TP6 — Metropolis RWM/IMH (corrige)

Source : TP6/notebook_MH_student_corrige.ipynb — PDF compile (code execute).

Lab on IMH and RWM

In [1] :

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import scipy.integrate as spi
import metropolis_code_student as metropolis_code

# Apply Seaborn style
sns.set_theme(style="whitegrid")
```

In [2] :

```
# general useful function that you may use

def cummean(x,axjj=0):
"""
Compute the cumulative mean of the input array x
"""
lenx = len(x)
return np.cumsum(x,axis=axjj) / (np.arange(len(x)) + 1)

def next_pow_two(n):
i = 1
while i < n:
i = i << 1
return i

def autocorrelation_fft(x):
"""
Compute the autocorrelation of the input array x using Fourier Transform
"""
xp = np.fft.ifftshift((x - np.average(x))/np.std(x))
n, = xp.shape
xp = np.r_[xp[:n//2], np.zeros_like(xp), xp[n//2:]]
f = np.fft.fft(xp)
p = np.absolute(f)**2
pi = np.fft.ifft(p)
return np.abs(np.real(pi)[:n//2]/(np.arange(n//2)[::-1]+n//2))


def cumvar(x):
"""
Compute the estimate variance of samples of a Markov chain in x
"""
return 1.96*np.sum(autocorrelation_fft(x))/np.sqrt(1+np.arange(len(x)))
```

In [3] :

```
def potential_function(x):
return 0.2 * x**2 + 0.5*np.log(1+(x**2-2)**2)

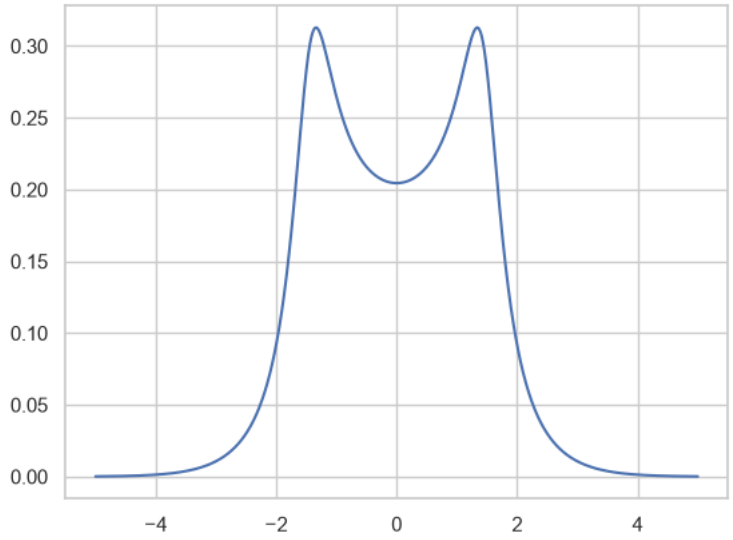

x_list = np.linspace(-5, 5, 1000)
y = np.exp(-potential_function(x_list))
normalizing_const = spi.quad(lambda x: np.exp(-potential_function(x)), -5, 5)[0]
y = y/normalizing_const # normalize
plt.plot(x_list, y)


test_fun = lambda x: (np.cos(x+1/2))

test_fun_inte = lambda x: test_fun(x)*np.exp(-potential_function(x))/normalizing_const

int_test_fun = spi.quad(test_fun_inte, -5, 5)[0]
print(int_test_fun)
```

```
0.33249428003665776
```




```
<Figure size 640x480 with 1 Axes>
```

In [4] :

```
n_samples = 10000
st_proposal_rwm = 2
st_proposal_ihm = 1.
x0 = np.zeros(1)

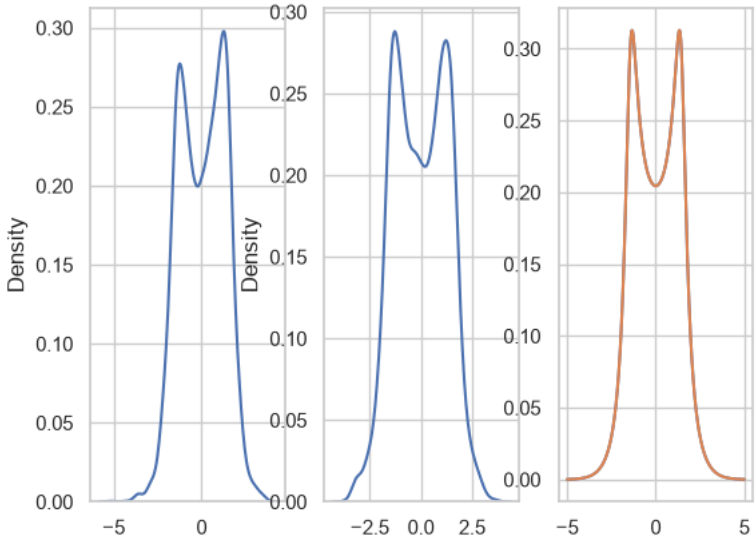
samples, acc_rwm = metropolis_code.random_walk_metropolis(n_samples, st_proposal_rwm, x0)
samples_ihm, acc_ihm = metropolis_code.independance_metropolis(n_samples, st_proposal_ihm,
x0)

#print(samples)
fig, ax = plt.subplots(1,3)
sns.kdeplot(samples[:,0], label='RWM', ax=ax[0])
sns.kdeplot(samples_ihm[:,0], label='IH', ax=ax[1])
ax[2] = plt.plot(x_list, y)
plt.plot(x_list, y)

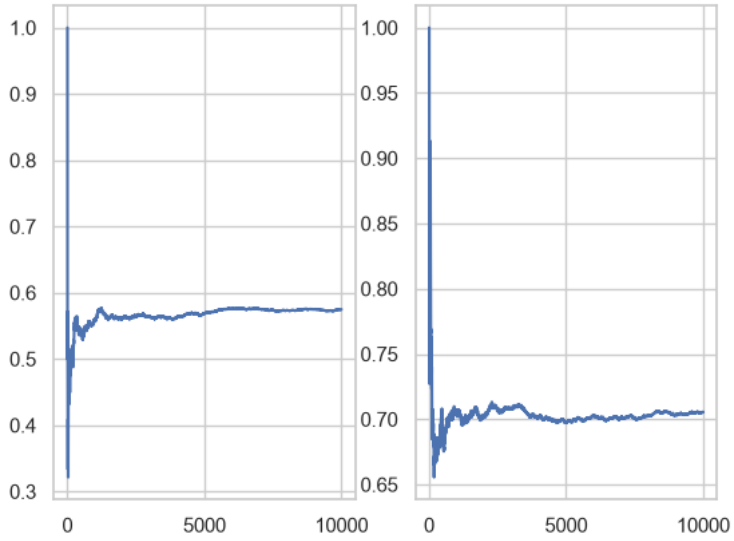
plt.show()


fig, ax = plt.subplots(1,2)
ax[0].plot(acc_rwm)
ax[1].plot(acc_ihm)

plt.show()
```




```
<Figure size 640x480 with 3 Axes>
```




```
<Figure size 640x480 with 2 Axes>
```

In [5] :

```
print(np.cumsum(test_fun(samples))/np.arange(1, n_samples+1))
estimator_rwm_list = cummean(test_fun(samples).flatten())
estimator_ihm_list = cummean(test_fun(samples_ihm).flatten())
var_est_rwm = cumvar(test_fun(samples).flatten())
var_est_ihm = cumvar(test_fun(samples_ihm).flatten())


data_rwm = pd.DataFrame({
'Index': range(len(estimator_rwm_list)),
'Cumulative Mean': estimator_rwm_list,
'Variance': np.sqrt(var_est_rwm)
})

data_ihm = pd.DataFrame({
'Index': range(len(estimator_ihm_list)),
'Cumulative Mean': estimator_ihm_list,
'Variance': np.sqrt(var_est_ihm)
})

# Plot the cumulative mean
fig, axi = plt.subplots(2, figsize=(10, 8))

sns.lineplot(x='Index', y='Cumulative Mean', data=data_rwm, ax=axi[0], label='RWM',
errorbar='sd')
sns.lineplot(x='Index', y='Cumulative Mean', data=data_ihm, ax=axi[1], label='IHM',
errorbar='sd')

ax = axi[0]
ax.plot(data_rwm['Index'], data_rwm['Cumulative Mean'], alpha=0.3)
ax.axhline(y=int_test_fun, color='r', linestyle='--', label='True Value')
ax.set_xlabel('Sample Index')
ax.set_ylabel('Cumulative Mean')
ax.set_title('Cumulative Mean with Empirical Variance for RWM')
ax.legend()

ax = axi[1]
ax.plot(data_ihm['Index'], data_ihm['Cumulative Mean'], alpha=0.3)
ax.axhline(y=int_test_fun, color='r', linestyle='--', label='True Value')
ax.set_xlabel('Sample Index')
ax.set_ylabel('Cumulative Mean')
ax.set_title('Cumulative Mean with Empirical Variance for IMH')
ax.legend()

plt.tight_layout()
plt.show()

# Plot the cumulative mean with empirical variance
fig, axi = plt.subplots(2, figsize=(10, 8))

sns.lineplot(x='Index', y='Cumulative Mean', data=data_rwm, ax=axi[0], label='RWM',
errorbar='sd')
sns.lineplot(x='Index', y='Cumulative Mean', data=data_ihm, ax=axi[1], label='IHM',
errorbar='sd')

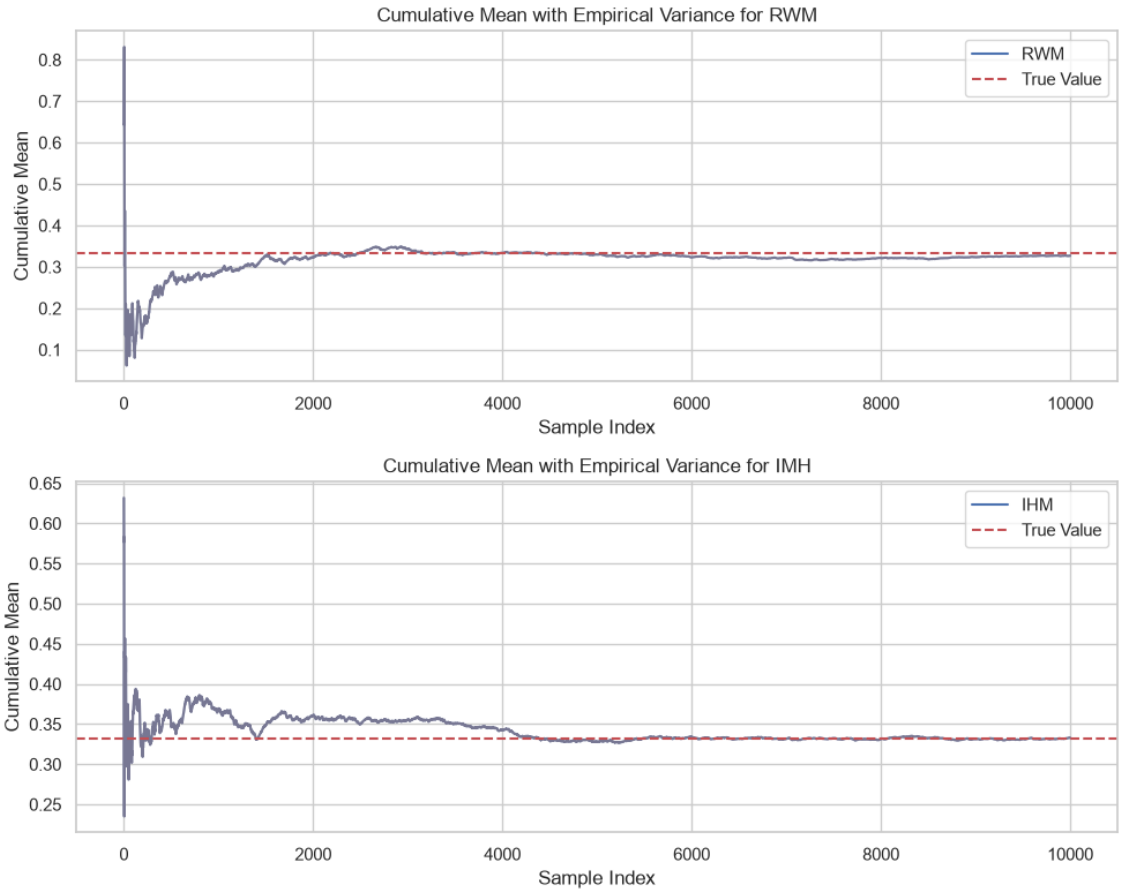
ax = axi[0]
ax.fill_between(data_rwm['Index'], data_rwm['Cumulative Mean'] - data_rwm['Variance'],
data_rwm['Cumulative Mean'] + data_rwm['Variance'], alpha=0.3)
ax.axhline(y=int_test_fun, color='r', linestyle='--', label='True Value')
ax.set_xlabel('Sample Index')
ax.set_ylabel('Cumulative Mean')
ax.set_title('Cumulative Mean with Empirical Variance for RWM')
ax.legend()

ax = axi[1]
```

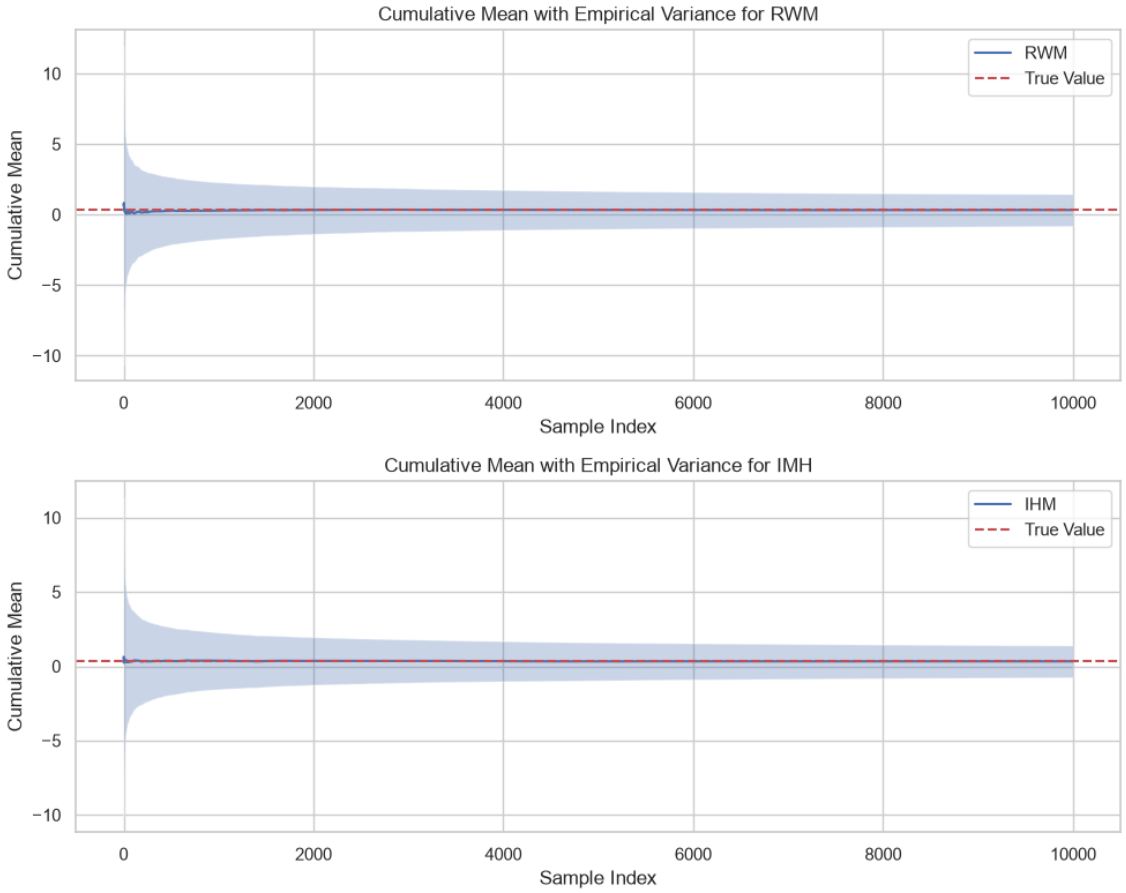
```
        ax.fill_between(data_ihm['Index'], data_ihm['Cumulative Mean'] - data_ihm['Variance'],
        data_ihm['Cumulative Mean'] + data_ihm['Variance'], alpha=0.3)
        ax.axhline(y=int_test_fun, color='r', linestyle='--', label='True Value')
        ax.set_xlabel('Sample Index')
        ax.set_ylabel('Cumulative Mean')
        ax.set_title('Cumulative Mean with Empirical Variance for IMH')
        ax.legend()

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

```
[0.64334844 0.64334844 0.74003017 ... 0.32683751 0.32682677 0.32681603]
```




```
<Figure size 1000x800 with 2 Axes>
```




```
<Figure size 1000x800 with 2 Axes>
```

In [6] :

```
        n_samples = 100000
        st_proposal_rwm = 0.1
        st_proposal_ihm = 1.25
        x0 = np.zeros(40)

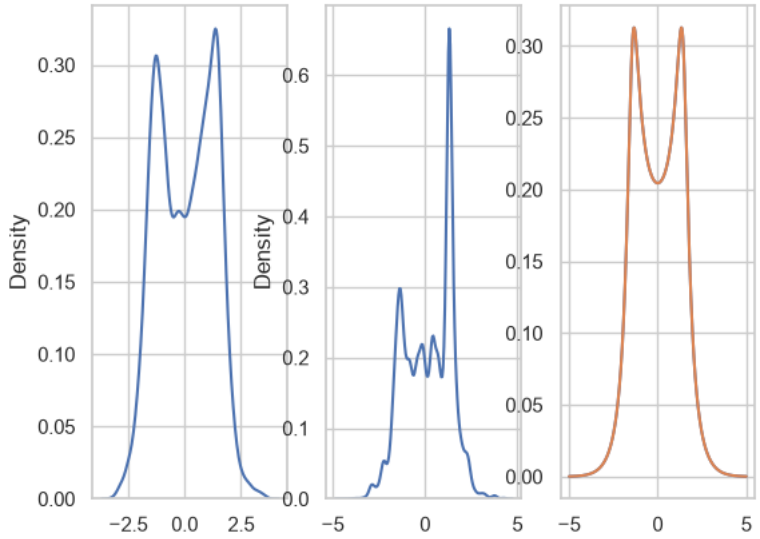
        samples, acc_rwm = metropolis_code.random_walk_metropolis(n_samples, st_proposal_rwm, x0)
        samples_ihm, acc_ihm = metropolis_code.independance_metropolis(n_samples, st_proposal_ihm,
        x0)

        #print(samples)
        fig, ax = plt.subplots(1,3)
        sns.kdeplot(samples[:,0], label='RWM', ax=ax[0])
        sns.kdeplot(samples_ihm[:,0], label='IH', ax=ax[1])
        ax[2] = plt.plot(x_list, y)
        plt.plot(x_list, y)

        plt.show()

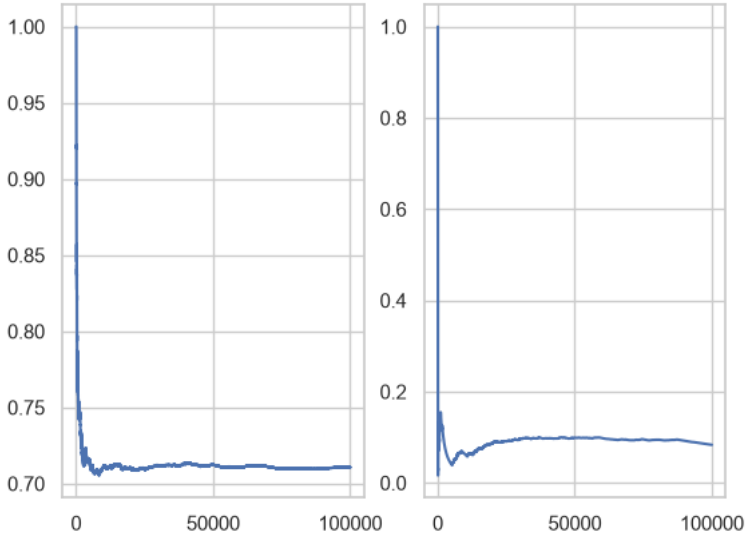
        fig, ax = plt.subplots(1,2)
        ax[0].plot(acc_rwm)
        ax[1].plot(acc_ihm)
        print(acc_ihm[-1])

        plt.show()
```




```
<Figure size 640x480 with 3 Axes>
```

```
0.08346
```

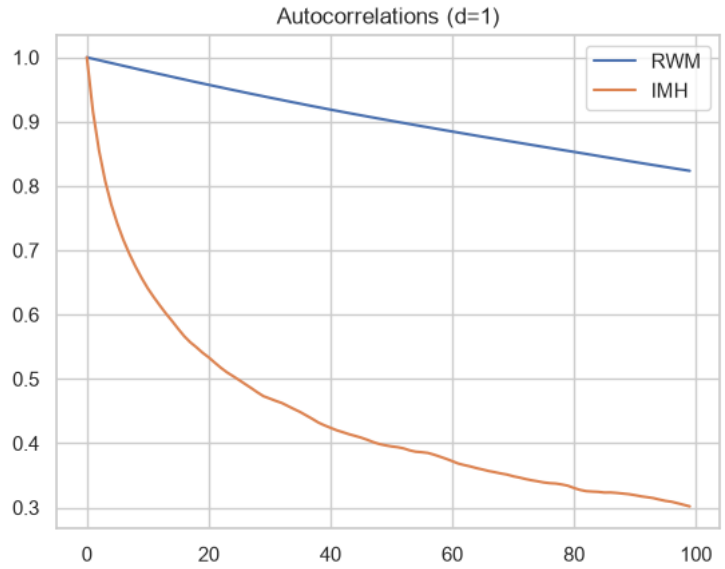



```
<Figure size 640x480 with 2 Axes>
```

In [7] :

```
        # Diagnostic : autocorrelations, ESS et taux d'acceptation (1D)
        for name, samp, acc in [("RWM", samples, acc_rwm), ("IMH", samples_ihm, acc_ihm)]:
        z = samp[:, 0]
        ac = autocorrelation_fft(z)
        ess = len(z) / (1 + 2 * np.sum(np.abs(ac[1:200])))
        print(f"{name} : accept final={acc[-1]:.3f}, ESS~{ess:.0f}/{len(z)},
        moyenne={z.mean():.3f}")
        plt.figure()
        plt.plot(autocorrelation_fft(samples[:,0])[:100], label="RWM")
        plt.plot(autocorrelation_fft(samples_ihm[:,0])[:100], label="IMH")
        plt.legend(); plt.grid(True); plt.title("Autocorrelations (d=1)"); plt.show()
        print("Conclusion 1D : les deux echantillonnent bien la cible bimodale ; IMH N(0,1)
        efficace ici.")
```

```
RWM : accept final=0.711, ESS~303/100000, moyenne=0.051
IMH : accept final=0.083, ESS~712/100000, moyenne=0.183
```




```
<Figure size 640x480 with 1 Axes>
```

```
Conclusion 1D : les deux echantillonnent bien la cible bimodale ; IMH N(0,1) efficace ici.
```

In [8] :

```
        # Conclusion grande dimension (d=40) : RWM petit pas (0.1) vs IMH
        # Avec d=40, IMH N(0,sd^2) ne couvre plus la cible (poids degenere, accept ~0) :
        # la chaine reste bloquee (voir acc_ihm plate). RWM explore lentement mais surement.
        print(f"RWM d=40 accept final={acc_rwm[-1]:.3f} ; IMH d=40 accept
        final={acc_ihm[-1]:.4f}")
        print("Conclusion : en grande dimension, l'IMH independant simple s'effondre (fleau de la
        dimension) ;")
        print("il faut un pas RWM calibre (~0.23 d'acceptation optimale) ou des methodes adaptees
        (MALA/HMC).")
```

```
RWM d=40 accept final=0.711 ; IMH d=40 accept final=0.0835
Conclusion : en grande dimension, l'IMH independant simple s'effondre (fleau de la dimension) ;
il faut un pas RWM calibre (~0.23 d'acceptation optimale) ou des methodes adaptees (MALA/HMC).
```