CompPhys_HW5

November 30, 2021

0.1 Exercise 5

Keito Watanabe (s6kewata), Haveesh Singirikonda (s6gusing)

Problem 2 (MH with Gaussian model)

See (attempts in) Problem 1 in end of PDF

[1]:
```python
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```python
def H(a,U):
    '''Iterates from 1 to N-1 with array of length N'''
    ham = 0

    for i in range(1, len(U)):

        ham = ham + (U[i]-U[i-1])**2

    return ham/a
```

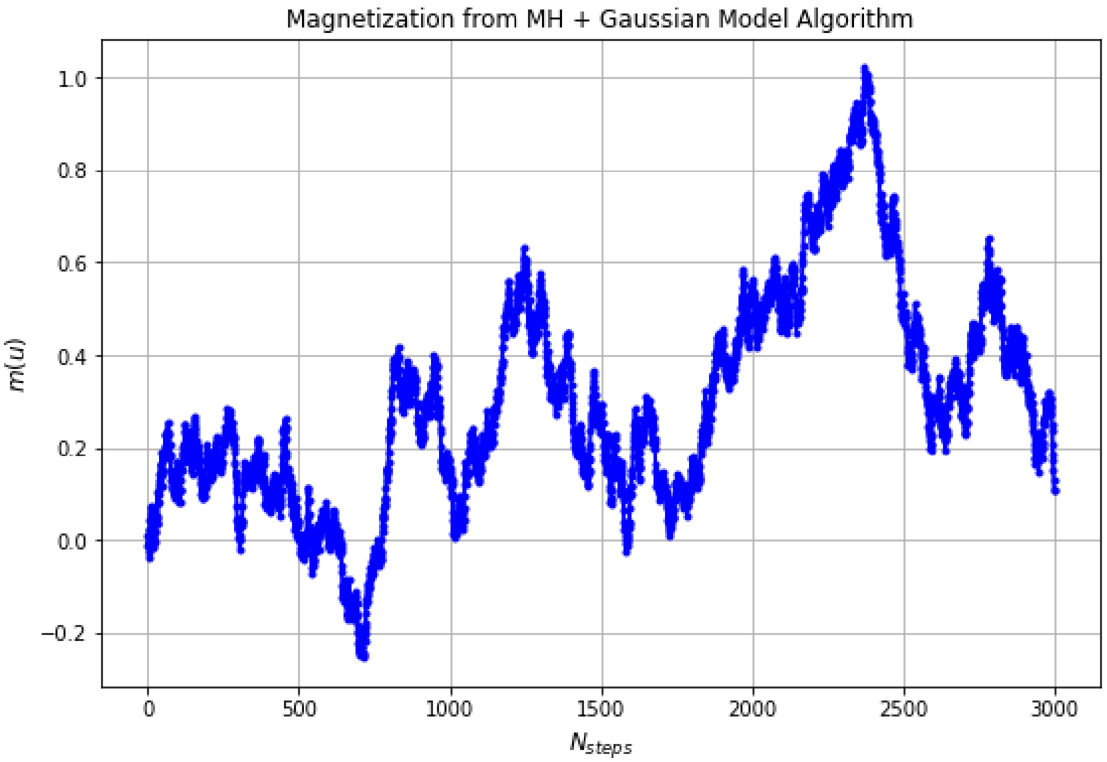
[3]:
```python
def MH(a,N,du, Ns):
    '''Metropolis-Hastings '''
    mag_arr = np.zeros(Ns)
    mag2_arr = np.zeros(Ns)
    en_arr = np.zeros(Ns)

    # take  N+1 array since we have points from 0 to N
    U_0 = np.zeros(N+1)

    for n in range(Ns):

        U_0[0] = 0
        U_0[N] = 0

        x = np.random.randint(1,N-1)

        r = np.random.uniform(-1,1)

        U_new = U_0
        U_new[x] = U_0[x] + r*du

        dH = H(a,U_new) - H(a,U_0)

        dP = np.exp(-dH)

        r_MH = np.random.uniform(0,1)

        if dP > 1:
            U_0 = U_new

        if dP > r_MH:
            U_0 = U_new

        # evaluate observables here
        mag_arr[n] = (np.sum(U_0[1:-2]) / N)
        mag2_arr[n] = (np.sum(U_0[1:-2]) / N)**2.
        en_arr[n] = H(a,U_0)

    return mag_arr, mag2_arr, en_arr
```

[14]:

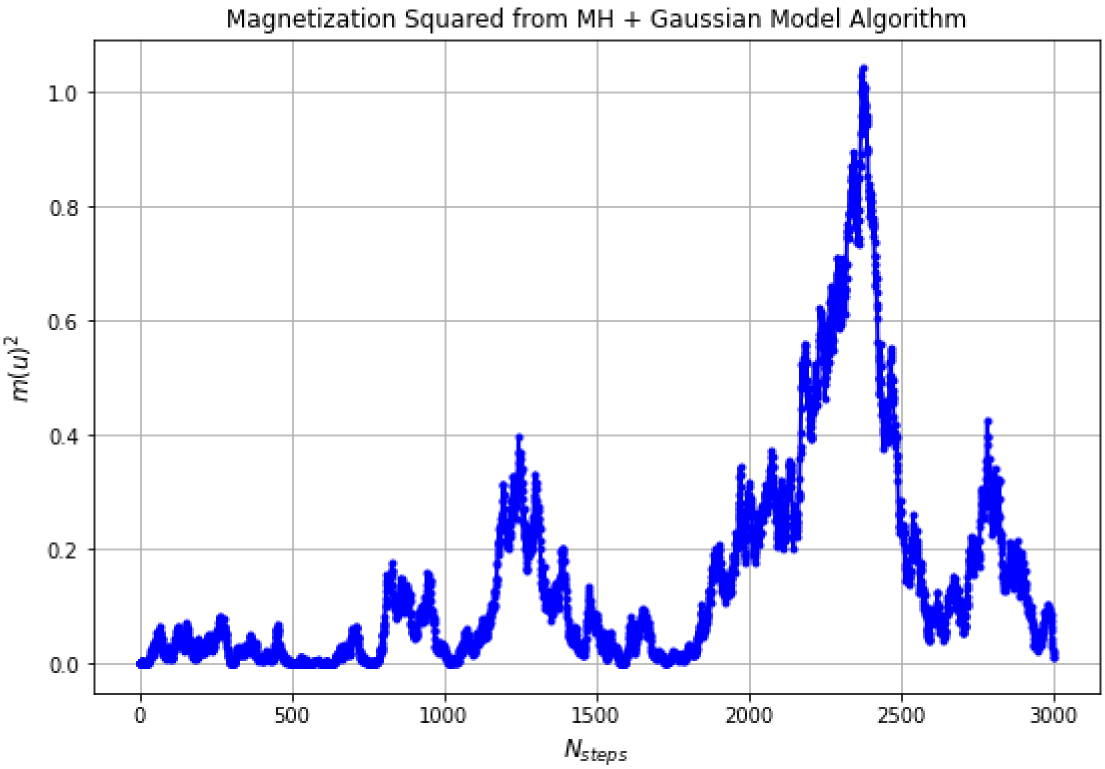
```
N = 64
a = 1
d = 2.
Ns = 3000

mag_arr, mag2_arr, en_arr = MH(a,N,d, Ns)
```

[15]:

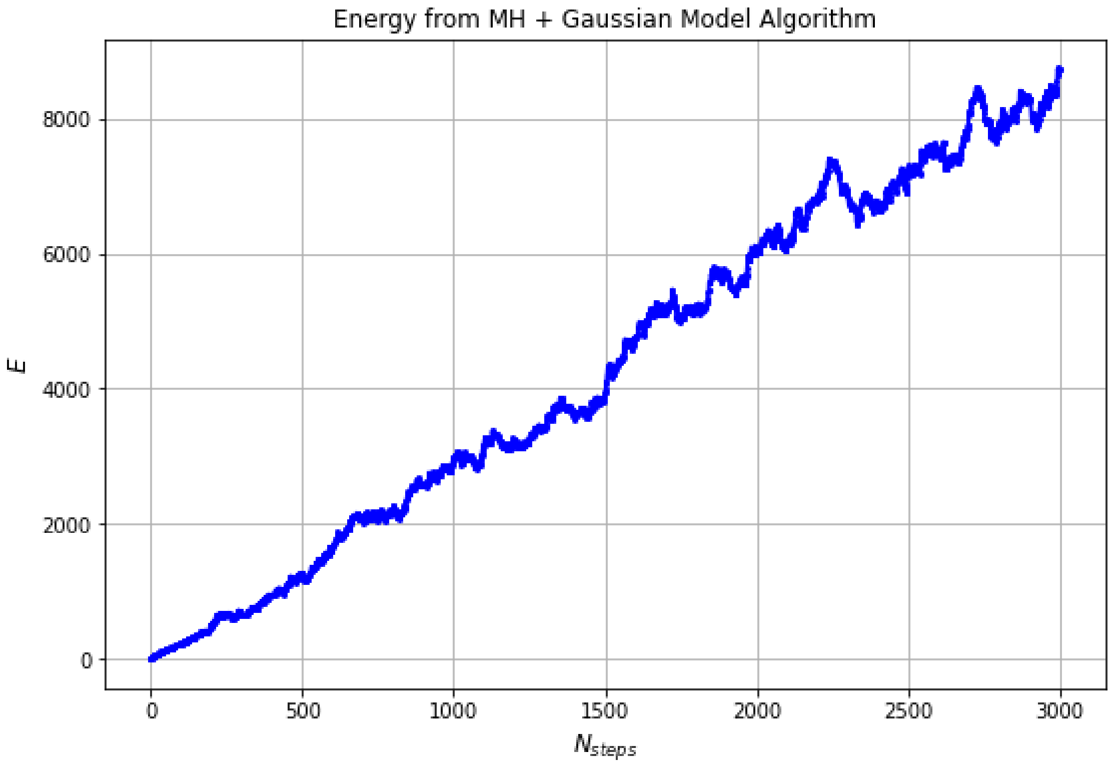
```
fig, ax = plt.subplots(figsize=(9,6))
ax.plot(mag_arr, marker="o", ms=3.0, lw=2.0, color="b", label="$m$")

ax.set_xlabel(r"$N_{{steps}}$", fontsize=12)
ax.set_ylabel(r"$m(u)$", fontsize=12)
ax.set_title("Magnetization from MH + Gaussian Model Algorithm")

ax.grid()
```



[16]:
```
fig, ax = plt.subplots(figsize=(9,6))
ax.plot(mag2_arr, marker="o", ms=3.0, lw=2.0, color="b", label="$m^2$")
# n_arr = np.arange(N)
# ax.plot((n_arr-1)/n_arr**(N-1))

ax.set_xlabel(r"$N_{{steps}}$", fontsize=12)
ax.set_ylabel(r"$m(u)^2$", fontsize=12)
ax.set_title("Magnetization Squared from MH + Gaussian Model Algorithm")

ax.grid()
```



[17]:
```
fig, ax = plt.subplots(figsize=(9,6))
ax.plot(en_arr, marker="o", ms=2.0, lw=2.0, color="b", label="$E$")

ax.set_xlabel(r"$N_{{steps}}$", fontsize=12)
ax.set_ylabel(r"$E$", fontsize=12)
ax.set_title("Energy from MH + Gaussian Model Algorithm")

ax.grid()
```



Some questions: Why is the magnetization varying so much with each iteration? Why is the energy increasing?

Problem 4 (Coarsening)

Refer to the end of the PDF for Q3

[ ]:

```
def H_new(u,a,phi):
    '''Generalized Hamiltonian'''

    ham1 = 0
    ham2 = 0

    for i in range(len(u)-1):

        ham1 = ham1 + (u[i+1]-u[i])**2
        ham2 = ham2 + phi[i]*u[i]

    return ham1/a + a*ham2

def coarse(u,a,phi):
    '''Restriction function'''

    Nu = len(u)

    Nu2 = int(Nu//2)
    u2 = np.ones(Nu2)
    phi2 = np.ones(Nu2)

    for i in range(1,Nu2-1):

        u2[i] = u[2*i]

        phi2[i] = 0.5*(0.5*phi[int(2*i-1)]+phi[int(2*i)]+0.5*phi[int(2*i+1)])

    phi2[0] = 0.5*(phi[0]+0.5*phi[1])
    u2[0] = 0
    u2[Nu2-1] = 0
    phi2[Nu2-1] = 0.5*(0.5*phi[2*Nu2-2] + phi[2*Nu2-1])

    return u2,2*a,phi2

def uncoarse(u,a,u_in):
    '''Prolongation function'''
    Nu = len(u)
    Nu2 = 2*Nu
    u_half = np.ones(Nu2)

    # zeroth index and Nth index already defined to be zero
    for i in range(1, Nu2-1):
        if i%2==0:
            # there is a bug in here that we cannot solve ;(
            u_half[i] =  u_in[i] + u[i//2]

        else:
            u_half[i] =  u_in[i] + 0.5*(u[(i-1)//2] + u[(i+1)//2])

    # set Dirichlet BC again
    u_half[0] = 0
    u_half[Nu2-1] = 0

#     u_half[Nu2-1] = u_in[Nu2-1] + u[Nu2//2-1]

    return u_half, a/2
```

[ ]:

```
def MH_new(U_0,a,du,Ns,phi):

    N = len(U_0)
    U_s = np.ones((Ns,N))

    for n in range(Ns):
```

```python
        # set BC by Dirichlet BC
        U_0[0] = 0
        U_0[N-1] = 0

        x = np.random.randint(1,N-1)

        r = np.random.uniform(-1,1)

        U_new = U_0

        U_new[x] = U_0[x] + r*du

        dH = H_new(U_new,a,phi) - H_new(U_0,a,phi)

        dP = np.exp(-dH)

        r_MH = np.random.uniform(0,1)

        if dP>1:
            U_0 = U_new

        if dP > r_MH:
            U_0 = U_new

        U_s[n,:] = U_0

    return U_s
```

[ ]:

```python
# test coarsening, this should half the field array
# u1 = np.linspace(1,10,20)
u1 = np.zeros(20)
phi1 = np.ones(20)
a = 4

u2,a2,phi2 = coarse(u1,a,phi1)
phi1,phi2, a2
```

[ ]:

```python
# test uncoarsening, this should double the lenght of the field array
u_ini = np.zeros(20)
u = np.linspace(0,9,10)
a = 4

u_h, a2 = uncoarse(u,a,u_ini)
u,u_h, a2
```

[ ]:

```python
def multi_grid(u_in,a_in,phi_in,l,g,n_pre,du,n_post,N_mh):
    '''Multigrid algorithm'''
    # u_in = initial config of u
    # a_in = initial coarsation
    # phi_in = phi in initial level
    # l = number of levels
    # g = gamma
    # n_pre = pre coarsational MH runs
    # n_post = post coarsational MH runs
    # du = for MH

    # pre-coarsening
    U_s = MH_new(u_in,a,du,n_pre,phi_in)

    # the precoarsened field
    u_pre = U_s[n_pre-1,:]

#     print(len(u_pre), u_pre.size)

    # max(coarsening) ==  when external field is described by 1 grid
    if len(u_pre)==1:

        U_pro = MH_new(u_pre,a,du,n_post,phi_in)

        return U_pro[n_post-1,:]

    # Initialising a list to store the value of the external field 'phi' at␣
 ↪each level.
    # A array cannot be used as each level as different dimensions for phi.
    # So in this, phi_l[l] would be equal to the field phi at level 'l'
    phi_l = []

    # Initialising a temporary variable to store the value of u for the initial␣
 ↪step.
    # This is updated every step through the coarsening (or uncoarsening later␣
 ↪on)
    # Similar to the previous case, a list is initialised to store the values␣
 ↪of initial values of u for each level
    # This is done, as we need to update the values of u for each uncoarsening.
    # So this list would help while uncoarsening to a particular layer, when␣
 ↪the value of u would be updated
    u_temp = u_pre
    ini = []

    # Initialising the value of 'a'
    a_temp = a_in

    # Initialising the value of field phi.
    # Creating a temporary variable as well, which can be used to get the value␣
 ↪of the field after coarsening
```

```
    phi_temp = phi_in


    # Updating the list 'ini' with the value of u at the initial level
    ini.append(u_temp)

    # Updating the list 'phi_l', with the value of phi for the first level
    phi_l.append(phi_temp)

    # going 'l' layers deep
    for i in range(l):

#         u_temp, a_temp, phi_l[l+1] = coarse(u_temp,a_temp,phi_l[l])
        u_temp, a_temp, phi_temp = coarse(u_temp,a_temp,phi_temp)
        ini.append(u_temp) # Updating the list 'ini' with the u for this level
        phi_l.append(phi_temp) # Updating the list 'phi_l' with the field of␣
 ↪phi for this level

    # starting the multigrid cycle for a value of gamma (g)
    # following the steps given in the lecture notes
    # from what we understood,
    # we follow a path given in the figure in Slide 11 in the lecture␣
 ↪'Approaching infinity, simulationally speaking'
    for j in range(1,g):
    #  we cover every level below g, so that is why we take a for loop from 1␣
 ↪to g.
        for k in range(j):
            # This for loop is to uncoarse up to level j
            # Meanwhile the respective elements of 'ini' get updated with the␣
 ↪new values of u
            u_temp,a_temp = uncoarse(u_temp,a_temp,ini[l-k])
            ini[l-k] += u_temp

        # After the level of j is reached, we do a Metropolis-Hastings (MH) run␣
 ↪for N_mh samples

        U_tmp = MH_new(u_temp,a_temp,du,N_mh,phi_l[l-j])

        # The last point of this MH cycle would give us the latest value for u,␣
 ↪which is again updated
        u_temp = U_tmp[N_mh-1,:]
        ini[l-j] = u_temp

        # After the MH run, we coarsen it back and go back to the last level
        for k in range(j):
            u_temp,a_temp,phi_temp = coarse(u_temp,a_mp,phi[l-j+(k+1)])

    # The above operations would be repeated until j= gamma.

    # As given in the lecture slides, we would like to do the similar cycles again for the lower levels,
    # in descending order (i.e, upto g-1, g-2) until we reach the deepest level l again.
    # Similar to the previous case, we run a MH cycle at each level and update the u in the list 'ini'
    for j in range(1,g-1)[::-1]:

        for k in range(j):

            u_temp,a_temp = uncoarse(u_temp,a_temp,ini[l-k])
            ini[l-k] += u_temp

        U_tmp = MH_new(u_temp,a_temp,du,N_mh,phi_l[l-j])

        u_temp = U_tmp[N_mh-1,:]

        for k in range(j):

            u_temp,a_temp,phi_temp = coarse(u_temp,a_mp,phi[l-j+(k+1)])

    # After the multigrid iterative cycles are done (for gamma), we uncoarsen the it back to the initial level.
    for i in range(l+1):

        u_temp, a_temp = uncoarse(u_temp,a_temp,ini[l-i])
        ini[l] += u_temp

    # u_first would be the value of u now at the initial level.
    u_first = ini[0]

    # After the multigrid cycle is finished, we run another Metropolis Hastings cycle for n_post samples.
    U_pro = MH_new(u_first,a,du,n_post,phi_in)

    # The last chain is returned, which is the result of the Multigrid algorithm
    return U_pro[n_post-1,:]
```

[ ]:

```
def autocorr(m2, tau):
    '''Normalized autocorrelation function'''
    N = len(m2)
    m2bar = np.mean(m2)
    gamma_arr = []

    if tau!= 0:
        for k in range(N-tau):

            l = k + tau

            gamma_arr.append( (m2[k] - m2bar) * (m2bar - m2[l]))

            temp_k = l
            temp_l = k

            gamma_arr.append( (m2[temp_l] - m2bar) * (m2bar - m2[temp_k]))

    if tau==0:

        for i in range(N):
            gamma_arr.append( (m2[i] - m2bar) * (m2bar - m2[i]))


    # take the average of gamma_arr
    gamma = np.sum(np.array(gamma_arr)) / len(gamma_arr)

    return gamma
```

[ ]:

```
# test
# u_ini = np.linspace(1,20,10)
u_ini = np.zeros(20)
phi_ini = np.ones(20)
a = 1
l = 2
g = 1

n_pre = 100
n_post = 100
N_MH = 100
du = 2.

# this part still does not work...
u_multi = multi_grid(u_ini,a,phi_ini,l,g,n_pre,du,n_post,N_mh)
```

[ ]:

```
l = 3   # number of levels
du = 2.
N = 64   # number of lattice points

# n_pre = [4,2,1]
# n_post = [4,2,1]
```

```
N_MH = 100

Ns = 100

u_ini = np.zeros(N)
phi_ini = np.ones(N)

cycles = [1,2]
mag2_arr = np.zeros(Ns)
tau = np.arange(0,Ns,step=20)
autocorr_arr = np.zeros((2,Ns))

# evaluate m^2 and the autocorrelation function for the different cycles

# since we have no idea how the implementation works with algorhtm, we set
# n_pre and n_post to be fixed. Sorry about this...

# here we will simply highlight what would be done if the bugs were fixed.
n_pre = 4
n_post = 4
for j, g in enumerate(cycles):

    # need an additional for loop here that iterates over each iteration
    for i in range(Ns):
        u_multi = multi_grid(u_ini,a,phi_ini,l,g,n_pre,du,n_post,N_MH)
        mag2_arr[i] = (np.sum(u_multi[1:-2]) / N)**2.

    # evaluate autocorrelation function
    C0_1 = autocorr(mag2_arr, 0)

    for i in range(len(tau_arr)):
        autocorr_arr[j, i] = autocorr(mag2_arr, tau_arr[i])

    autocorr_arr[j, :] /= C0_1
```

[ ]:

```
# plot autocorrelation function with m^2 vs tau
fig, ax = plt.subplots(figsize=(9,6))

ax.plot(tau_arr,autocorr_arr[1,:],marker='o',label='$\gamma = 1$', ms=2.0)
ax.plot(tau_arr,autocorr_arr[2,:],marker='o',label='$\gamma = 2$', ms=2.0)

ax.set_xlabel(r"$\tau$", fontsize=14)
ax.set_ylabel(r"$C(\tau) = \bar\Gamma^{{(m^2)}}(\tau)   /  ␣
 ↪\bar\Gamma^{{(m^2)}}(0)$", fontsize=14)
ax.set_title("Estimator for Normalized Autocorrelation function")
ax.grid()
ax.legend()
```

Below is the obsolete / WIP code (i.e. our attempts). Please forgive us for leaving this in the PDF as well, we will leave this here so that we can explore our mistakes if we have the time later on.

[ ]:

```
# def mgrid_cycle(u_i, a, phi_i, level, n_cycles, du, N_MH, n_pre=100, n_post=100):
#     '''
#     Performs the recursive step at a current level.

#     Parameters:
#     - u_i : the initial field (u_tilde in notes) at this level
#     - a: the lattice spacing == coarsation at this level
#     - phi_i: the external field at this level
#     - n_cycles: gamma in notes, number of multigrid cycles
#     - du: interval used in MH step
#     - N_MH: number of MH steps in each cycle
#     - n_pre: number of precoarsening steps for MH
#     - n_post: number of post-prolongation steps for MH

#     '''

#     # pre-coarsening
#     # get array of (Ns, len(u_in))
#     U_s = MH_new(u_i,a,du,n_pre,phi_i)

#     # the precoarsened field
#     # take last element of field value
#     u_pre = U_s[-1,:]

#     # max(coarsening) ==  when external field is described by 1 grid
#     # then perform post prolongation step immediately
#     if len(u_pre)==1:
#         U_pro = MH_new(u_pre,a,du,n_post,phi_in)
#         return U_pro[-1,:]

#     # perform coarsening
#     u = u_pre
#     a = a
#     phi = phi_i

#     # containers to keep field and lattice sizes
#     # for each level
#     mg_level_list = []

#     # go n_cycles levels deep
#     for nl in range(level):
# #         print(phi)
#         u, a, phi = coarse(u, a, phi)
#         mg_level_list.append((u, a, phi))

#     # perform multigrid MH step
#     # i.e. from lowest level, uncoarse -> MH -> coarse
#     # for gamma > 1, this happens in a symmetric fashion

#     # for gamma == 1, MH -> uncoarsen up to the finest level
#     if n_cycles == 1:
#         # perform MH at lowest level
#         u_l, a_l, phi_l = mg_level_list[-1]
#         u_l_MH = MH_new(u_l,a_l,du,N_MH,phi_l)[-1,:]

# #         print(u_l_MH.size)

#         # now uncoarsen
#         u_unc = u_l_MH
#         a_unc = a_l
#         # iterate backwards since we want to access the
#         # finer field values
#         for nl in range(level):
#             u_tilde = mg_level_list[level-nl][0]
#             u_unc, a_unc = uncoarse(u_unc, a_unc, u_tilde)

#         # final field given by uncoarsened u
#         u_f = u_unc

#     else:  # then do multigrid cycle recursion in symmetric fashion
#         pass

# #     print(u_f.size)

#     # perform post-prolongation step
#     # can use a, phi from initial state since
#     # we are back at the initial level
#     U_pro = MH_new(u_f,a,du,n_post,phi_i)

#     return U_pro[-1,:]

# #     for nc in range(1, n_cycles):
# #         # perform uncoarsening / coarsening depending on
# #         # n_cycles
# #         for nc_unc in range(nc):
# #             # first uncoarsen
# #             u_unc, a_unc = uncoarse()

# #     # perform MH step at each level going upwards,
# #     # starting from the lowest level in multigrid cycle
# #     for nc in range(n_cycles, step=-1):  # start from lowest level
# #         u_l, a_l, phi_l = mg_level_list[nc]

# #         u_l_MH = MH_new(u_l,a_l,du,N_MH,phi_l)[-1,:]

# #         u_temp, a_temp = uncoarse(u_temp,a_temp,ini[l-i])
```

[ ]:

```
# def multigrid(u_init, a, phi_init, nlevels, mgcycle_params, n_pre, n_post):
#     '''
#     Multigrid algorithm, iterates for nlevels and returns
#     field values evaluated for each level
#     '''
#     n_cycles, du, N_MH = mgcycle_params

#     u_arr = np.zeros((nlevels, len(u_init)))

#     for i, level in enumerate(range(1, nlevels+1)):
#         u_arr[i, :] = mgrid_cycle(u_init, a, phi_init, level, n_cycles, \
#                                    du, N_MH, n_pre[i], n_post[i])
#     return u_arr
```

[ ]:

```
# # test
# # u_ini = np.linspace(1,20,10)
# u_ini = np.zeros(20)
# phi_ini = np.ones(20)
# a = 1
# l = 2
# g = 1

# n_pre = 100
# n_post = 100
# N_MH = 100
# du = 2.

# # mgrid_cycle(u_ini, a, phi_ini, l, g, du, N_MH, n_pre=100, n_post=100)
# # u_multi = multi_grid(u_ini,a,phi_ini,l,g,n_pre,du,n_post,N_mh)

# mgcycle_params = (g, du, N_MH)

# multigrid(u_ini,a,phi_ini, l, mgcycle_params, n_pre=[4,2,1], n_post =␣
 ↪[4,2,1])
```

[ ]:

```
# u_ini = np.zeros(128)
# phi_ini = np.ones(128)
# a = 1
# l = 2
# g = 1

# n_pre = 100
# n_post = 100
# N_MH = 100
# du = 2.

# multi_grid(u_ini,a,phi_ini,l,g,n_pre,du,n_post,N_MH)
```

3. We start with the decomposition of the Hamiltonian after prolongation:

$$H_a(u^{(a)}) - H_a(\tilde{u}^{(a)}) = H_{2a}(u^{(2a)})$$

$$\Rightarrow \frac{1}{a} \sum_{i=1}^{N} (u_i^{(a)} - u_{i-1}^{(a)})^2 - (\tilde{u}_i^{(a)} - \tilde{u}_{i-1}^{(a)})^2 + a \sum_{i=1}^{N-1} \varphi_i^{(a)} (u_i^{(a)} - \tilde{u}_i^{(a)})$$

$$= \frac{1}{2a} \sum_{i=1}^{N/2} \left(u_{2i}^{(a)} - u_{2(i-1)}^{(a)} \right)^2 + 2a \sum_{i=1}^{N/2-1} \varphi_i^{(2a)} u_{2i}^{(a)}$$

Now using $u_i = \tilde{u}_i + I_{(2a)}^{(a)} u_i^{(2a)} \Rightarrow u_i - \tilde{u}_i = I_{(2a)}^{(a)} u_i^{(2a)}$

$$=: I\, u_i^{(2a)},$$

we can compare the second term in the LHS and RHS to see how we can express $\varphi_i^{(2a)}$ in terms of $\varphi_i^{(a)}$.

So we have:

$$a \cdot \sum_{i=1}^{N-1} \varphi_i^{(a)} (u_i - \tilde{u}_i) = a \cdot \sum_i \varphi_i^{(a)} I u_i^{(2a}$$

$$= a \cdot \sum_{i \text{ even}}^{N-2} \varphi_i^{(a)} \cdot u_{i/2}^{(2a)} + a \cdot \sum_{i \text{ odd}}^{N-1} \varphi_i^{(a)} \left[\frac{1}{2} \left(u_{(i-1)/2}^{(2a)} + u_{(i+1)/2}^{(2a)} \right) \right]$$

↓ shift index $j = i-1$

$$= a \cdot \sum_{k=1}^{\frac{N}{2}-1} \varphi_{2k}^{(a)} \cdot u_k^{(2a)} + a \cdot \sum_{j=0,\ j \text{ even}}^{N-2} \varphi_{j+1}^{(a)} \cdot \frac{1}{2} \left(u_{j/2}^{(2a)} + u_{j+2/2}^{(2a)} \right)$$

$$= a \cdot \sum_{k=1}^{\frac{N}{2}-1} \varphi_{2k}^{(a)} \cdot u_k^{(2a)} + \frac{a}{2} \sum_{j=1}^{\frac{N}{2}-1} \varphi_{2j+1}^{(a)} \left(u_j^{(2a)} + u_{j+1}^{(2a)} \right)$$

$$+ \frac{a}{2} \varphi_1^{(a)} \left(u_0^{(2a)} + u_1^{(2a)} \right)$$

$$= \frac{a}{2} \cdot \varphi_1^{(a)} \left(\cancelto{0}{U_0^{(2a)}} + U_1^{(2a)} \right) + a \sum_{k=1}^{\frac{N}{2}-1} \left\{ \varphi_{2k}^{(a)} U_k^{(2a)} + \frac{1}{2} \left(\varphi_{2k+1}^{(a)} \left(U_k^{(2a)} + U_{k+1}^{(2a)} \right) \right) \right\}$$

Since U(0)=0 by B.C.

$$= \frac{a}{2} \varphi_1^{(a)} U_1^{(2a)} + a \sum_{k=1}^{\frac{N}{2}-1} \left\{ \left[\varphi_{2k}^{(a)} + \frac{1}{2} \varphi_{2k+1}^{(a)} \right] U_k^{(2a)} + \frac{1}{2} \varphi_{2k+1}^{(a)} U_{k+1}^{(2a)} \right\}$$

$$= \frac{a}{2} \varphi_1^{(a)} U_1^{(2a)} + a \sum_{k=1}^{\frac{N}{2}-1} \left[\varphi_{2k}^{(a)} + \frac{1}{2} \varphi_{2k+1}^{(a)} \right] U_k^{(2a)} + \frac{a}{2} \sum_{k=1}^{\frac{N}{2}-1} \varphi_{2k+1}^{(a)} U_{k+1}^{(2a)}$$

shift $k+1 = l$

$$= \frac{a}{2} \varphi_1^{(a)} U_1^{(2a)} + a \sum_{k=1}^{\frac{N}{2}-1} \left[\varphi_{2k}^{(a)} + \frac{1}{2} \varphi_{2k+1}^{(a)} \right] U_k^{(2a)} + \frac{a}{2} \underbrace{\sum_{l=2}^{\frac{N}{2}} \varphi_{2(l-1)+1}^{(a)} U_l^{(2a)}}_{S'}$$

$$S' = \sum_{l=2}^{\frac{N}{2}} \varphi_{2(l-1)+1}^{(a)} U_l^{(2a)} = \sum_{l=2}^{\frac{N}{2}} \varphi_{2l-1}^{(a)} U_l^{(2a)}$$

$$= \sum_{l=2}^{\frac{N}{2}-1} \varphi_{2l-1}^{(a)} U_l^{(2a)} + \varphi_{N-1}^{(a)} \cancelto{0}{U_N^{(2a)}}$$

by B.C.

$$\Rightarrow \frac{a}{2} \varphi_1^{(a)} U_1^{(2a)} + \frac{a}{2} S' = \frac{a}{2} \varphi_1^{(a)} U_1^{(2a)} + \frac{a}{2} \sum_{l=2}^{\frac{N}{2}-1} \varphi_{2l-1}^{(a)} U_l^{(2a)}$$

$$= \frac{a}{2} \sum_{l=1}^{\frac{N}{2}-1} \varphi_{2l-1}^{(a)} U_l^{(2a)}$$

relabel $k := l$

$$= \frac{a}{2} \sum_{k=1}^{\frac{N}{2}-1} \varphi_{2k-1}^{(a)} U_k^{(2a)}$$

$$\Rightarrow a \cdot \sum_{i=1}^{N} \varphi_i^{(a)} (u_i - \tilde{u}_i)$$

$$= a \cdot \sum_{k=1}^{\frac{N}{2}-1} \left[\frac{1}{2} \varphi_{2k-1}^{(a)} + \varphi_{2k}^{(a)} + \frac{1}{2} \varphi_{2k+1}^{(a)} \right] U_k^{(2a)}$$

$$= 2a \cdot \sum_{k=1}^{\frac{N}{2}-1} \frac{1}{2} \left[\frac{1}{2} \varphi_{2k-1}^{(a)} + \varphi_{2k}^{(a)} + \frac{1}{2} \varphi_{2k+1}^{(a)} \right] U_k^{(2a)}$$

$$\overset{!}{=} 2a \sum_{k=1}^{N/2-1} \varphi_k^{(2a)} \, u_k^{(2a)}$$

$$\boxed{\begin{array}{l} \therefore \; \varphi_k^{(2a)} = \frac{1}{2} \cdot \left[\frac{1}{2} \varphi_{2k-1}^{(a)} + \varphi_{2k}^{(a)} + \frac{1}{2} \varphi_{2k+1}^{(a)} \right] \\ \qquad \text{for } k \in \left[1, \frac{N}{2} - 1\right] \end{array}}$$

1. $$\exp(-\beta H(u)) = \exp\left(-\frac{\beta}{a}\sum_{i=1}^{N}(u_i - u_{i-1})^2\right)$$

$$= \prod_{i=1}^{N-1} \exp\left(-\frac{\beta}{a}(u_i - u_{i-1})^2\right)$$

$$\Rightarrow \quad Z = \prod_{i=1}^{N-1} \int_{-\infty}^{+\infty} \left\{ \prod_{i=1}^{N-1} \exp\left(-\frac{\beta}{a}(u_i - u_{i-1})^2\right)\right\} du_i$$

$$= \int du_1 \int \cdots \int du_{N-1} \exp\left(-\frac{\beta}{a}(u_{N-1} - u_{N-2})^2\right) \cdot \cdots \exp\left(-\frac{\beta}{a}(u_1 - u_0)^2\right)$$

Evaluate the inner most integral ($i = N-1$):

$$\int du_{N-1} \exp\left(-\frac{\beta}{a}(u_{N-1} - u_{N-2})^2\right)$$

$$= \int du_{N-1} \exp\left(-\frac{\beta}{a}u_{N-1}^2 + \frac{2\beta}{a}u_{N-1}u_{N-2} - \frac{\beta}{a}u_{N-2}^2\right)$$

$$= e^{-\frac{\beta}{a}u_{N-2}^2} \int du_{N-1} \exp\left(-\frac{\beta}{a}u_{N-1}^2 + \frac{2\beta}{a}u_{N-2}u_{N-1}\right)$$

$$= \cancel{e^{-\frac{\beta}{a}u_{N-2}^2}} \cdot \sqrt{\frac{a\pi}{\beta}} \cdot \cancel{e^{\frac{4\beta^3 u_{N-2}^2}{4a^2 \cdot \frac{\beta}{a}}}}$$

$$= \sqrt{\frac{a\pi}{\beta}}$$

So proceed inductively to get:

$$Z = \left(\frac{a\pi}{\beta}\right)^{\frac{N-2}{2}} \cdot \int du_1 \exp\left(-\frac{\beta}{a}(u_1 - u_0)^2\right)$$

$$= \left(\frac{a\pi}{\beta}\right)^{\frac{N-1}{2}}. \qquad \Rightarrow \quad Z = \left(\frac{a\pi}{\beta}\right)^{\frac{N-1}{2}}$$

Now we go and evaluate the expectation values of the observables:

- Magnetization: $m(u) = \frac{a}{L} \sum_{i=1}^{N-1} u_i = \frac{1}{N} \sum_{i=1}^{N-1} u_i$

$$\Rightarrow \langle m \rangle = \frac{1}{Z} \cdot \prod_{i=1}^{N-1} \int du_i \, m(u) \cdot \exp(-\beta H(u))$$

$$= \frac{1}{Z} \prod_{i=1}^{N-1} \int du_i \cdot \left(\frac{1}{N} \cdot \sum_{i=1}^{N-1} u_i \right) \left(\cdot \prod_{i=1}^{N-1} \exp\left(-\frac{\beta}{a} (u_i - u_{i-1})^2\right) \right)$$

$$= \frac{1}{Z} \cdot \frac{1}{N^{N-1}} \int du_i \cdots \int du_{N-1} \cdot (u_1 + \cdots + u_{N-1})$$
$$\cdot \exp\left(-\frac{\beta}{a}(u_{N-1} - u_{N-2})^2\right) \cdots \exp\left(-\frac{\beta}{a}(u_1 - u_0)^2\right)\Big)$$

Now only look at the innermost integral $(i = N-1)$:

$$I_{N-1} = \int du_{N-1} \left(u_1 + \cdots + u_{N-1} \right) \exp\left(-\frac{\beta}{a} (u_{N-1} - u_{N-2})^2\right)$$

$$= \int du_{N-1} \, u_1 \cdot \exp\left(-\frac{\beta}{a} (u_{N-1} - u_{N-2})^2\right)$$
$$+ \cdots + \int du_{N-1} \, u_{N-1} \exp\left(-\frac{\beta}{a} (u_{N-1} - u_{N-2})^2\right)$$

$$= (u_1 + \cdots + u_{N-2}) \int du_{N-1} \exp\left(-\frac{\beta}{a} (u_{N-1} - u_{N-2})^2\right)$$
$$+ \underbrace{\int du_{N-1} \, u_{N-1} \exp\left(-\frac{\beta}{a} (u_{N-1} - u_{N-2})^2\right)}_{\tilde{I}_{N-1}}$$

$$\tilde{I}_{N-1} = \int du_{N-1} \, u_{N-1} \exp\left(-\frac{\beta}{a} (u_{N-1} - u_{N-2})^2\right)$$

$$= u_{N-2} \cdot \sqrt{\frac{a\pi}{\beta}}$$

$$\Rightarrow I_{N-1} = u_1 \sqrt{\frac{a\pi}{\beta}} + u_2 \sqrt{\frac{a\pi}{\beta}} + \cdots + 2\, u_{N-2} \sqrt{\frac{a\pi}{\beta}}$$

$$= \left(u_1 + \cdots + 2u_{N-2}\right)\sqrt{\frac{a\pi}{\beta}}$$

Now for $i = N-2$:

$$I_{N-2} = \sqrt{\frac{a\pi}{\beta}} \int du_{N-2} \left(u_1 + \cdots + 2u_{N-2}\right) \exp\left(-\frac{\beta}{a}\left(u_{N-2} - u_{N-3}\right)^2\right)$$

$$= \sqrt{\frac{a\pi}{\beta}} \cdot \sqrt{\frac{a\pi}{\beta}} \left(u_1 + \cdots + u_{N-3}\right)$$

$$+ 2\sqrt{\frac{a\pi}{\beta}} \int du_{N-2}\, u_{N-2} \exp\left(-\frac{\beta}{a}\left(u_{N-2} - u_{N-3}\right)^2\right)$$

$$= \left(\frac{a\pi}{\beta}\right)^{\frac{2}{2}} \left(u_1 + \cdots + u_{N-3}\right) + 2 \cdot \left(\frac{a\pi}{\beta}\right)^{\frac{2}{2}} u_{N-3}$$

$$= \left(\frac{a\pi}{\beta}\right)^{\frac{3}{2}} \left(u_1 + \cdots + 3u_{N-3}\right)$$

Proceed inductively. For the last stop, we get:

$$I_1 = \left(\frac{a\pi}{\beta}\right)^{\frac{N-2}{2}} (N-2) \int du_1 \cdot u_1 \exp\left(-\frac{\beta}{a}\left(u_1 - u_0\right)^2\right)$$

$$= \left(\frac{a\pi}{\beta}\right)^{\frac{N-1}{2}} (N-1) \cdot u_0$$

$$\boxed{\therefore \langle m \rangle = \frac{\cancel{1}}{\cancel{Z}} \cdot \frac{1}{N^{N-1}} \cdot \cancel{\left(\frac{a\pi}{\beta}\right)^{\frac{N-1}{2}}} \cdot (N-1) \cdot u_0 = \frac{N-1}{N^{N-1}} u_0}$$

- $$m^2(u) = \left(\frac{1}{N} \sum_{i=1}^{N-1} u_i\right)^2 = \frac{1}{N^2} \cdot \left(\sum_{i=1}^{N-1} u_i\right)^2$$

$$= \frac{1}{N^2} \left\{ \sum_{i=1}^{N-1} u_i^2 + \sum_{j \neq i} u_i u_j \right\}$$

Since we take $u_\ell = \sum_{k=1}^{N-1} c_k \sin\left(\frac{k\pi\ell}{N}\right) \Rightarrow u_\ell u_{\ell'} = 0 \ \forall \ell \neq \ell'$

So the cross terms will cancel.

$$\Rightarrow \quad m^2(u) = \frac{1}{N^2} \sum_{i=1}^{N-1} u_i^{\,2}$$

$$\Rightarrow \quad \langle m^2 \rangle = \frac{1}{Z} \cdot \prod_{i=1}^{N-1} \int du_i \cdot \frac{1}{N^2} \sum_{i=1}^{N-1} u_i^{\,2} \prod_{i=1}^{N-1} \exp\left(-\frac{\beta}{a}\left(u_i - u_{i-1}\right)^2\right)$$

$$= \frac{1}{Z} \cdot \prod_{i=1}^{N-1} \int du_i \cdots \int du_{N-1} \cdot \left(u_1^2 + \cdots + u_{N-1}^2\right)$$
$$\cdot \exp\left(-\frac{\beta}{a}\left(u_{N-1} - u_{N-2}\right)^2\right) \cdots \exp\left(-\frac{\beta}{a}\left(u_1 - u_0\right)^2\right)$$

Again, look at the $i = N-1$ case:

$$I_{N-1} \int du_{N-1} \left(u_1^{\,2} + \cdots + u_{N-1}^{\,2}\right) \exp\left(-\frac{\beta}{a}\left(u_{N-1} - u_{N-2}\right)^2\right)$$
$$= \left(u_1^{\,2} + \cdots + u_{N-2}^{\,2}\right)\sqrt{\frac{a\pi}{\beta}} + \underbrace{\int du_{N-1}\, u_{N-1}^{\,2} \exp\left(-\frac{\beta}{a}\left(u_{N-1} - u_{N-2}\right)^2\right)}_{\tilde{I}_{N-1}}$$

$$\tilde{I}_{N-1} = \int du_{N-1}\, u_{N-1}^{\,2} \exp\left(-\frac{\beta}{a}\left(u_{N-1} - u_{N-2}\right)^2\right)$$
$$= \frac{a}{2\beta} \cdot \sqrt{\frac{a\pi}{\beta}} \cdot \left(2\frac{\beta}{a} \cdot u_{N-2}^{\,2} + 1\right)$$
$$= \left(u_{N-2}^{\,2} + \frac{a}{2\beta}\right)\sqrt{\frac{a\pi}{\beta}}$$

$$\Rightarrow \quad I_{N-1} = \left(u_1^{\,2} + \cdots + 2u_{N-2}^{\,2} + \frac{a}{2\beta}\right)\sqrt{\frac{a\pi}{\beta}}$$

Now preceding inductively, we get:

$$\langle m^2 \rangle = \cancel{\frac{1}{Z}} \cdot \frac{1}{N^{2(N-1)}} \cdot \left((N-1)\, u_0 + (N-1)\frac{a}{2\beta}\right)\cancel{\left(\frac{a\pi}{\beta}\right)^{\frac{N-1}{2}}}$$

$$\boxed{\langle m^2 \rangle = \frac{N-1}{N^{2(N-1)}} \left(u_0 + \frac{a}{2\beta}\right)}$$

- Energy / Hamiltonian:

$$E = \langle H_a(u) \rangle = \frac{1}{Z} \cdot \prod_{i=1}^{N-1} \int du_i \; \frac{1}{a} \sum_{i=1}^{N-1} (u_i - u_{i-1})^2 \; \exp\left(-\frac{\beta}{a} H_a(u)\right)$$

... ...