y')))
  x = x0
  y = y0
  for i in range(N-1):
    xi,yi = step(x,y,dx,F.time[i])
    v = math.vec(x=xi,y=yi)
    v_with_time = math.expand(v, spatial(time=1))
    p.append(v_with_time)
    x=xi
    y=yi

  return concat(p,spatial('time'))

p = traj(x0,y0,F,dx,N)
show(p)
```



[114]:
```
landscape = CenteredGrid(potential, x=100, y=100, bounds=Box(x=(-2, 2), y=(-2,␣
↪2)))
```

```
pot_grad = math.gradient(potential, wrt='p', get_output=False)
show([landscape,landscape.with_values(pot_grad)*.0001],show_color_bar=False)
```

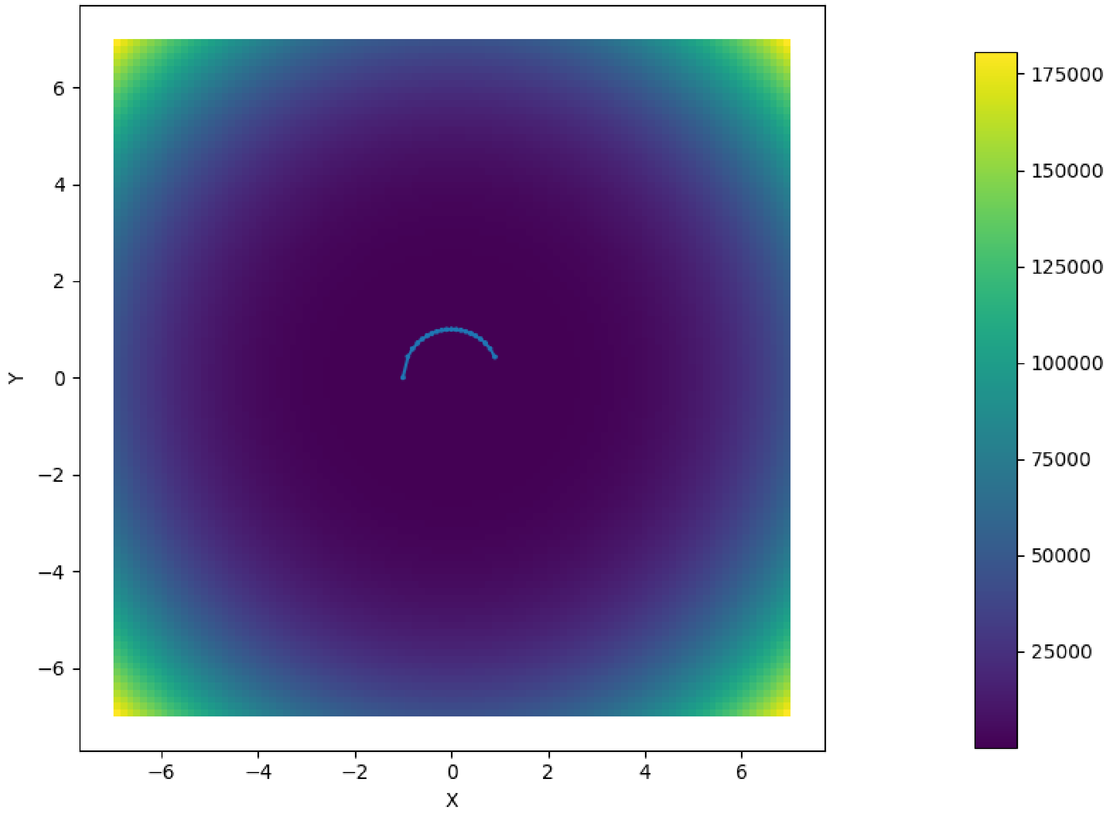


```
[115]: def gradient_descent_step(F,v,x0,y0):
         loss_fn = lambda F: loss(F,x0, y0)
         grad = math.gradient(loss_fn, 'F', get_output=False)
         # Clip gradients to avoid explosion
         v = beta * v - lr*grad(F)
         return F + v,v
       def gradient_descent(F,v,x0,y0,steps):
         for i in range(steps):
           F,v = gradient_descent_step(F,v,x0,y0)
         return F

       print(loss(F,x0,y0))
       F = gradient_descent(F,v,x0,y0,500)
       print(loss(F,x0,y0))

       landscape = CenteredGrid(potential, x=100, y=100, bounds=Box(x=(-7, 7), y=(-7,␣
        ↪7)))
       p = traj(x0,y0,F,dx,N)
       show([landscape,p],overlay='list',size=[10,6])
```

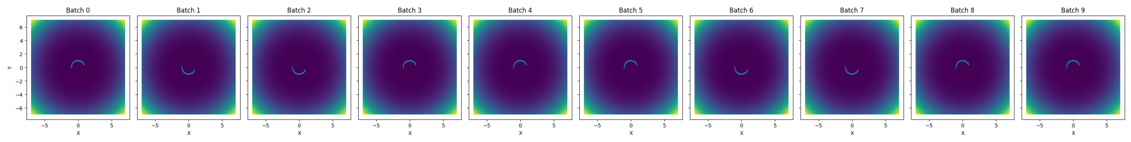
```
8.173978
0.00021289331
```

```
/usr/local/lib/python3.11/dist-
packages/phi/vis/_matplotlib/_matplotlib_plots.py:167: UserWarning: This figure
includes Axes that are not compatible with tight_layout, so results might be
incorrect.
```

```
  plt.tight_layout()  # because subplot titles can be added after figure
creation
```



[116]:
```
# Let's do that several times for different initila values
p_list=[]
for i in range(10):
  F=math.random_uniform(spatial(time=N),low=-.1,high=.1)
  v = math.zeros(spatial(time=N))
  F = gradient_descent(F,v,x0,y0,500)
  p = traj(x0,y0,F,dx,N)
  best_p_batched = math.expand(p, batch(batch=1))
  p_list.append(best_p_batched)

t = math.concat(p_list,batch('batch'))
show([landscape,t],size=[30,4],overlay='list',show_color_bar=False)
```



As we can notice some solutions perfor a clockwise or counterclockwise trajectory due to the symmetry of the potential lanscape

0.2 (2) Optimal Path: Supervised Learning Approach

[117]:
```
# Creation of the Dataset
steps = 500
samples=100
x0=-1
# Creation of 100 vector (-1,eps)
p0_list = []
F_list = []
for i in range(samples):
  y0 = math.random_uniform(low=-.01,high=.01) # Added math. prefix
  # Reinitialize F and v for each new trajectory
  F0 = math.random_uniform(spatial(time=N),low=-.1,high=.1)
  v = math.zeros(spatial(time=N))
  # Use iterate to perform gradient descent for 500 steps
  F=F0
  F = gradient_descent(F,v,x0,y0,steps)
  p0 = math.vec(x=x0,y=y0)
  # Expand dimension
  F = math.expand(F,batch(batch=1))
  p0 = math.expand(p0,batch(batch=1))
  # Append to list
  F_list.append(F)
  p0_list.append(p0)

#concat the tensors
p0_data = math.concat(p0_list,batch('batch'))
F_data = math.concat(F_list,batch('batch'))
```

0.2.1 Training

[118]:
```
# NN and optimizer
net = dense_net(2, 20, layers=[10,1], activation='ReLU')
optimizer = adam(net, 0.01)
```

[119]:
```
#@math.jit_compile
def loss_function(p0,F):
  F_pred = math.native_call(net,p0)
  F_pred = math.rename_dims(F_pred,channel('vector'),spatial('time'))
  diff = F-F_pred
  return math.l2_loss(diff)
```

```
[120]: #Training
       for i in range(1500):
         loss_value = update_weights(net,optimizer,loss_function,p0_data,F_data)
         if i % 100 == 0: print(f'loss: {loss_value}')
```

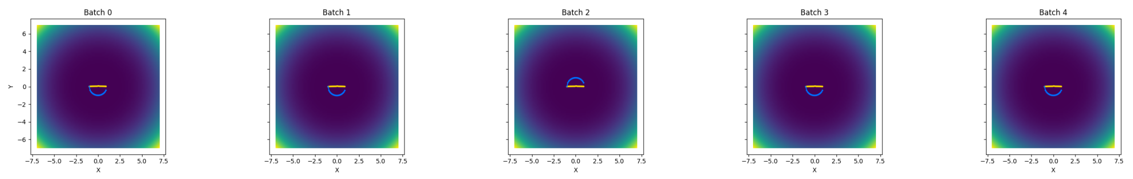
```
loss: (batch =100) 0.241 ± 0.275 (2e-01…2e+00)
loss: (batch =100) 0.241 ± 0.270 (2e-01…2e+00)
loss: (batch =100) 0.241 ± 0.270 (2e-01…2e+00)
loss: (batch =100) 0.241 ± 0.270 (2e-01…2e+00)
loss: (batch =100) 0.241 ± 0.270 (2e-01…2e+00)
loss: (batch =100) 0.241 ± 0.270 (2e-01…2e+00)
loss: (batch =100) 0.241 ± 0.270 (2e-01…2e+00)
loss: (batch =100) 0.241 ± 0.270 (2e-01…2e+00)
loss: (batch =100) 0.241 ± 0.270 (2e-01…2e+00)
loss: (batch =100) 0.241 ± 0.270 (2e-01…2e+00)
loss: (batch =100) 0.241 ± 0.270 (2e-01…2e+00)
loss: (batch =100) 0.241 ± 0.270 (2e-01…2e+00)
loss: (batch =100) 0.241 ± 0.270 (2e-01…2e+00)
loss: (batch =100) 0.241 ± 0.270 (2e-01…2e+00)
loss: (batch =100) 0.241 ± 0.270 (2e-01…2e+00)
```

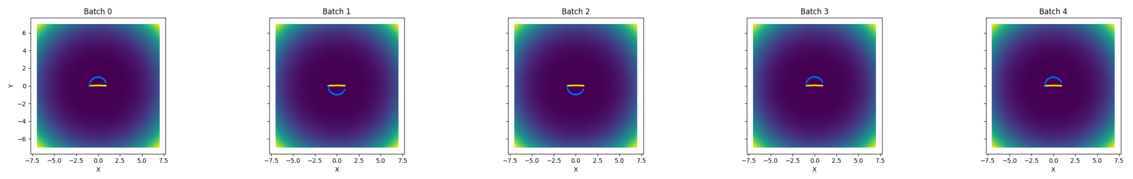
0.2.2 Evaluation

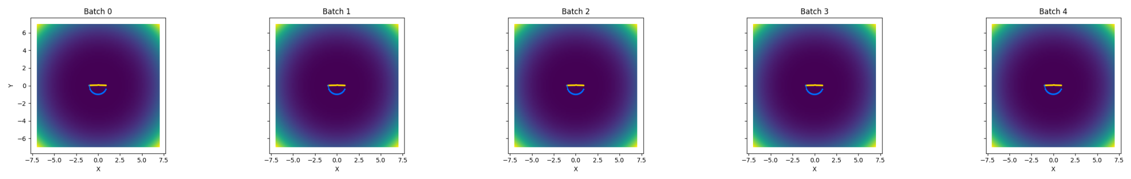
```
[121]: steps = 500
       samples=25
       x0=-1
       # Creation of 25 vector (-1,eps)
       p0_list = []
       F_list = []
       for i in range(samples):
         y0 = math.random_uniform(low=-.01,high=.01) # Added math. prefix
         # Reinitialize F and v for each new trajectory
         F0 = math.random_uniform(spatial(time=N),low=-.1,high=.1)
         v = math.zeros(spatial(time=N))
         # Use iterate to perform gradient descent for 500 steps
         F=F0
         F = gradient_descent(F,v,x0,y0,steps)
         p0 = math.vec(x=x0,y=y0)
         # Expand dimension
         F = math.expand(F,batch(batch=1))
         p0 = math.expand(p0,batch(batch=1))
         # Append to list
         F_list.append(F)
         p0_list.append(p0)

       #concat the tensors
       p0_eval = math.concat(p0_list,batch('batch'))
```

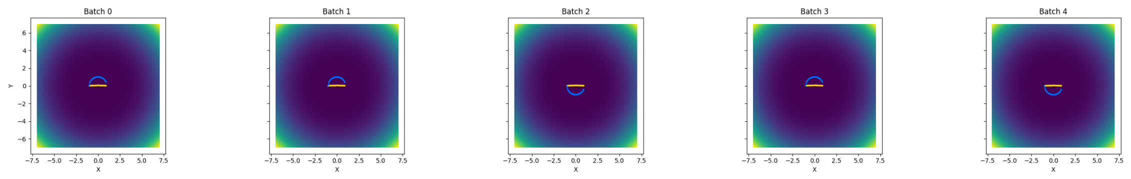
```
F_eval = math.concat(F_list,batch('batch'))
```

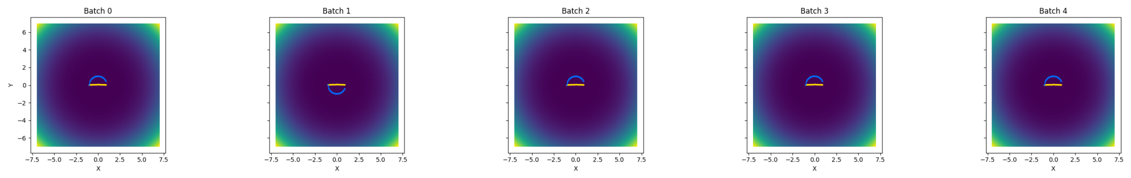
[122]:
```
#Predict the Forces
F_pred = math.native_call(net,p0_eval)
F_pred = math.rename_dims(F_pred,channel('vector'),spatial('time'))
p_list=[]
p_eval_list=[]
for i in range(25):
  y0 = math.random_uniform(low=-.01,high=.01)
  p = traj(x0,y0,F_pred.batch[i],dx,N)
  p_eval = traj(x0,y0,F_eval.batch[i],dx,N)
  best_p_batched = math.expand(p, batch(batch=1))
  best_p_eval_batched = math.expand(p_eval, batch(batch=1))
  p_list.append(best_p_batched)
  p_eval_list.append(best_p_eval_batched)

#Plot the results
t = math.concat(p_list,batch('batch'))
t_eval = math.concat(p_eval_list,batch('batch'))
color = [None,wrap(['#fcd700'],instance(t)),wrap(['#006dfc'],instance(t_eval))]


show([landscape,t.batch[0:5],t_eval.batch[0:
 ↪5]],size=[30,4],overlay='list',color=color,show_color_bar=False)
show([landscape,t.batch[5:10],t_eval.batch[5:
 ↪10]],size=[30,4],overlay='list',color=color,show_color_bar=False)
show([landscape,t.batch[10:15],t_eval.batch[10:
 ↪15]],size=[30,4],overlay='list',color=color,show_color_bar=False)
show([landscape,t.batch[15:20],t_eval.batch[15:
 ↪20]],size=[30,4],overlay='list',color=color,show_color_bar=False)
show([landscape,t.batch[20:25],t_eval.batch[20:
 ↪25]],size=[30,4],overlay='list',color=color,show_color_bar=False)
```











0.3 (3) Optimal Path: Differentiable Physics Training

[123]:
```
# NN and optimizer
net_dp = dense_net(2, 20, layers=[10,1], activation='ReLU')
optimizer = adam(net_dp, 0.01)
```

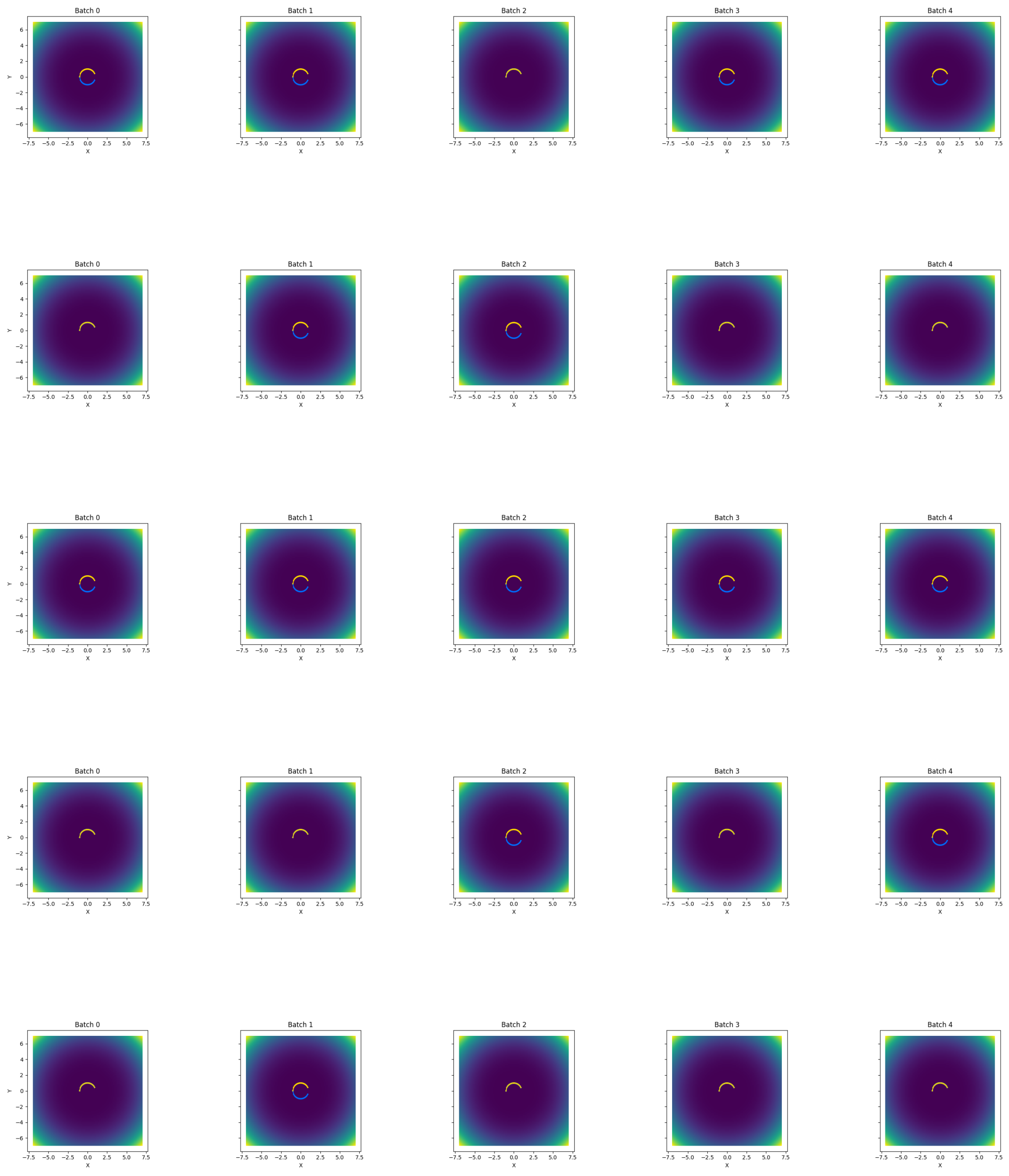
[124]:
```
def physics_loss(p0,net=net_dp):
  # Simulation end to end
  F = math.native_call(net, p0)
  F = math.rename_dims(F, channel('vector'), spatial('time'))
  p = traj(p0.vector['x'],p0.vector['y'],F,dx,N)
  energy = potential(p)
  return energy
```

[125]:
```
#Training
for i in range(1500):
  loss_value = update_weights(net_dp,optimizer,physics_loss,p0_data)
  if i % 100 == 0: print(f'loss: {loss_value}')
```

```
loss: (batch =100) 1.07e+01 ± 8.2e-04 (1e+01…1e+01)
loss: (batch =100) 0.031 ± 0.002 (3e-02…4e-02)
loss: (batch =100) 0.010 ± 0.002 (9e-03…2e-02)
```

```
loss: (batch =100) 0.006 ± 0.002 (4e-03…1e-02)
loss: (batch =100) 0.004 ± 0.002 (2e-03…8e-03)
loss: (batch =100) 0.003 ± 0.002 (8e-04…6e-03)
loss: (batch =100) 0.002 ± 0.002 (3e-04…6e-03)
loss: (batch =100) 0.002 ± 0.002 (1e-04…6e-03)
loss: (batch =100) 0.002 ± 0.002 (3e-05…5e-03)
loss: (batch =100) 0.002 ± 0.002 (7e-06…5e-03)
loss: (batch =100) 0.002 ± 0.002 (2e-06…5e-03)
loss: (batch =100) 0.002 ± 0.002 (4e-07…5e-03)
loss: (batch =100) 0.002 ± 0.002 (2e-07…5e-03)
loss: (batch =100) 0.002 ± 0.002 (2e-07…5e-03)
loss: (batch =100) 0.002 ± 0.002 (2e-07…5e-03)
```

[126]:
```
#Predict the Forces
F_pred = math.native_call(net_dp,p0_eval)
F_pred = math.rename_dims(F_pred,channel('vector'),spatial('time'))
p_list=[]
p_eval_list=[]
for i in range(25):
  y0 = math.random_uniform(low=-.01,high=.01)
  p = traj(x0,y0,F_pred.batch[i],dx,N)
  p_eval = traj(x0,y0,F_eval.batch[i],dx,N)
  best_p_batched = math.expand(p, batch(batch=1))
  best_p_eval_batched = math.expand(p_eval, batch(batch=1))
  p_list.append(best_p_batched)
  p_eval_list.append(best_p_eval_batched)

#Plot the results
t = math.concat(p_list,batch('batch'))
t_eval = math.concat(p_eval_list,batch('batch'))
color = [None,wrap(['#006dfc'],instance(t_eval)),wrap(['#fcd700'],instance(t))]


show([landscape,t_eval.batch[0:
 ↪5]],size=[30,4],overlay='list',color=color,show_color_bar=False)
show([landscape,t_eval.batch[5:10],t.batch[5:
 ↪10]],size=[30,4],overlay='list',color=color,show_color_bar=False)
show([landscape,t_eval.batch[10:15],t.batch[10:
 ↪15]],size=[30,4],overlay='list',color=color,show_color_bar=False)
show([landscape,t_eval.batch[15:20],t.batch[15:
 ↪20]],size=[30,4],overlay='list',color=color,show_color_bar=False)
show([landscape,t_eval.batch[20:25],t.batch[20:
 ↪25]],size=[30,4],overlay='list',color=color,show_color_bar=False)
```



As we can observe, the trajectories are significantly improved using this approach. The issue with supervised learning is that the mean value of the solution tends to be a straight line, leading the model to learn and predict a flat trajectory. In contrast, with differentiable physics training, the model learns to minimize the potential by selecting appropriate forces.

The difference between the predicted and evaluated trajectories lies in the fact that, at times, an upper arc is predicted as a lower one or vice versa. Nevertheless, the predictions are still valid, as all the resulting forces produce trajectories with very low potential.

[128]:
```
%%capture
!jupyter nbconvert --to pdf --output /content/drive/MyDrive/Fisica/ADL4P/
 ↪Exercise_7.pdf /content/drive/MyDrive/Fisica/ADL4P/Exercise_7.ipynb
```