```
#             break
#         elif sample_balance >= 200:  # stop if balance reaches $200
#             balance[si] = 200
#             break

# play out each game (vectorised) (ihat=2.009666286616056e-06)
game_diffs = 2*games - 1 # convert 0/1 to -1/1 to track winnings/cost
running_balances = 100 + game_diffs.cumsum(dim=1)
# set all values after a 0 to 0 (since player quits at $0)
if (running_balances==0).any():
    running_balances[(running_balances==0).nonzero(as_tuple=True)[0].item() + 1:
 ↪] = 0
balances = running_balances.max(dim=1).values  # if 200 crossed, will be the␣
 ↪maximum

f = pdf(p, games)
g = pdf(q, games)
samp = ((balances>=200)*f/g)

ihat = samp.mean()
print(ihat.item())
```

```
1.9842930575780995e-06
```

3 Problem 6

[ ]:
```
def plot_sgd_path(history, ax=None):
    mu = np.array(history[:, 0])
    sigma = np.array(history[:, 1])
    if ax is not None:
        p = ax
    else:
        p = plt

    p.plot(mu, sigma, linestyle='solid', color='blue', zorder=0)
    p.grid(True, which='both', linestyle='--')

    # show labels next to start and end scatter points
    p.scatter(mu[0], sigma[0], color='blue', label='Initial', zorder=1)
    p.text(mu[0], sigma[0], 'Start', verticalalignment='bottom',␣
 ↪horizontalalignment='right', c='blue', zorder=1)
    p.scatter(mu[-1], sigma[-1], color='red', label='Final', zorder=1)
    p.text(mu[-1], sigma[-1], 'End', verticalalignment='bottom',␣
 ↪horizontalalignment='right', c='red', zorder=1)

    # label axes
    plt.xlabel('$\mu$')
```

```
    plt.ylabel(r'$\tau$')
```

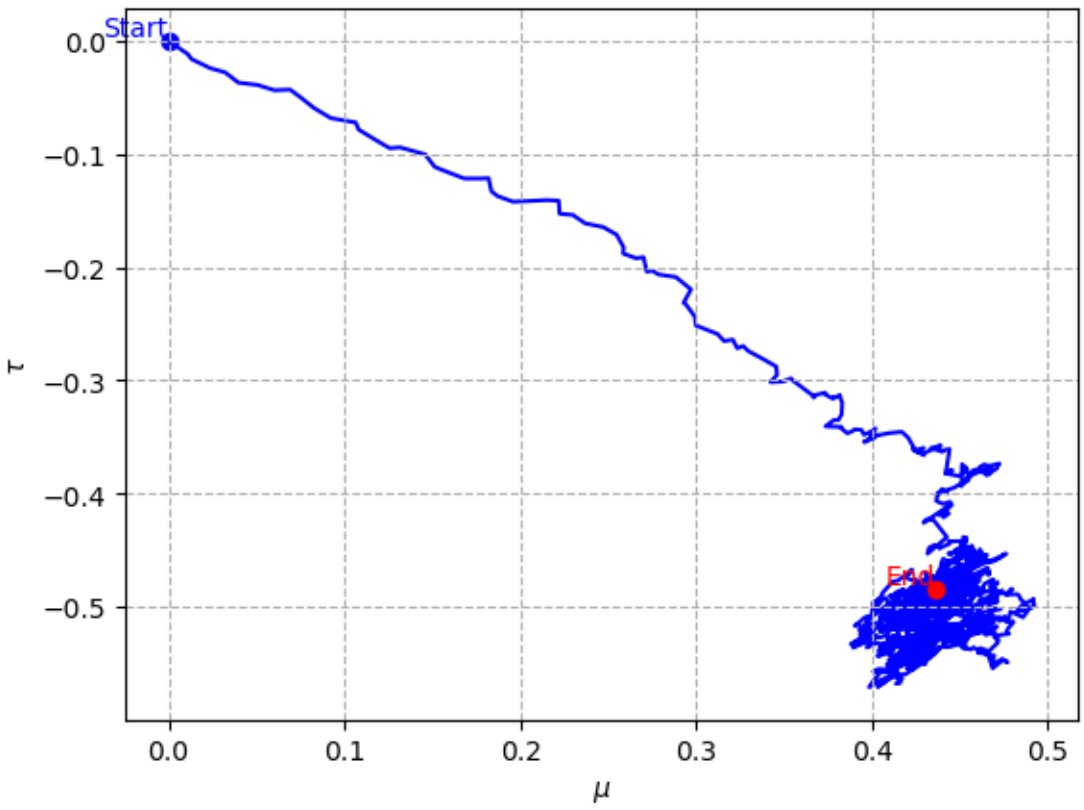
[ ]:
```
def evaluate(mu, tau, samples=5000):
    sigma = torch.exp(tau)
    X = torch.normal(mu, sigma, size=(samples,))
    return ((X*torch.sin(X)).sum() + 0.5 * (mu - 1)**2 + sigma - torch.
 ↪log(sigma)).item()
```

[ ]:
```
lr = 1e-2
B = 16
iterations = 1500
```

3.1 Part (a): log-derivative trick

[ ]:
```
torch.manual_seed(42)
mu = torch.zeros(1)
tau = torch.zeros(1)
history1 = torch.zeros((iterations+1, 2))

for itr in range(iterations):
    history1[itr] = torch.tensor([mu, tau])

    X = torch.normal(mu.item(), torch.exp(tau).item(), size=(B,))

    # SGD update
    grad_mu = (X * torch.sin(X) * (X - mu)/torch.exp(2*tau)).mean() + mu - 1
    # previously mistakenly used: torch.exp(2*tau + 1)
    grad_tau = (X * torch.sin(X) * (torch.pow(X - mu, 2)/torch.exp(2*tau) - 1)).
 ↪mean() + torch.exp(tau) - 1

    mu -= lr * grad_mu
    tau -= lr * grad_tau

history1[-1] = torch.tensor([mu, tau])

print(f"Optimal mu: {mu.item():.3f}\nOptimal sigma (tau): {torch.exp(tau).
 ↪item():.3f} ({tau.item():.3f})")
plot_sgd_path(history1)
```

```
Optimal mu: 0.437
Optimal sigma (tau): 0.616 (-0.485)
```

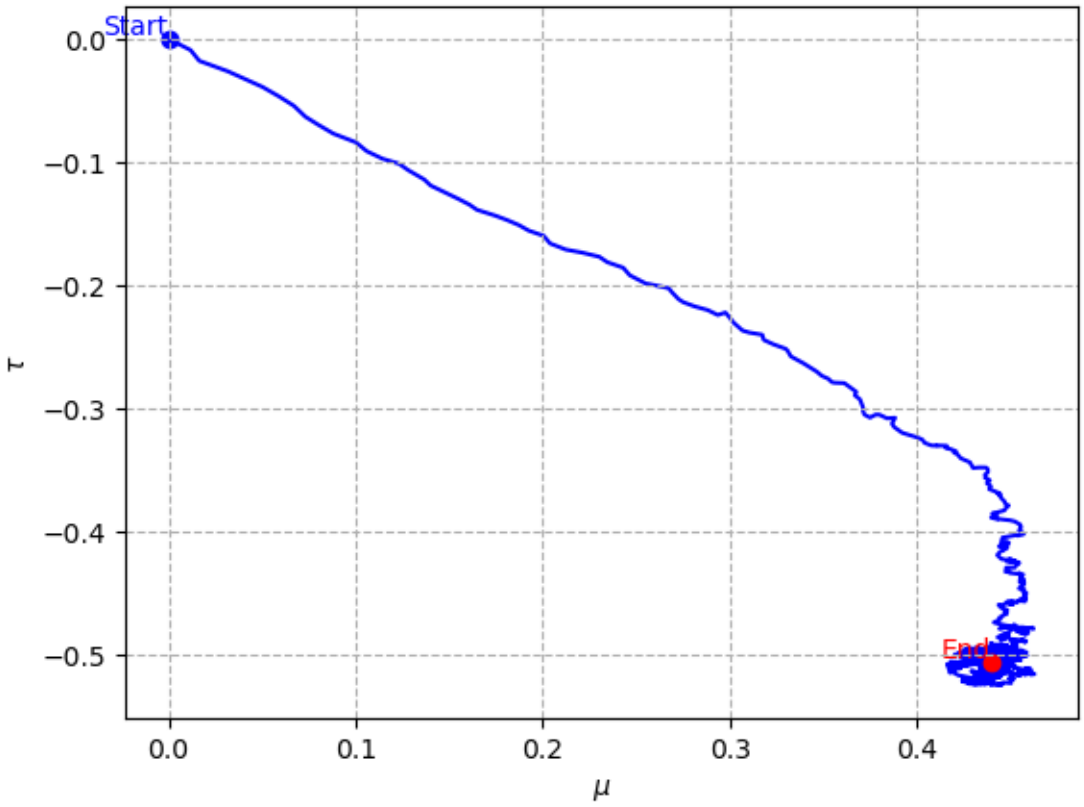


3.2 Part (b): reparameterisation trick

[ ]:
```
torch.manual_seed(42)
mu = torch.zeros(1)
tau = torch.zeros(1)
history2 = torch.zeros((iterations+1, 2))

for itr in range(iterations):
    history2[itr] = torch.tensor([mu, tau])

    Y = torch.normal(0, 1, size=(B,))

    # SGD update
    sigma = torch.exp(tau)
    Z = mu + sigma*Y

    grad_mu = (torch.sin(Z) + Z*torch.cos(Z)).mean() + mu - 1
    grad_tau = (sigma*Y*torch.sin(Z) + Z*sigma*Y*torch.cos(Z)).mean() + torch.
 ↪exp(tau) - 1
```

```
    mu -= lr * grad_mu
    tau -= lr * grad_tau

history2[-1] = torch.tensor([mu, tau])

print(f"Optimal mu: {mu.item():.3f}\nOptimal sigma (tau): {torch.exp(tau).
  ↪item():.3f} ({tau.item():.3f})")
plot_sgd_path(history2)
```

```
Optimal mu: 0.440
Optimal sigma (tau): 0.602 (-0.507)
```



3.3 Comparison plot

[ ]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4), sharey=True)

plot_sgd_path(history1, ax1)
ax1.set_title('Log-derivative Trick')

plot_sgd_path(history2, ax2)
ax2.set_title('Reparameterisation Trick')
```

```
# plt.tight_layout()
plt.show()
```