model_B

October 27, 2025

1 Equilibrium model B

1.1 Conserved dynamics

Let us define $\phi(\mathbf{r}, t)$ to be the rescaled density. The coarse-grained Hamiltonian can be written as:

$$\mathcal{H}[\phi] = \int_V d\mathbf{r} \left\{ \frac{a}{2}\phi^2 + \frac{b}{4}\phi^4 + \frac{\kappa}{2}|\nabla\phi|^2 \right\}, \tag{1}$$

where $b, \kappa > 0$ (otherwise the energy is not bounded from below). a can be positive or negative. Note that $d\mathbf{r}$ is the differential volume, which is sometimes also written as dV or $d^d r$ (in d-dimension). The dynamics then follows the conservation law:

$$\frac{\partial\phi}{\partial t} + \nabla\cdot\mathbf{J} = 0 \quad \text{and} \quad \mathbf{J} = -\lambda\nabla\frac{\delta\mathcal{H}}{\delta\phi} + \Lambda, \tag{2}$$

where $\lambda > 0$ is the mobility. Correspondingly, the global density $\phi_0 = \frac{1}{V}\int\phi\, d\mathbf{r}$ is constant with time. $\Lambda(\mathbf{r}, t)$ in the equation above is a Gaussian white noise with zero mean and Dirac delta-correlation:

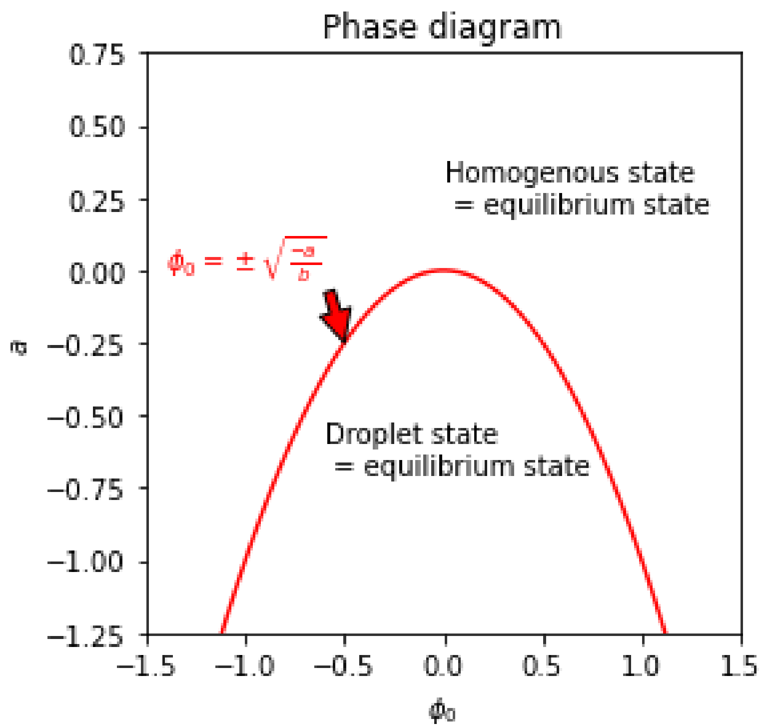
$$\left\langle \Lambda_\alpha(\mathbf{r}, t)\Lambda_\beta(\mathbf{r}', t') \right\rangle = 2\lambda T\delta_{\alpha\beta}\delta(\mathbf{r}-\mathbf{r}')\delta(t-t'). \tag{3}$$

The noise correlation above satisfies FDT, which guarantees that the system will be in thermal equilibrium with a heat bath of temperature T at steady state $t\to\infty$.

The equilibrium state of the system depends on the value of a and ϕ_0: - $a > 0$ or $a < 0$ and $|\phi_0| > \sqrt{\frac{-a}{b}}$: homogenous state - $a < 0$ and $|\phi_0| < \sqrt{\frac{-a}{b}}$: phase-separated state.

[2]:
```
import numpy as np
import matplotlib.pyplot as plt

b, kappa = 1.0, 1.0
phi = np.arange(-2.0, 2.0, 0.001)

fig, ax = plt.subplots(figsize=(4,4))

ax.set_title('Phase diagram')
ax.set_ylabel('$a$')
ax.set_xlabel('$\phi_0$')
ax.set_xlim([-1.5, 1.5])
ax.set_ylim([-1.25, 0.75])
```

```
a = -b*phi**2
ax.plot(phi, a, c='red')

ax.annotate('Homogenous state \n = equilibrium state', xy=(0.0,0.2))
ax.annotate('Droplet state \n = equilibrium state', xy=(-0.6,-0.7))
ax.annotate('$\phi_0=\pm\sqrt{\\frac{-a}{b}}$', c='red', xy=(-0.5,-0.25),␣
↪xytext=(-1.4,0.0), arrowprops=dict(facecolor='red'))

plt.show()
```



1.2 Steady state statistics

In the steady state $t \to \infty$, and for a fixed value of a, the probability of obtaining some configuration $\phi(\mathbf{r})$, for any time t, is given by the Boltzmann distribution:

$$P_s[\phi(\mathbf{r})] = \frac{1}{\mathcal{Z}} e^{-\mathcal{H}[\phi(\mathbf{r})]/T}, \tag{4}$$

where $\mathcal{Z}$ is the partition function:

$$\mathcal{Z} = \int \mathcal{D}\phi \, e^{-\mathcal{H}[\phi]/T}. \tag{5}$$

Note that the Hamiltonian $\mathcal{H}[\phi]$ is a fluctuating quantity since ϕ is a fluctuating field. To get the thermodynamic energy, we then have to average $\mathcal{H}[\phi]$ over the stationary distribution $P_s[\phi]$:

$$\langle \mathcal{H} \rangle_s = \int \mathcal{D}\phi \, \mathcal{H}[\phi] P_s[\phi], \tag{6}$$

where in the above $\langle \dots \rangle_s$ indicates averaging over stationary distribution $P_s[\phi]$. Now the thermodynamic entropy of the system $\mathcal{S}$ is defined to be:

$$\mathcal{S} = - \langle \ln P_s \rangle_s . \tag{7}$$

Substituting $P_s[\phi]$ to the above equation, we then derive the *total* free energy of the system:

$$\mathcal{F} = -T \ln \mathcal{Z} = \langle \mathcal{H} \rangle_s - T\mathcal{S}. \tag{8}$$

Note that in some literature $\mathcal{H}$ is sometimes called the coarse-grained free energy and $\mathcal{F}$ is the *total* free energy.

1.3 Gaussian approximation

Let us consider the equilibrium homogenous state. In steady state, we have an ensemble of different configurations $\phi(\mathbf{r})$'s from different time steps. Let us now write $\phi(\mathbf{r})$ as:

$$\phi(\mathbf{r}) = \underbrace{\phi_0}_{\text{mean field}} + \underbrace{\delta\phi(\mathbf{r})}_{\text{fluctuations around mean field}} , \tag{9}$$

where $\delta\phi(\mathbf{r})$ is assumed to be small. Substituting the above into the Hamiltonian $\mathcal{H}[\phi]$, we get:

$$\mathcal{H}[\phi] = \int_V d\mathbf{r} \left\{ \frac{a}{2}(\phi_0 + \delta\phi)^2 + \frac{b}{4}(\phi_0 + \delta\phi)^4 + \frac{\kappa}{2}|\nabla \delta\phi|^2 \right\} \tag{10}$$

$$\simeq \int_V d\mathbf{r} \left\{ \frac{a}{2}(\phi_0^2 + 2\phi_0\delta\phi + \delta\phi^2) + \frac{b}{4}(\phi_0^4 + 4\phi_0^3\delta\phi + 6\phi_0^2\delta\phi^2) + \frac{\kappa}{2}|\nabla \delta\phi|^2 \right\}, \tag{11}$$

where we have ignored higher order terms $\sim \delta\phi^3$. Now since ϕ is conserved, we must have $\int_V \delta\phi \, d\mathbf{r} = 0$, and thus:

$$\mathcal{H}[\delta\phi] \simeq \underbrace{V\left(\frac{a}{2}\phi_0^2 + \frac{b}{4}\phi_0^4\right)}_{\mathcal{H}_0} + \int_V d\mathbf{r} \left\{ \left(\frac{a}{2} + \frac{3b\phi_0^2}{2}\right) \delta\phi^2 + \frac{\kappa}{2}|\nabla \delta\phi|^2 \right\}. \tag{12}$$

The first term $\mathcal{H}_0 =$ constant is the mean field energy. Let us consider a d-dimenional box as our system. Now we can define the Fourier transform of $\delta\phi(\mathbf{r})$:

$$\delta\phi(\mathbf{r}) = \frac{1}{\sqrt{V}} \sum_{\mathbf{q}} \delta\phi_{\mathbf{q}} e^{i\mathbf{q}\cdot\mathbf{r}} \tag{13}$$

$$\delta\phi_{\mathbf{q}} = \frac{1}{\sqrt{V}} \int_V d\mathbf{r} \, \delta\phi(\mathbf{r}) e^{-i\mathbf{q}\cdot\mathbf{r}}, \tag{14}$$

where $V = L^d$ and

$$q_\alpha = 0, \pm\frac{2\pi}{L}, \pm\frac{4\pi}{L}, \pm\frac{6\pi}{L}, \dots \quad \text{, where } \alpha = 1, 2, \dots, d. \tag{15}$$

More succintly, we can write

$$\mathbf{q} = \frac{2\pi}{L}\mathbf{n} \quad \text{, where } \mathbf{n} \in \mathbb{Z}^d. \tag{16}$$

The Hamiltonian then becomes:

$$\mathcal{H}\{\delta\phi_{\mathbf{q}}\} = \mathcal{H}_0 + \int_V d\mathbf{r} \left\{ \left(\frac{a}{2} + \frac{3b\phi_0^2}{2} \right) \frac{1}{V} \sum_{\mathbf{q},\mathbf{q}'} \delta\phi_{\mathbf{q}} \delta\phi_{\mathbf{q}'} e^{i(\mathbf{q}+\mathbf{q})\cdot\mathbf{r}} + \frac{\kappa}{2} \frac{1}{V} \sum_{\mathbf{q},\mathbf{q}'} (i\mathbf{q}) \cdot (i\mathbf{q}') \delta\phi_{\mathbf{q}} \delta\phi_{\mathbf{q}'} e^{i(\mathbf{q}+\mathbf{q})\cdot\mathbf{r}} \right\} \tag{17}$$

$$= \mathcal{H}_0 + \sum_{\mathbf{q},\mathbf{q}'} \left(\frac{a}{2} + \frac{3b\phi_0^2}{2} \right) \delta\phi_{\mathbf{q}} \delta\phi_{\mathbf{q}'} \underbrace{\frac{1}{V} \int_V d\mathbf{r} e^{i(\mathbf{q}+\mathbf{q}')\cdot\mathbf{r}}}_{\delta_{\mathbf{q},-\mathbf{q}'}} + \sum_{\mathbf{q},\mathbf{q}'} \frac{\kappa}{2} (i\mathbf{q}) \cdot (i\mathbf{q}') \delta\phi_{\mathbf{q}} \delta\phi_{\mathbf{q}'} \underbrace{\frac{1}{V} \int_V d\mathbf{r} e^{i(\mathbf{q}+\mathbf{q}')\cdot\mathbf{r}}}_{\delta_{\mathbf{q},\mathbf{q}'}} \tag{18}$$

$$= \mathcal{H}_0 + \frac{1}{2} \sum_{\mathbf{q}} \left(a + 3b\phi_0^2 + \kappa q^2 \right) |\delta\phi_{\mathbf{q}}|^2 \tag{19}$$

To simplify the notation, let us define:

$$G(\mathbf{q}) = \frac{a + 3b\phi_0^2 + \kappa q^2}{T}, \tag{20}$$

so that the stationary probability distribution becomes:

$$P_s\{\delta\phi_{\mathbf{q}}\} = \frac{1}{\mathcal{Z}} e^{-\frac{1}{2}\sum_{\mathbf{q}} G(\mathbf{q})|\delta\phi_{\mathbf{q}}|^2} \tag{21}$$

$$\mathcal{Z} = \left(\prod_{\mathbf{q}} \int d\delta\phi_{\mathbf{q}} \right) e^{-\frac{1}{2}\sum_{\mathbf{q}} G(\mathbf{q})|\delta\phi_{\mathbf{q}}|^2} \tag{22}$$

Note that since $\mathcal{H}_0 =$ constant, we can absorb it inside $\mathcal{Z}$. Now we can compute $\mathcal{Z}$:

$$\mathcal{Z} = \left(\prod_{\mathbf{q}} \int d\delta\phi_{\mathbf{q}} \right) e^{-\frac{1}{2}\sum_{\mathbf{q}} G(\mathbf{q})|\delta\phi_{\mathbf{q}}|^2} \tag{23}$$

$$= \prod_{\mathbf{q}} \left(\int d\delta\phi_{\mathbf{q}} \, e^{-\frac{1}{2} G(\mathbf{q})|\delta\phi_{\mathbf{q}}|^2} \right). \tag{24}$$

The integral inside the round bracket is a Gaussian integral over the two random variables: $\mathrm{Re}(\delta\phi_{\mathbf{q}})$ and $\mathrm{Im}(\delta\phi_{\mathbf{q}})$. However these two variables are not independent since $\delta\phi_{\mathbf{q}} = \delta\phi^*_{-\mathbf{q}}$, and effectively, this is just a one-dimensional Gaussian integral. Thus,

$$\mathcal{Z} = \prod_{\mathbf{q}} \sqrt{\frac{2\pi}{G(\mathbf{q})}}. \tag{25}$$

In particular, we can calculate the total free energy:

$$\mathcal{F} = -T \ln \mathcal{Z} \tag{26}$$

$$= -\frac{T}{2} \sum_{\mathbf{q}} \Delta\mathbf{n} \ln \left(\frac{2\pi}{G(\mathbf{q})} \right) \tag{27}$$

$$\simeq -T \frac{V}{(2\pi)^d} \int_0^{q_{\max}} dq \, \Omega_d q^{d-1} \ln \left(\frac{2\pi}{G(\mathbf{q})} \right), \tag{28}$$

Note that $\mathbf{1} = \Delta \mathbf{n} = \frac{L}{2\pi}\Delta \mathbf{q}$. In the equation above, Ω_d is the solid angle in d-dimension:

$$\Omega_d = \frac{2\pi^{d/2}}{\Gamma(d/2)}, \tag{29}$$

and $q_{\max}$ is the cutoff wavevector. Typically $q_{\max} \simeq \pi/\Delta x$, where Δx is the lattice size.

1.4 Spatial correlation

The spatial correlation function is defined to be:

$$C(\mathbf{r}, \mathbf{r}') = \langle \delta\phi(\mathbf{r})\delta\phi(\mathbf{r}')\rangle_s . \tag{30}$$

This measures the correlation of the density field at $\mathbf{r}$ and $\mathbf{r}'$. Substituting the definition of Fourier transform, we get:

$$C(\mathbf{r}, \mathbf{r}') = \frac{1}{V}\sum_{\mathbf{q},\mathbf{q}'} \left\langle \delta\phi_{\mathbf{q}}\delta\phi_{\mathbf{q}'}\right\rangle_s e^{i\mathbf{q}\cdot\mathbf{r}} e^{i\mathbf{q}'\cdot\mathbf{r}'}. \tag{31}$$

However, since we have translational symmetry, we must $C(\mathbf{r}, \mathbf{r}')$ only depends on $\mathbf{r} - \mathbf{r}'$, *i.e.*, $C(\mathbf{r}, \mathbf{r}') = C(\mathbf{r} - \mathbf{r}')$. Thus, $\left\langle \delta\phi_{\mathbf{q}}\delta\phi_{\mathbf{q}'}\right\rangle_s$ must have the following form:

$$\left\langle \delta\phi_{\mathbf{q}}\delta\phi_{\mathbf{q}'}\right\rangle_s = \left\langle |\delta\phi_{\mathbf{q}}|^2\right\rangle_s \delta_{\mathbf{q},-\mathbf{q}'} \tag{32}$$

so that

$$C(\mathbf{r} - \mathbf{r}') = \frac{1}{V}\sum_{\mathbf{q}} \underbrace{\left\langle |\delta\phi_{\mathbf{q}}|^2\right\rangle_s}_{S(\mathbf{q})} e^{i\mathbf{q}\cdot(\mathbf{r}-\mathbf{r}')} \tag{33}$$

is a function of $\mathbf{r} - \mathbf{r}'$ only. $S(\mathbf{q}) = \left\langle |\delta\phi_{\mathbf{q}}|^2\right\rangle_s$, which is the Fourier transform of $C(\mathbf{r})$, is called the structure factor.

For Gaussian statistics, the partition function can be written as:

$$\mathcal{Z} = \left(\prod_{\mathbf{q}} \int d\delta\phi_{\mathbf{q}}\right) e^{-\frac{1}{2}\sum_{\mathbf{q}} G(\mathbf{q})|\delta\phi_{\mathbf{q}}|^2} \tag{34}$$

Now consider:

$$\frac{1}{\mathcal{Z}}\frac{\partial \mathcal{Z}}{\partial G(\mathbf{q})} = -\frac{1}{2}\left(\prod_{\mathbf{q}} \int d\delta\phi_{\mathbf{q}}\right) |\delta\phi_{\mathbf{q}}|^2 \frac{1}{\mathcal{Z}} e^{-\frac{1}{2}\sum_{\mathbf{q}} G(\mathbf{q})|\delta\phi_{\mathbf{q}}|^2} \tag{35}$$

$$= -\frac{1}{2}\left(\prod_{\mathbf{q}} \int d\delta\phi_{\mathbf{q}}\right) |\delta\phi_{\mathbf{q}}|^2 P_s\{\delta\phi_{\mathbf{q}}\} \tag{36}$$

$$= -\frac{1}{2}\left\langle |\delta\phi_{\mathbf{q}}|^2\right\rangle . \tag{37}$$

Thus we obtain the formula for the structure factor from the partition function:

$$S(\mathbf{q}) = \left\langle |\delta\phi_{\mathbf{q}}|^2\right\rangle_s = -2\frac{\partial \ln \mathcal{Z}}{\partial G(\mathbf{q})}. \tag{38}$$

Using the expression for $\ln \mathcal{Z}$, we compute above, we can find:

$$S(\mathbf{q}) = \frac{\partial}{\partial G(\mathbf{q})} \sum_{\mathbf{q}'} \ln \left(\frac{G(\mathbf{q}')}{2\pi} \right) \tag{39}$$

$$= \frac{1}{G(\mathbf{q})} \tag{40}$$

$$= \frac{T}{a + 3b\phi_0^2 + \kappa q^2}. \tag{41}$$

1.5 Numerical simulation

Let's consider $d = 1$ system for now. The generalization to higher dimension is straightforward. The equation we are solving is:

$$\frac{\partial \phi}{\partial t} = M \frac{\partial^2}{\partial x^2} \left(\frac{\delta \mathcal{H}}{\delta \phi} \right) + \sqrt{2MT} \frac{\partial}{\partial x} \Lambda(x,t), \tag{42}$$

where $\Lambda(x,t)$ is Gaussian white noise with mean and variance:

$$\langle \Lambda(x,t) \rangle = 0 \tag{43}$$

$$\langle \Lambda(x,t)\Lambda(x',t') \rangle = \delta(x - x')\delta(t - t'). \tag{44}$$

In computer simulations, the space x is discretized into lattice with step size Δx:

$$x \to i\Delta x, \text{ where } i = 0, 1, 2, \ldots N_x - 1, \tag{45}$$

where N_x is the total number of lattice sites. The system size is then $L = N_x \Delta x$. Similarly, time is also discretized into:

$$t \to n\Delta t, \text{ where } n = 0, 1, 2, \ldots, N_t - 1, \tag{46}$$

where N_t is the total number of timesteps we are running the simulation for. Consequently, the density field and the noise current become:

$$\phi(x,t) \to \phi_i^n \tag{47}$$

$$\Lambda(x,t) \to \Lambda_i^n \tag{48}$$

Next we need to regularize the Dirac delta function:

$$\delta(x - x') \to \frac{\delta_{i,i'}}{\Delta x} \tag{49}$$

$$\delta(t - t') \to \frac{\delta_{n,n'}}{\Delta t}. \tag{50}$$

Thus we need to define a new noise:

$$\tilde{\Lambda}_i^n = \sqrt{\Delta x \Delta t} \Lambda_i^n, \tag{51}$$

so that the correlation for this new noise is just a Kronecker delta:

$$\left\langle \tilde{\Lambda}_i^n \tilde{\Lambda}_j^m \right\rangle = \delta_{mn}\delta_{ij}. \tag{52}$$

Recall the Hamiltonian functional:

$$\mathcal{H}[\phi] = \int_0^L dx \left\{ f(\phi) + \frac{\kappa}{2} \left(\frac{\partial \phi}{\partial x} \right)^2 \right\}, \tag{53}$$

where $f(\phi) = \frac{a}{2}\phi^2 + \frac{b}{4}\phi^4$. In discrete space, the gradient operator becomes:

$$\frac{\partial \phi}{\partial x} \to \frac{\phi_{i+1} - \phi_{i-1}}{2\Delta x} + \mathcal{O}(\Delta x^2). \tag{54}$$

Therefore, the Hamiltonian functional becomes:

$$\begin{aligned} \mathcal{H}[\phi] &\to \mathcal{H}\{\phi_i\} && (55) \\ &= \sum_{i=1}^{N_x - 1} \Delta x \left\{ f(\phi_i) + \frac{\kappa}{2} \left(\frac{\phi_{i+1} - \phi_{i-1}}{2\Delta x} \right)^2 \right\} && (56) \\ &= \sum_{i=1}^{N_x - 1} \Delta x \left\{ f(\phi_i) + \frac{\kappa}{8\Delta x^2} \left(\phi_{i+1}^2 - 2\phi_{i+1}\phi_{i-1} + \phi_{i-1}^2 \right) \right\} && (57) \end{aligned}$$

The functional derivative in discrete space is defined to be (for $d = 1$):

$$\begin{aligned} \frac{\delta \mathcal{H}}{\delta \phi} &\to \frac{1}{\Delta x} \frac{\partial \mathcal{H}}{\partial \phi_i} && (58) \\ &= \frac{\partial}{\partial \phi_i} \sum_{j=1}^{N_x - 1} \left\{ f(\phi_j) + \frac{\kappa}{8\Delta x^2} \left(\phi_{j+1}^2 - 2\phi_{j+1}\phi_{j-1} + \phi_{j-1}^2 \right) \right\} && (59) \\ &= \sum_{j=1}^{N_x - 1} \left\{ f'(\phi_j)\delta_{ij} + \frac{\kappa}{8\Delta x^2} \left(2\phi_{j+1}\delta_{i,j+1} - 2\phi_{j+1}\delta_{i,j-1} - 2\phi_{j-1}\delta_{i,j+1} + 2\phi_{j-1}\delta_{i,j-1} \right) \right\} && (60) \\ &= f'(\phi_i) + \frac{\kappa}{4\Delta x^2} (\phi_i - \phi_{i+2} - \phi_{i-2} + \phi_i) && (61) \\ &= f'(\phi_i) - \kappa \left(\frac{\phi_{i+2} - 2\phi_i + \phi_{i-2}}{4\Delta x^2} \right). && (62) \end{aligned}$$

Now if we recall the continuum version of functional derivative,

$$\frac{\delta \mathcal{H}}{\delta \phi} = f'(\phi) - \kappa \frac{\partial^2 \phi}{\partial x^2}, \tag{63}$$

the Laplacian operator should then be equal to:

$$\frac{\partial^2 \phi}{\partial x^2} \to \frac{\phi_{i+2} - 2\phi_i + \phi_{i-2}}{4\Delta x^2} + \mathcal{O}(\Delta x). \tag{64}$$

Notice that the second derivative skips a lattice site, compared to the first derivative.

Now putting everything together, the discretized dynamics has become:

$$\phi_i^{n+1} = \phi_i^n + \Delta t \, \frac{M}{\Delta x} \left(\frac{\partial \mathcal{H}/\partial \phi_{i+2}^n - \partial \mathcal{H}/\partial \phi_i^n + \partial \mathcal{H}/\partial \phi_{i-2}^n}{4\Delta x^2} \right) + \sqrt{\Delta t} \sqrt{\frac{2MT}{\Delta x}} \left(\frac{\tilde{\Lambda}_{i+1}^n - \tilde{\Lambda}_{i-1}^n}{2\Delta x} \right) \tag{65}$$

where $\{\tilde{\Lambda}_i^n\}$ are a set of independent Gaussian random variables with zero mean and unit variance. Taking the limit of continuous time, we can write the above equation as:

$$\frac{\partial \phi_i}{\partial t} = -\Gamma_{ij}\frac{\partial \mathcal{H}}{\partial \phi_j^n} + g_{ij}\tilde{\Lambda}_j, \tag{66}$$

where

$$\Gamma_{ij} = \frac{M}{4\Delta x^3}(2\delta_{i,j} - \delta_{i,j-2} - \delta_{i,j+2}) \tag{67}$$

$$g_{ij} = \sqrt{\frac{MT}{2\Delta x^3}}(\delta_{i,j-1} - \delta_{i,j+1}). \tag{68}$$

Now we can verify FDT

$$g_{ik}g_{jk} = \frac{MT}{2\Delta x^3}(\delta_{i,k-1} - \delta_{i,k+1})(\delta_{j,k-1} - \delta_{j,k+1}) \tag{69}$$

$$= \frac{MT}{2\Delta x^3}(\delta_{i,j} + \delta_{i,j} - \delta_{i,j+2} - \delta_{i,j-2}) \tag{70}$$

$$= 2\Gamma_{ij}T, \tag{71}$$

or $gg^T = g^T g = 2\Gamma T$.

For $d = 2$ dimension, the spatial coordinates are:

$$x \to i\Delta x, \text{ where } i = 0, 1, 2, \dots, N_x - 1 \tag{72}$$

$$y \to j\Delta y, \text{ where } j = 0, 1, 2, \dots, N_y - 1, \tag{73}$$

The discretized Langevin equation is:

$$\phi_{ij}^{n+1} = \phi_{ij} + \Delta t M \nabla^2 \mu_{ij}^n + \sqrt{\Delta t}\sqrt{\frac{2MT}{\Delta x \Delta y}}\nabla \cdot \Lambda_{ij}^n, \tag{74}$$

where the gradient and Laplacian operator are:

$$\frac{\partial \phi_{ij}}{\partial x} = \frac{\phi_{i+1,j} - \phi_{i-1,j}}{2\Delta x} + \mathcal{O}(\Delta x^2) \tag{75}$$

$$\frac{\partial \phi_{ij}}{\partial y} = \frac{\phi_{i,j+1} - \phi_{i,j-1}}{2\Delta y} + \mathcal{O}(\Delta y^2) \tag{76}$$

$$\nabla^2 \phi_{ij} = \frac{\phi_{i+2,j} - 2\phi_{ij} + \phi_{i-2,j}}{4\Delta x^2} + \frac{\phi_{i,j+2} - 2\phi_{ij} + \phi_{i,j-2}}{4\Delta y^2} + \mathcal{O}(\Delta x). \tag{77}$$

In Numpy, ϕ is represented as an array:

$$\phi = \begin{pmatrix} \phi_{00} & \phi_{01} & \cdots & \phi_{0,N_y-1} \\ \phi_{10} & \phi_{11} & & \phi_{1,N_y-1} \\ \vdots & & \ddots & \vdots \\ \phi_{N_x-1,0} & \phi_{N_x-1,1} & \cdots & \phi_{N_x-1,N_y-1} \end{pmatrix} \downarrow x\text{-direction} \tag{78}$$

$$\longrightarrow y\text{-direction} \tag{79}$$

Notice that the x and the y axis are transposed.

[14]:

```python
import numpy as np
import matplotlib.pyplot as plt

dx = 0.25  # lattice step size dx = dy
dt = 0.001  # time discretization
Nx, Ny = 256, 256  # system size Lx = Nx.dx and Ly = Ny.dy
Nt = 100000  # total number of timesteps

M = 1.0
a, b, kappa = 0.5, 1.0, 1.0
T = 0.1
phi0 = 0.5

# array of cartesian coordinates (needed for plotting)
x = np.arange(0, Nx)*dx
y = np.arange(0, Ny)*dx
y, x = np.meshgrid(y, x)

# method to calculate the laplacian
def laplacian(phi):
    # axis=0 --> roll along x-direction
    # axis=1 --> roll along y-direction
    laplacianphi = (np.roll(phi,+2,axis=0) - 2.0*phi + np.roll(phi,-2,axis=0))/
↪(4*dx*dx) \
                  + (np.roll(phi,+2,axis=1) - 2.0*phi + np.roll(phi,-2,axis=1))/
↪(4*dx*dx)
    return laplacianphi

# method to calculate the gradient
def diff_x(phi):
    diff_x_phi = (np.roll(phi,+1,axis=0) - np.roll(phi,-1,axis=0))/(2*dx)
    return diff_x_phi

def diff_y(phi):
    diff_y_phi = (np.roll(phi,+1,axis=1) - np.roll(phi,-1,axis=1))/(2*dx)
    return diff_y_phi

# update phi
def update(phi):
    # calculate noise: create an Nx by Ny matrix of random numberes
    Lambda_x = np.random.normal(0, 1, size=(Nx, Ny))
    Lambda_y = np.random.normal(0, 1, size=(Nx, Ny))

    # calculate mu
    mu = a*phi + b*phi*phi*phi - kappa*laplacian(phi)
    divLambda = diff_x(Lambda_x) + diff_y(Lambda_y)
```

```
    # update phi
    phi = phi + dt*M*laplacian(mu) + np.sqrt(2*M*T*dt/dx**2)*divLambda

    return phi

# plot phi
def plot(phi):
    # initialize figure and movie objects
    fig, ax = plt.subplots(figsize=(6,6))

    ax.set_title('$\phi(\mathbf{r})$', fontsize=14)
    ax.set_aspect('equal')
    ax.set_xlabel('$x$', fontsize=14)
    ax.set_ylabel('$y$', fontsize=14)
    ax.set_xlim(0, Nx*dx)
    ax.set_ylim(0, Ny*dx)
    ax.tick_params(axis='both', which='major', labelsize=12)

    colormap = ax.pcolormesh(x, y, phi, shading='auto', vmin = -1, vmax = 1)
    colorbar = plt.colorbar(colormap)
    colorbar.ax.tick_params(labelsize=12)

    plt.show()

# plot A(t)
def plot_A(dt, A):
    Nt = len(A)
    t = np.arange(0, Nt, 1)*dt

    fig, ax = plt.subplots(figsize=(6,4))

    ax.set_title('global average of $\delta\phi^2$', fontsize=14)
    ax.set_xlabel('$t$', fontsize=14)
    ax.set_ylabel('$A(t)$', fontsize=14)
    ax.tick_params(axis='both', which='major', labelsize=12)

    ax.plot(t, A)

    plt.show()
```

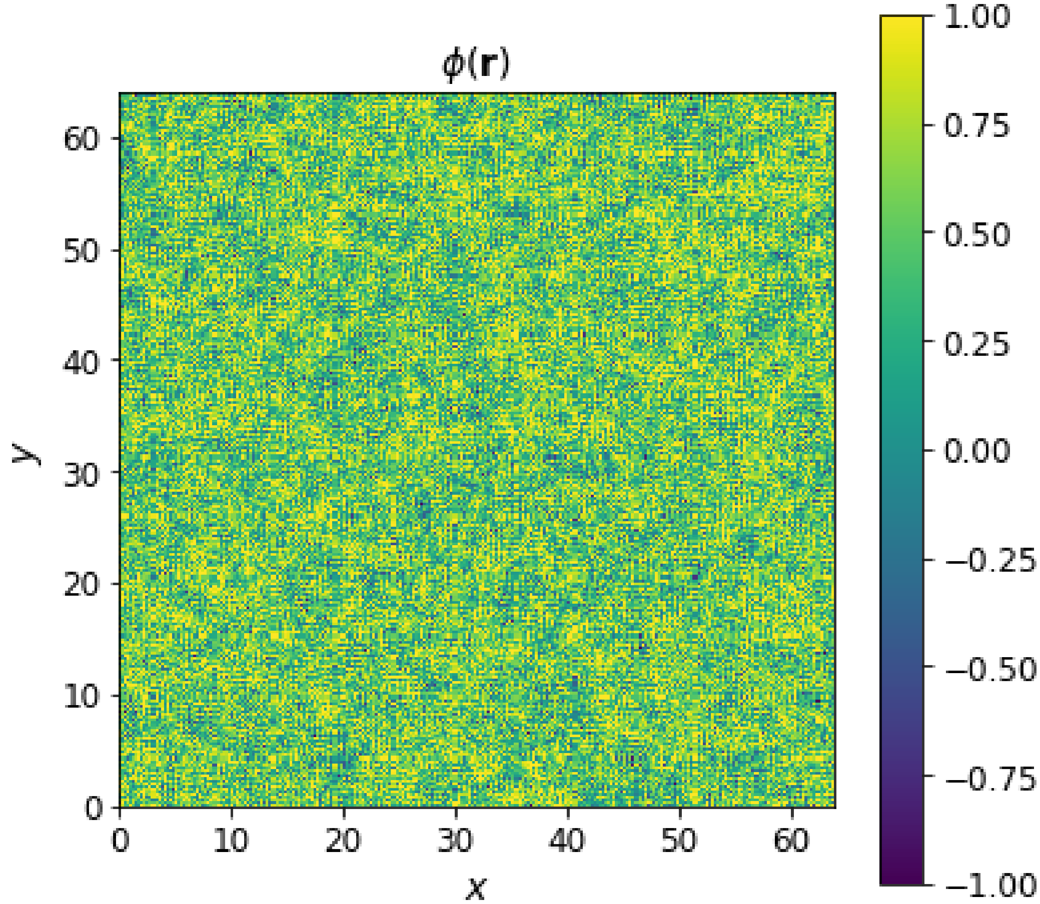
In simulation it might be useful to track some macroscopic quantity to check whether the simulation has reached a steady state or not. For instance, we may track the global fluctuations squared:

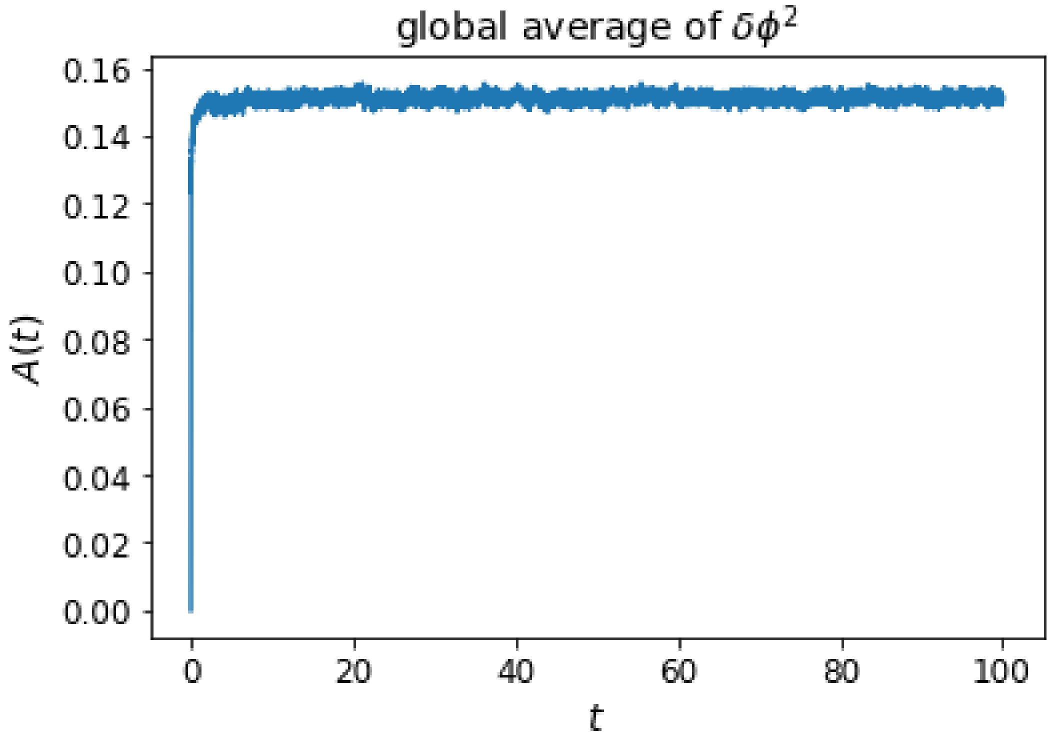
$$A(t) = \frac{1}{V}\int_V \delta\phi(\mathbf{r},t)^2 \, d\mathbf{r}. \tag{80}$$

[15]:
```
######################
# run the simulation #
```

```python
######################

# initialize phi
phi = np.ones((Nx, Ny))*phi0
A = np.zeros(Nt)

# loop for Nt timesteps for until system reaches steady state
for n in range(0, Nt, 1):
    dphi = phi - np.ones((Nx, Ny))*phi0
    A[n] = np.sum(dphi*dphi)/(Nx*Ny)
    if n % 10000 == 0:
        print(f't = {n*dt}')
    phi = update(phi)

plot(phi)
plot_A(dt, A)
```

```
t = 0.0
t = 10.0
t = 20.0
t = 30.0
t = 40.0
t = 50.0
t = 60.0
t = 70.0
t = 80.0
t = 90.0
```


$\phi(\mathbf{r})$
0
10
20
30
40
50
60
x
y
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00



1.6 Using fast Fourier transform in Numpy

The method to perform a 2-dimensional discrete Fourier transform on the array phi and save it to another array phi_q is:

```
phi_q = numpy.fft.fft2(phi, norm='ortho')
```

The discrete Fourier transform in Numpy is defined to be:

$$\phi(\mathbf{r}) = \frac{1}{\sqrt{N_x N_y}} \sum_{\mathbf{q}} \phi_{\mathbf{q}} e^{i\mathbf{q}\cdot\mathbf{r}} \quad \text{(inverse Fourier transform)} \tag{81}$$

$$\phi_{\mathbf{q}} = \frac{1}{\sqrt{N_x N_y}} \sum_{\mathbf{r}} \phi(\mathbf{r}) e^{-i\mathbf{q}\cdot\mathbf{r}} \quad \text{(forward Fourier transform).} \tag{82}$$

But we want:

$$\phi(\mathbf{r}) = \frac{1}{\sqrt{L_x L_y}} \sum_{\mathbf{q}} \phi_{\mathbf{q}} e^{i\mathbf{q}\cdot\mathbf{r}} \tag{83}$$

$$\phi_{\mathbf{q}} = \frac{1}{\sqrt{L_x L_y}} \underbrace{\sum_{\mathbf{r}} \Delta x \Delta y}_{\int d\mathbf{r}} \, \phi(\mathbf{r}) e^{-i\mathbf{q}\cdot\mathbf{r}}. \tag{84}$$

so we need to multiply the forward Fourier transform in Numpy by $\sqrt{\Delta x \Delta y}$ and divide the inverse Fourier transform in Numpy by $\sqrt{\Delta x \Delta y}$. Also note that the array ϕ_q is arranged in a peculiar way in Numpy:

$$\phi_q = \underbrace{\boxed{\phi_0 \mid \phi_{\frac{2\pi}{L}} \mid \phi_{\frac{2\pi(2)}{L}} \mid \cdots \mid \phi_{\frac{2\pi(N/2-1)}{L}} \mid \phi_{\frac{2\pi(-N/2)}{L}} \mid \phi_{\frac{2\pi(-N/2+1)}{L}} \mid \cdots \mid \phi_{\frac{2\pi(-1)}{L}}}}_{\text{total length}=N} \tag{85}$$

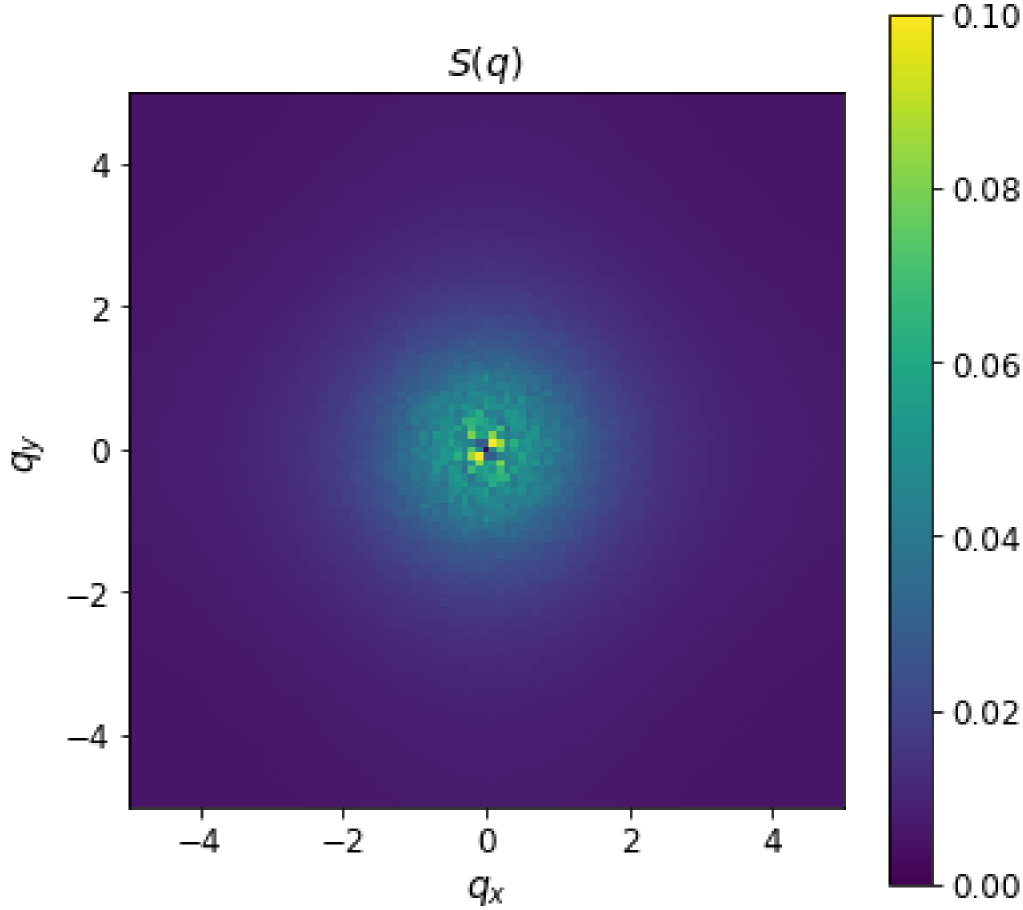
So we also need to shift each element of the array to the right by $N/2$.

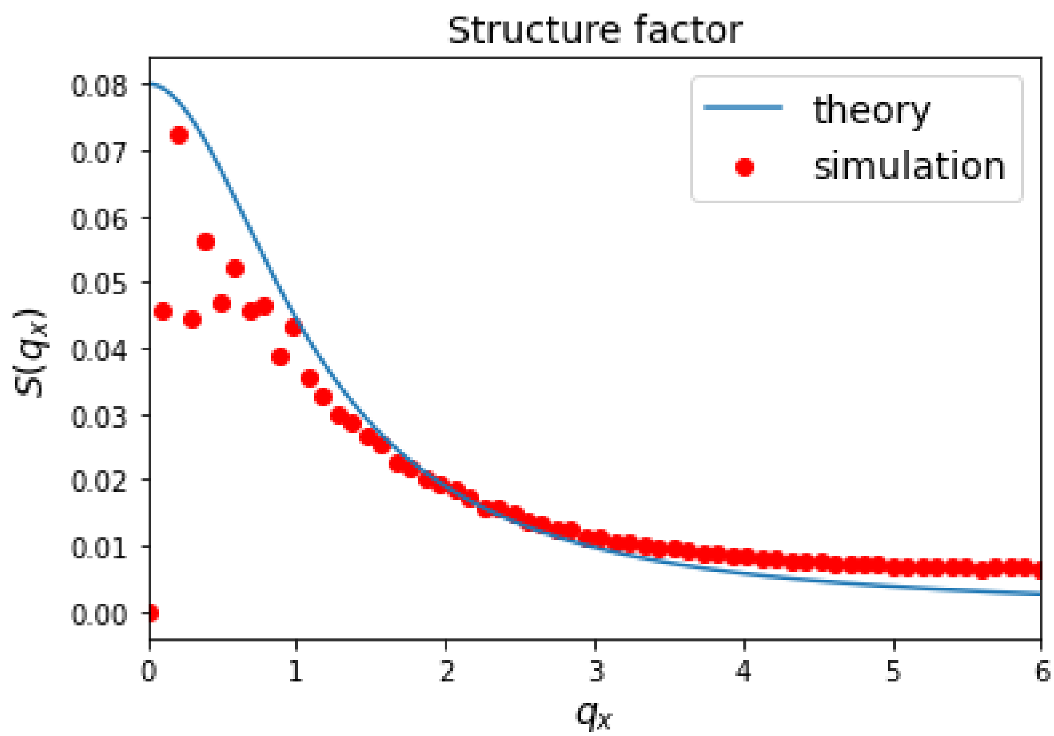
[16]:
```python
# method to plot S(q)
def plot_Sq(dx, dy, Sq):
    Nx = np.shape(Sq)[0]
    Ny = np.shape(Sq)[1]

    # define wavevector
    qx = 2*np.pi/(Nx*dx)*np.arange(-Nx/2,Nx/2,1)
    qy = 2*np.pi/(Ny*dy)*np.arange(-Ny/2,Ny/2,1)
    qy, qx = np.meshgrid(qy, qx)

    # contour plot of S(q)
    fig, ax = plt.subplots(figsize=(6,6))

    ax.set_title('$S(q)$', fontsize=14)
    ax.set_aspect('equal')
    ax.set_xlabel('$q_x$', fontsize=14)
    ax.set_ylabel('$q_y$', fontsize=14)
    ax.set_xlim(-5,5)
    ax.set_ylim(-5,5)
    ax.tick_params(axis='both', which='major', labelsize=12)

    colormap = ax.pcolormesh(qx, qy, Sq, shading='auto', vmin=0, vmax=0.1)
    colorbar = plt.colorbar(colormap)
    colorbar.ax.tick_params(labelsize=12)

    plt.show()

    # slice plot of S(q)
    fig, ax = plt.subplots(figsize=(6,4))

    ax.set_title('Structure factor', fontsize=14)
    ax.set_xlabel('$q_x$', fontsize=14)
    ax.set_ylabel('$S(q_x)$', fontsize=14)
    ax.set_xlim(0, 6)

    q = 2*np.pi/(Nx*dx)*np.arange(-Nx/2,Nx/2,1)
    q1 = 2*np.pi/(Nx*dx)*np.arange(-Nx/2,Nx/2,0.0001)
    Sq_theory = T/(a+3*b*phi0**2+kappa*q1**2)

    ax.scatter(q, Sq[:,int(Ny/2)], c='red', label='simulation')
    ax.plot(q1, Sq_theory, label='theory')
```

```
    plt.legend(fontsize=14)
    plt.show()
```

[17]:
```
##################################
# calculate the structure factor #
##################################

# calculate the time average <|dphi_q|^2>, or S(q)
Sq = np.zeros((Nx, Ny))

# loop again for Nt timesteps
for n in range(0, Nt, 1):
    dphi = phi - np.ones((Nx, Ny))*phi0
    dphi_q = np.fft.fft2(dphi, norm='ortho')*np.sqrt(dx*dx)
    Sq = Sq + np.real(dphi_q*np.conjugate(dphi_q))
    if n % 10000 == 0:
        print(f't = {n*dt}')
    phi = update(phi)

Sq = Sq/Nt

# needs to shift rows and columns in order before plotting
Sq = np.roll(Sq,+int(Nx/2),axis=0)
Sq = np.roll(Sq,+int(Ny/2),axis=1)

plot_Sq(dx, dx, Sq)
```

```
t = 0.0
t = 10.0
t = 20.0
t = 30.0
t = 40.0
t = 50.0
t = 60.0
t = 70.0
t = 80.0
t = 90.0
```


$S(q)$
q_y
q_x
4
2
0
−2
−4
−4
−2
0
2
4
0.10
0.08
0.06
0.04
0.02
0.00



1.7 Pseudo-spectral simulation

The equation we are solving is:

$$\frac{\partial \phi}{\partial t} = Ma\nabla^2\phi + Mb\nabla^2\phi^3 - M\kappa\nabla^4\phi + \sqrt{2MT}\nabla \cdot \Lambda \tag{86}$$

Taking Fourier transform, we get:

$$\frac{\partial \phi_{\mathbf{q}}}{\partial t} = -M\left(aq^2 + \kappa q^4\right)\phi_{\mathbf{q}} - Mbq^2\mathcal{F}[\phi^3]_{\mathbf{q}} + \sqrt{2MT}i\mathbf{q} \cdot \Lambda_{\mathbf{q}}. \tag{87}$$

The algorithm will be:

$$\begin{array}{c|c|c} \text{Real space} & \longleftrightarrow & \text{Fourier space} \\ \hline \text{calculate } \phi,\ \phi^3 \text{ and } \nabla\cdot\Lambda & \underset{\text{forward FFT}}{\longrightarrow} & \phi_{\mathbf{q}},\ \mathcal{F}[\phi^3]_{\mathbf{q}} \text{ and } \mathcal{F}[\nabla\cdot\Lambda]_{\mathbf{q}} \\ & & \text{update} \downarrow \text{timestep} \\ \text{get } \phi \text{ at the next timestep} & \underset{\text{inverse FFT}}{\longleftarrow} & \phi_{\mathbf{q}} \text{ at the next timestep} \end{array} \tag{88}$$

Recall that Fourier transform array in Numpy is arranged in a slightly peculiar way. So for this code, we have defined the q-vector to be:

$$q = \underbrace{\boxed{0 \,\Big|\, \tfrac{2\pi}{L} \,\Big|\, \tfrac{2\pi(2)}{L} \,\Big|\, \dots \,\Big|\, \tfrac{2\pi(N/2-1)}{L} \,\Big|\, \tfrac{2\pi(-N/2)}{L} \,\Big|\, \tfrac{2\pi(-N/2+1)}{L} \,\Big|\, \dots \,\Big|\, \tfrac{2\pi(-1)}{L}}}_{\text{total length}=N} \tag{89}$$

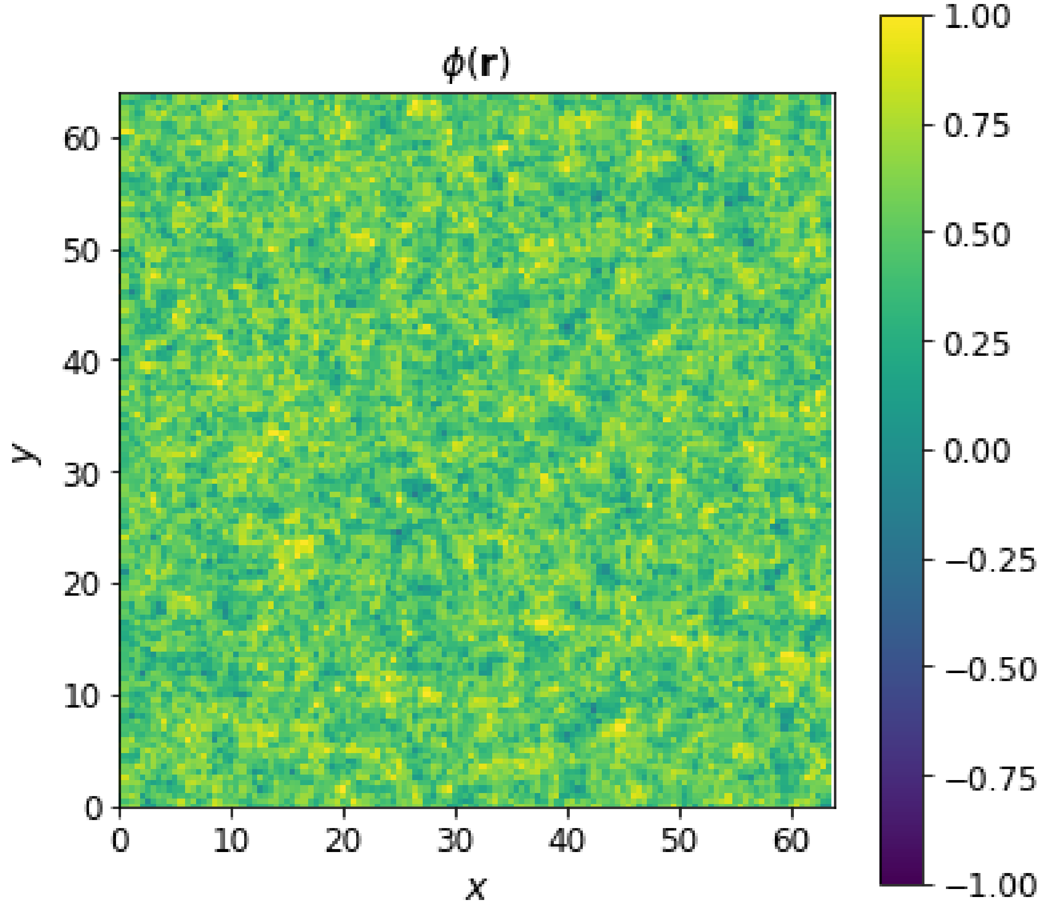
[18]:
```
import numpy as np
import matplotlib.pyplot as plt

dx = 0.5  # lattice step size dx = dy
dt = 0.0001  # time discretization
Nx, Ny = 128, 128  # system size Lx = Nx.dx and Ly = Ny.dy
Nt = 1000000  # total number of timesteps

M = 1.0
a, b, kappa = 0.5, 1.0, 1.0
T = 0.1
phi0 = 0.5

# array of cartesian coordinates (needed for plotting)
x = np.arange(0, Nx)*dx
y = np.arange(0, Ny)*dx
y, x = np.meshgrid(y, x)

# define wavevector
qx = 2*np.pi/(Nx*dx)*np.concatenate((np.arange(0, Nx/2, 1), np.arange(-Nx/2, 0,␣
↪1)))
qy = 2*np.pi/(Ny*dx)*np.concatenate((np.arange(0, Ny/2, 1), np.arange(-Ny/2, 0,␣
↪1)))
qy, qx = np.meshgrid(qy, qx)
q2 = qx*qx + qy*qy
q4 = q2*q2

# update phi
def update_pseudo_spectral(phi):
    # calculate noise: create an Nx by Ny matrix of random numberes
    Lambda_x = np.random.normal(0, 1, size=(Nx, Ny))
    Lambda_y = np.random.normal(0, 1, size=(Nx, Ny))

    # Fourier transform phi, phi^3, and Lambda
    phi_q = np.fft.fft2(phi, norm='ortho')*np.sqrt(dx*dx)
    phi_cube_q = np.fft.fft2(phi*phi*phi, norm='ortho')*np.sqrt(dx*dx)
    Lambda_x_q = np.fft.fft2(Lambda_x, norm='ortho')*np.sqrt(dx*dx)
    Lambda_y_q = np.fft.fft2(Lambda_y, norm='ortho')*np.sqrt(dx*dx)

    # update phi in Fourier
    phi_q = phi_q - dt*M*(a*q2 + kappa*q4)*phi_q \
                  - dt*M*b*q2*phi_cube_q \
                  + np.sqrt(2*M*T*dt/dx**2)*complex(0,1)*(qx*Lambda_x_q +␣
↪qy*Lambda_y_q)

    # inverse Fourier transform to get back phi in real space
    phi = np.real(np.fft.ifft2(phi_q, norm='ortho'))/np.sqrt(dx*dx)
```

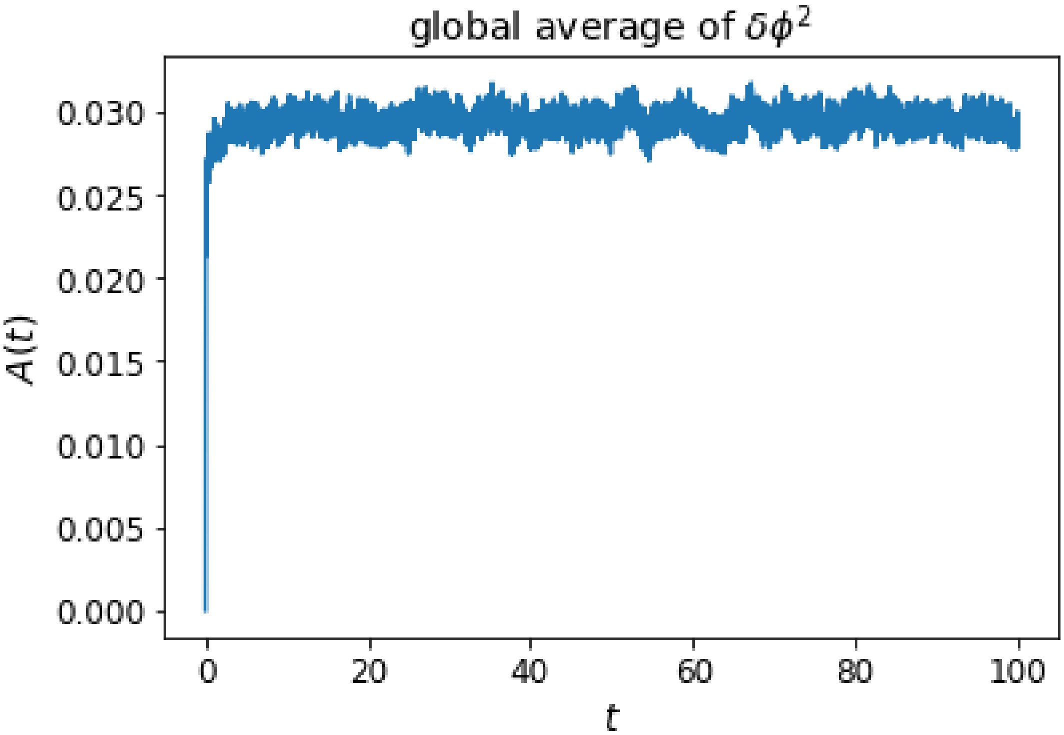
```python
    return phi, phi_q

# plot phi
def plot(phi):
    # initialize figure and movie objects
    fig, ax = plt.subplots(figsize=(6,6))

    ax.set_title('$\phi(\mathbf{r})$', fontsize=14)
    ax.set_aspect('equal')
    ax.set_xlabel('$x$', fontsize=14)
    ax.set_ylabel('$y$', fontsize=14)
    ax.set_xlim(0, Nx*dx)
    ax.set_ylim(0, Ny*dx)
    ax.tick_params(axis='both', which='major', labelsize=12)

    colormap = ax.pcolormesh(x, y, phi, shading='auto', vmin=-1, vmax=1)
    colorbar = plt.colorbar(colormap)
    colorbar.ax.tick_params(labelsize=12)

    plt.show()
```

[19]:
```python
##############################
# pseudo-spectral simulation #
##############################

# initialize phi
phi = np.ones((Nx, Ny))*phi0
phi_q = np.zeros((Nx, Ny))

A = np.zeros(Nt)

# loop for Nt timesteps for equilibriation
for n in range(0, Nt, 1):
    dphi = phi - np.ones((Nx, Ny))*phi0
    A[n] = np.sum(dphi*dphi)/(Nx*Ny)
    if n % 100000 == 0:
        print(f't = {n*dt}')
    phi, phi_q = update_pseudo_spectral(phi)

plot(phi)
plot_A(dt, A)
```

```
t = 0.0
t = 10.0
t = 20.0
t = 30.0
t = 40.0
```

```
t = 50.0
t = 60.0
t = 70.0
t = 80.0
t = 90.0
```

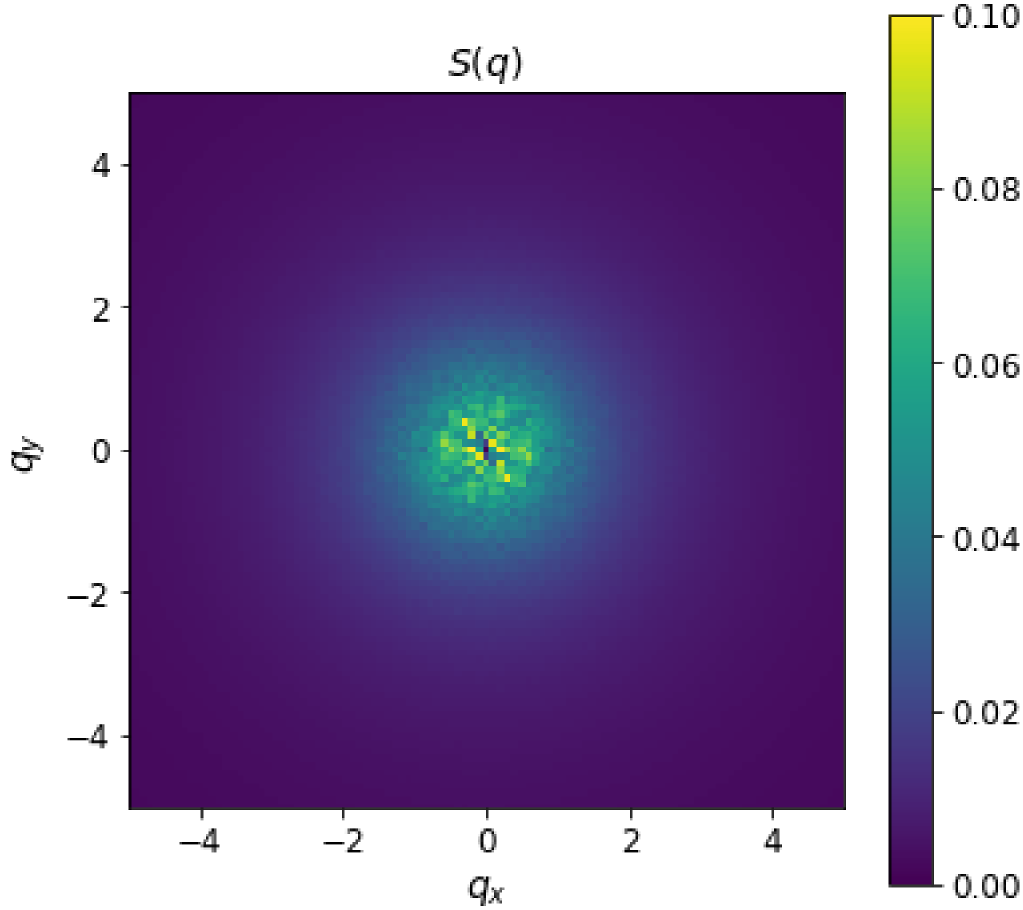


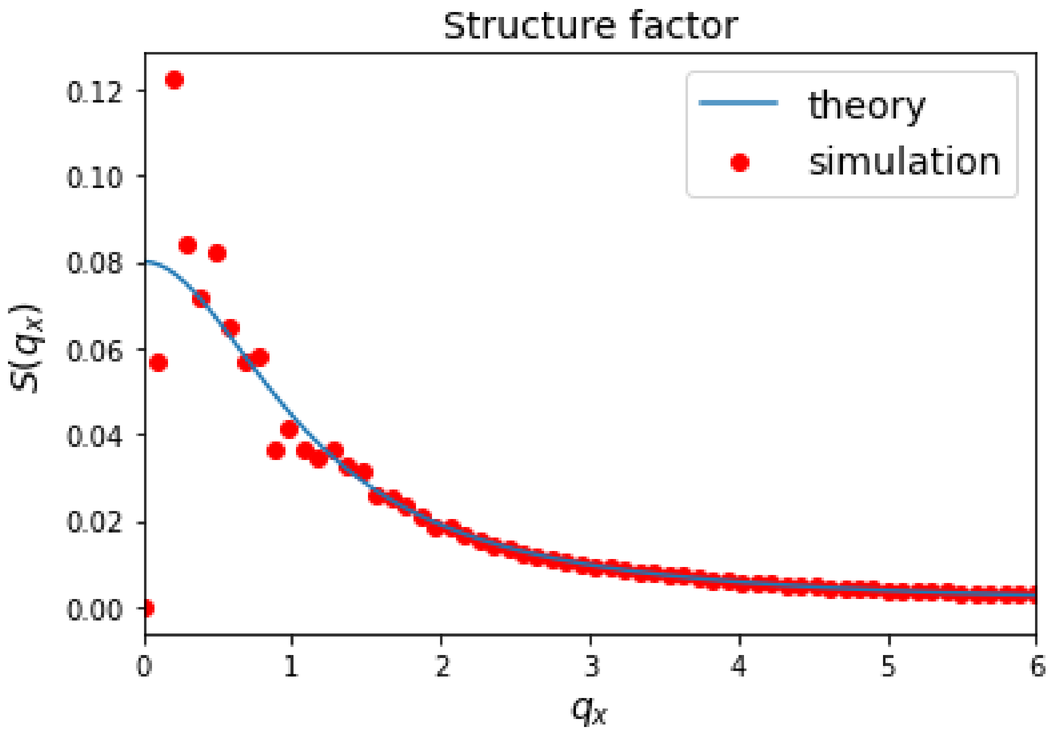


[20]:
```
####################################
# calculate the structure factor #
####################################

# define structure factor
Sq = np.zeros((Nx, Ny))
phi_q = np.fft.fft2(phi, norm='ortho')*np.sqrt(dx*dx)

# loop again for Nt timesteps
for n in range(0, Nt, 1):
    Sq = Sq + np.real(phi_q*np.conjugate(phi_q))
    if n % 100000 == 0:
        print(f't = {n*dt}')
    phi, phi_q = update_pseudo_spectral(phi)

Sq = Sq/Nt

# set q=0 mode to zero since this corresponds to phi=constant
Sq[0,0] = 0

# shift rows and columns to make S(q) in order
Sq = np.roll(Sq,+int(Nx/2),axis=0)
```

```
Sq = np.roll(Sq,+int(Ny/2),axis=1)

plot_Sq(dx, dx, Sq)
```

```
t = 0.0
t = 10.0
t = 20.0
t = 30.0
t = 40.0
t = 50.0
t = 60.0
t = 70.0
t = 80.0
t = 90.0
```





Structure factor
theory
simulation
0.12
0.10
0.08
0.06
0.04
0.02
0.00
$S(q_x)$
0
1
2
3
4
5
6
q_x


```
[ ]:
```