In [1]:
```
import numpy as np
import random as rd
import matplotlib.pyplot as plt
```

Homework 5 from Martin Gräf and Richard Baumann

In this homework we look at a Multigrid simulation of the Gaussian model. A hamiltonian for that problem is given by:

$$H_a = \frac{1}{a}\sum_{i=1}^{N}(u_i - u_{i-1})^2. \tag{1}$$

We also use the Dirichlet boundary conditions, which state:

$$u(0) = u_0 = 0, \tag{2}$$

$$u(L) = u_N = 0. \tag{3}$$

This Hamiltonian is implemented below:

In [2]:
```
def Hamiltonian(u_array, a):
    sum=0

    # Dirichlet boundary condition (u_0 = 0)
    sum += (u_array[1] - 0)**2

    # The middle is normal (u_i - u_(i-1))**2
    for i in range(2, len(u_array)-2):
        sum += (u_array[i] - u_array[i-1])**2

    # Dirichlet boundary condition (u_N = 0)
    sum += (0 - u_array[len(u_array)-1])**2

    return (sum/a)
```

The value for the magnetization is given as:

$$m = \frac{a}{L}\sum_{i=1}^{N-1} u_i \tag{4}$$

And naturally the value for the squared magnetization is given as:

$$m^2 = \frac{a^2}{L^2}\sum_{i=1}^{N-1} u_i^2 \tag{5}$$

In [3]:
```
def magnetization(u_array, a):
    sum = 0
    # Just summing over all u`s as stated above
    for i in range(1, len(u_array)-1):
        sum += u_array[i]

    # The final result is (L/a)*sum
    return (sum/len(u_array))

def magnetization_squared(u_array, a):
    sum = 0
    # The same as above just for the
    # squared value of the magnetization
    for i in range(1, len(u_array)-1):
        sum += u_array[i]**2

    # The final result is (L/a)*sum
    return (sum/len(u_array))
```

The partition sum for the system is given by:

$$Z(\beta, N, a) = \prod_{i=1}^{N-1}\int_{-\inf}^{\inf} du_i e^{-\beta H_a(u)} \tag{6}$$

Furthermore the expectation value of a observable is given by:

$$< o >= \frac{1}{Z}\int du\, o \cdot e^{-\beta H} \tag{7}$$

We can use the fourier decomposition:

$$u_l = \sum_{k=1}^{N-1} c_k \sin\left(i\frac{k\pi a}{L}\right), \tag{8}$$

to rewrite the hamiltonian to:

$$H_a = \frac{2N}{a} \sum_{k=1}^{N-1} c_k^2 \sin^2\left(i\frac{k\pi}{2N}\right), \tag{9}$$

1: Based on the above Fourier decomposition determine the analytic formula for the expectation value 1 of:

- the magnetization
- its square m
- the energy

As stated before, the magnetization is defined as:

$$m = \frac{a}{L} \sum_{i=1}^{N-1} u_i \tag{10}$$

With the fourier decomposition this becomes:

$$m = \frac{a}{L} \sum_{i=1}^{N-1} u_i = \frac{a}{L} \sum_{i=1}^{N-1} \sum_{k=1}^{N-1} c_k \sin\left(i\frac{k\pi a}{L}\right) = \frac{a}{L} \sum_{i=1}^{N-1} \sum_{k=1}^{N-1} c_k \sin\left(i\frac{k\pi}{N}\right) \tag{11}$$

Here we can use the identity, that was given on the exercise sheet:

$$\sum_{i=1}^{N-1} c_k \sin\left(i\frac{k\pi}{N}\right) = \begin{cases} 0, & \text{if k even} \\ \cot\left(\frac{k\pi}{2N}\right), & \text{if k odd} \end{cases} \tag{12}$$

We can write:

$$\frac{a}{L} \sum_{i=1}^{N-1} \sum_{k=1}^{N-1} c_k \sin\left(i\frac{k\pi}{N}\right) = \frac{a}{L} \sum_{i=1}^{N-1} \sum_{k=1}^{N-1} c_k \sin\left(i\frac{k\pi}{N}\right) = \frac{a}{L} \sum_{k=1}^{N-1} c_k \sum_{i=1}^{N-1} \sin\left(i\frac{k\pi}{N}\right) = \frac{a}{L} \sum_{k=1}^{N-1} c_k \begin{cases} 0, & \text{if k even} \\ \cot\left(\frac{k\pi}{2N}\right), & \text{if k odd} \end{cases} \tag{13}$$

We also need to say, that for an expectation value we integrate over u, but the fourier decomposition changes that. Therefore we get:

$$\frac{du}{dc} = const \rightarrow du = const \cdot dc \tag{14}$$

Therefore we can rewrite the integral from being over u to over c.

Putting that in the prior derived formula for an expected value we get:

$$< m >= \frac{1}{Z} \int du\, m \cdot e^{-\beta H} = \frac{1}{Z} \int du\, e^{-\beta \frac{2N}{a} \sum_{k=1}^{N-1} c_k^2 \sin^2\left(i\frac{k\pi}{2N}\right)} \frac{a}{L} \sum_{k=1}^{N-1} c_k \begin{cases} 0, & \text{if k even} \\ \cot\left(\frac{k\pi}{2N}\right), & \text{if k odd} \end{cases} \tag{15}$$

A relation, that can be used to simplefy the result is:

$$\int_{\infty}^{\infty} dc\, c\, e^{-\frac{c^2}{2\sigma^2}} = 0 \tag{16}$$

With this relation we can see that the expected value for the magnetization is 0.

For the squared magnetization we can see:

$$m^2 = \frac{a}{L} \sum_{i=1}^{N-1} u_i^2 \tag{17}$$

We can again introduce the fourier decomposition:

$$m^2 = \frac{a^2}{L^2} \sum_{i=1}^{N-1} u_i^2 = \frac{a^2}{L^2} \sum_{i=1}^{N-1} \sum_{k=1}^{N-1} c_k \sin\left(i\frac{k\pi}{N}\right) \sum_{l=1}^{N-1} c_l \sin\left(i\frac{l\pi}{N}\right) = \frac{a^2}{L^2} \sum_{k=1}^{N-1} c_k \sum_{l=1}^{N-1} c_l \sum_{i=1}^{N-1} \sin\left(i\frac{k\pi}{N}\right) \sin\left(i\frac{l\pi}{N}\right) \tag{18}$$

This can be simplefied using a given identity:

$$\sum_{i=1}^{N-1} \sin\left(i\frac{k\pi}{N}\right) \sin\left(i\frac{l\pi}{N}\right) = \frac{N}{2} \delta_{k,l} \tag{19}$$

Putting this in our equation we get:

$$m^2 = \frac{a^2}{L^2} \sum_{i=1}^{N-1} \sum_{k=1}^{N-1} c_k \sin\left(i\frac{k\pi}{N}\right) \sum_{l=1}^{N-1} c_l \sin\left(i\frac{l\pi}{N}\right) = \frac{a^2}{L^2} \sum_{k=1}^{N-1} c_k \sum_{l=1}^{N-1} c_l \frac{N}{2} \delta_{k,l} = \frac{Na^2}{2L^2} \sum_{k=1}^{N-1} c_k^2 \tag{20}$$

Therefore, for the expectation value, we get:

$$< m^2 >= \frac{1}{Z} \int du \, \frac{Na^2}{2L^2} \sum_{k=1}^{N-1} c_k^2 \cdot e^{-\beta \frac{2N}{a} \sum_{k=1}^{N-1} c_k^2 \sin^2\left(i\frac{k\pi}{2N}\right)} \tag{21}$$

Finallay the expected value for the energy can be calculated by taking the already fourier decomposed Hamiltonian:

$$H_a = \frac{2N}{a} \sum_{k=1}^{N-1} c_k^2 \sin^2\left(i\frac{k\pi}{2N}\right), \tag{22}$$

Therefore the expected value for the energy becomes:

$$< E >= \frac{1}{Z} \int du \, \frac{2N}{a} \sum_{k=1}^{N-1} c_k^2 \sin^2\left(i\frac{k\pi}{2N}\right) \cdot e^{-\beta \frac{2N}{a} \sum_{k=1}^{N-1} c_k^2 \sin^2\left(i\frac{k\pi}{2N}\right)} \tag{23}$$

We can substitute terms:

$$y = \sum_{k=1}^{N-1} c_k^2 \sin^2\left(i\frac{k\pi}{2N}\right) \tag{24}$$

Which leads to

$$\frac{dy}{dc} = \sum_{k=1}^{N-1} 2c_k \sin^2\left(i\frac{k\pi}{2N}\right) \rightarrow dc = \frac{dy}{\sum_{k=1}^{N-1} 2c_k \sin^2\left(i\frac{k\pi}{2N}\right)} \tag{25}$$

Right now I think I am on the wrong track for m^2 and E, therefore I will leave this here and improve it if there is more time.

2: Implement the above version of the Metropolis-Hastings sweep. Test your algorithm by sampling with δ = 2., N = 64, β = 1. Perform a measurement of the magnetization and energy after each sweep. Compare to the analytic results for the expectation values.

In [4]:
```
def metropolis_hastings(u_array, delta, a):
```

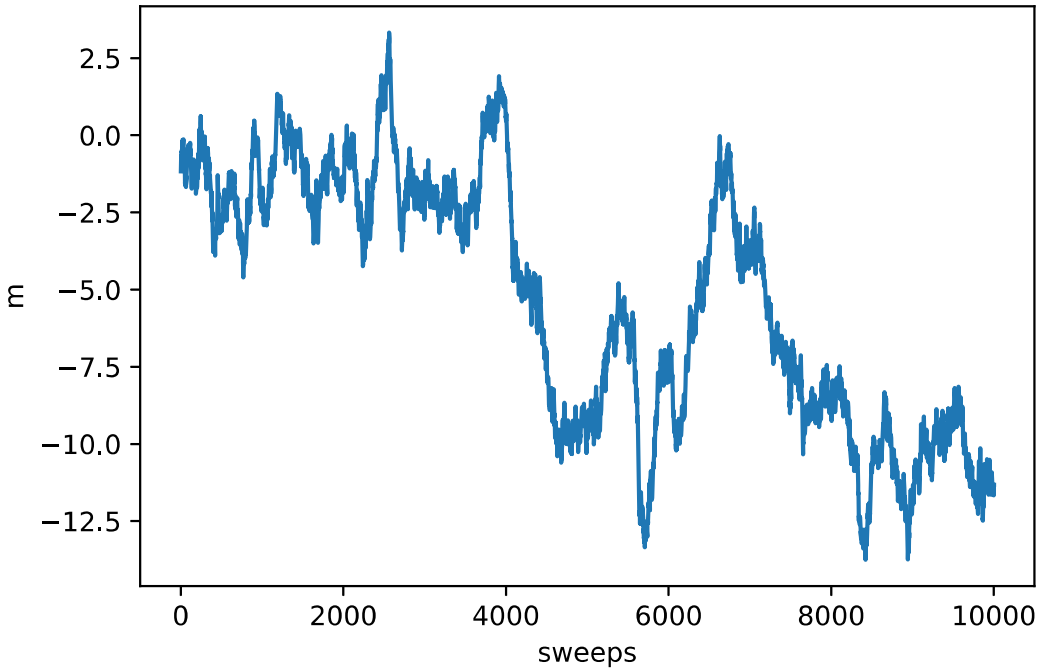
```
    # Make a copy of the u_array to do the update,
    # but keep the original array
    u_array_temp=u_array

    # The position of the update is determined
    # ({1, ..., N-1})
    position=rd.randint(0, len(u_array_temp)-2)

    # Calculate the energy before the update
    energy_before=Hamiltonian(u_array_temp, a)

    # Do the update: u'(x)=u(x)+\delta*r
    # where r is a random number between -1 and 1.
    u_array_temp[position]=u_array_temp[position]+rd.uniform(-1, 1)*delta

    # Now calculate the energy after the update
    energy_after=Hamiltonian(u_array_temp, a)

    # Here is the accept reject step as done in previous exercises
    if np.random.uniform(0, 1) <= np.exp(-(energy_after-energy_before)):
        # If the change is accepted, the u_array is overwritten with the
        # temporary copy, if not, the u_array stays as it is
        u_array=u_array_temp
    #else:
        #print("rejected")

    # Return the array
    return (u_array)
```

In [5]:

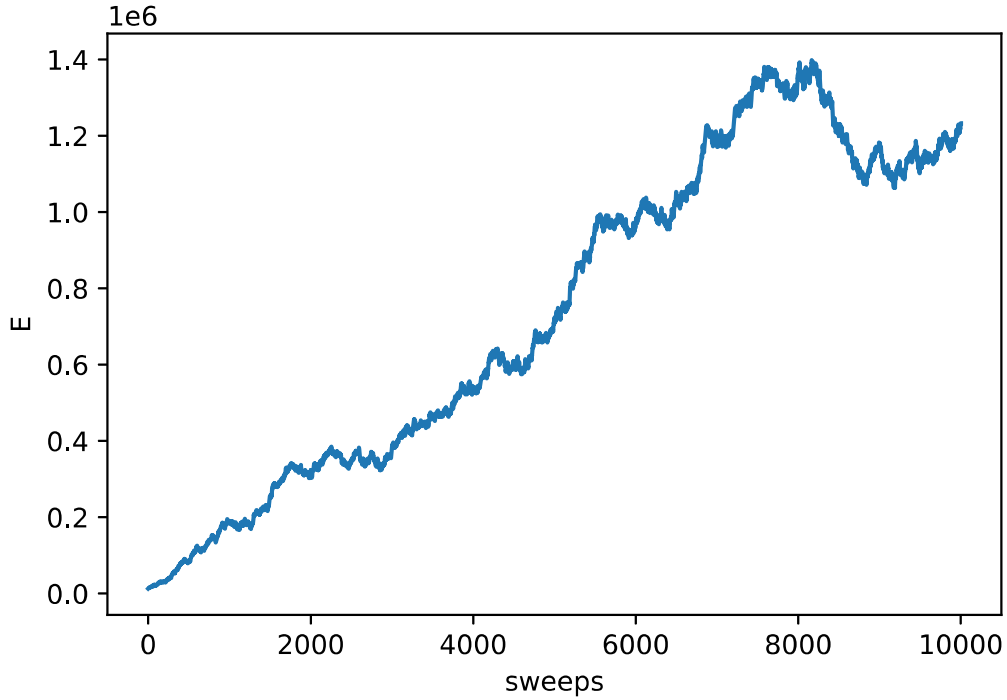
```
ntherm=4000
measurements=10000
u_array=np.zeros(64)

energy_array=[]
magnetization_array=[]
magnetization_squared_array=[]

# Do a lot of metropolis Hastings steps in order to
# thermalize the array
for i in range(ntherm):
    metropolis_hastings(u_array, 2, 1)

# Now we do k measurements
for k in range(measurements):
```

```
    # But before the measurement we do N-1
    # accept reject steps, which are one sweep
    for i in range(len(u_array)-1):
        metropolis_hastings(u_array, 2, 1)

    # Now we append the measurements to the respective array:
    energy_array.append(Hamiltonian(u_array, 1))
    magnetization_array.append(magnetization(u_array, 1))
    magnetization_squared_array.append(magnetization_squared(u_array, 1))
```

In [6]:
```
plt.plot(magnetization_array)
plt.xlabel("sweeps")
plt.ylabel("m")
plt.savefig("pictures/magnetization.png", dpi=600)
```



In [7]:
```
plt.plot(energy_array)
plt.xlabel("sweeps")
plt.ylabel("E")
plt.savefig("pictures/energy.png", dpi=600)
```



If we look at our generated values, we can see, that they are very fluctulant even if we have a lot of thermalziation and measurements. We can see, that the magnetization is oscillating around 0 and the energy is converging towards a high value. This system seems to take a long time to converge against the value that we are looking for. Therefore we look at the multigrid algorithm.

Now we want to introduce the actual simulation of the multigrid algorithm. In general we want to coarsen or fine our grid, to shorten our calculations:

If we want to make our grid more coarse, we just use every 2nd grid point:

$$u_i^{(2a)} = u_{2i}^{(a)} \text{with i } \in [0, \ldots, N/2] \tag{26}$$

Making our grid finer again is a bit more complicated, but in general we just make an interpolation:

$$I_{(2a)}^{(a)} u^{(2a)} = \begin{cases} u_{i/2}^{(2a)}, & i \in [0, 2, \ldots, N] \\ \left[u_{(i+1)/2}^{(2a)} + u_{(i-1)/2}^{(2a)} \right] /2, & i \in [1, 3, \ldots, N-1] \end{cases} \tag{27}$$

Untill now, we have neglected the external field (or pretended it is 0). Now we can write the new Hamiltonian as:

$$H_a = \frac{1}{a}\sum_{i=1}^{N}(u_i - u_{i-1})^2 + a\sum_{i=1}^{N-1}\Phi_i^a u_i \tag{28}$$

The prolongation can be expressed as:

$$u^{(a)} = \tilde{u}^{(a)} + I_{(2a)}^{(a)} u^{(2a)} \tag{29}$$

The hamiltonian for a one step coarser leads to:

$$H_a = H_a(\tilde{u}) + H_{2a}(u^{(2a)}) \tag{30}$$

With

$$H_{2a}(u^{(2a)}) = \frac{1}{2a}\sum_{i=1}^{N/2}(u_i^{(2a)} - u_{i-1}^{(2a)})^2 + 2a\sum_{i=1}^{N/2-1}\Phi_i^{2a} u_i^{(2a)} \tag{31}$$

In [8]:
```python
def coarsen(array):
    coarsen_array=[]
    for i in range(len(array)):
        if i%2==0:
            coarsen_array.append(array[i])
    return coarsen_array

def interpolate(array):
    interpolated_array=[]
    for i in range(2*len(array)):
        if i%2==1:
            interpolated_array.append(array[int(i/2)])
        else:
            interpolated_array.append((array[int((i+1)/2)]+array[int((i-1)/2)])/2)
    return interpolated_array
```

3: Give the explicit form of $\Phi^{(2a)}$: how does the coarse-level external field $\Phi^{(2a)}$ depend on the fine-level fields ? Implement the restriction and prolongation functions.

We know the contribution of the external field is given as:

$$\sum_i (\tilde{u}_i^{(a)} + [Iu^{(2a)}]_i)\Phi_i^{(a)} = \sum_i \tilde{u}_i^{(a)}\Phi_i^{(a)} + \sum_i [Iu^{(2a)}]_i \Phi_i^{(a)} \tag{32}$$

In this equation the second part gives the contribution, that we are looking for. The contribution that can be attributed to the coarse field can be expresses as:

$$a\sum_i^{N/2-1} u_i^{(2a)}\Phi_{2i}^{(a)} + \frac{a}{2}\sum_i^{N/2-1}(u_i^{(2a)} + u_{i+1}^{(2a)})\Phi_{2i+1}^{(a)} \tag{33}$$

Shifting the index of this equation finally gives us:

$$2a\sum_i^{N/2-1} u_i^{(2a)}\frac{\Phi_{2i-1}^{(a)} + 2\Phi_{2i}^{(a)} + \Phi_{2i+1}^{(a)}}{4} \tag{34}$$

Comparing that to the equation above gives us:

$$2a\sum_{i=1}^{N/2-1}\Phi_i^{2a}u_i^{(2a)} = 2a\sum_i^{N/2-1} u_i^{(2a)}\frac{\Phi_{2i-1}^{(a)} + 2\Phi_{2i}^{(a)} + \Phi_{2i+1}^{(a)}}{4} \tag{35}$$

And therefore:

$$\frac{\Phi_{2i-1}^{(a)} + 2\Phi_{2i}^{(a)} + \Phi_{2i+1}^{(a)}}{4} = \Phi_i^{2a} \tag{36}$$

Now we can look at the multigrid algorithm. The algorithm is defined as a recursive function:

- 1 : do ν_{pre} pre-coarsening sweeps at the current level; if the current level is the coarsest level, proceed to step 5
- 2 : generate the next coarser level for grid spacing 2a, i.e. determine $\Phi_{(2a)}$ and thus the Hamiltonian $H_{(2a)}$ (to use in sweeps on the coarser level)
- 3 : do γ multigrid cycles for the coarser level; start initially with $u_{(2a)} \equiv 0$;
- 4 : update the current u*{(a)} by u*{(a)} $\leftarrow u^{(a)} = \tilde{u}^{(a)} + I_{(2a)}^{(a)}u^{(2a)}$
- 5 : do ν_{post} post-prolongation sweeps at the current level

4: Implement the above multigrid algorithm for n_{level} levels and number of cycles γ

In [9]:
```
def multigrid(array, gamma, delta, n_levels, pre_post_sweeps):
    for levels in range(n_levels):

        # Doing pre coarsening sweeps. The ammount is
        # given by the k-th entry in the array pre_post_sweeps
        for k in range(pre_post_sweeps[levels]):
            # Do N-1 accept rejects (1 Sweep)
            for i in range(len(u_array)-1):
                metropolis_hastings(u_array, delta, levels)

        coarser_grid=coarsen(array)
        for j in range(gamma):
            for i in range(len(u_array)-1):
                metropolis_hastings(coarser_grid, delta, levels)
```

5: Test your implementation with N= 64, δ= 2, γ= 1,2 and n_{level}= 3. Moreover,set the following number of sweeps for coarsest to finest level: $\nu_{pre} = \nu_{post}$= 4,2,1.Plot the autocorrelation function of the squared magnetization as defined in eq. (5) for this choice of parameters.

Conclusion

All in all we can say, that this exercise was not our best submission. We failed to get the analytical solutions wanted in Task 1 and we did fail to get an explicit form of $\Phi^{(2a)}$. The problem with both of these things are, that they are needed to proceed. Therefore we did not do a lot of progress in this exercise, which is sad, because we really enjoyed working with the method.