Discrete Fourier Transform

Hsi-Yu Schive (薛熙于)
National Taiwan University

Fourier Series

- **Assuming** $f(x)$ **has a period of** L

$$f(x) = \sum_{n=0}^{\infty} \left[a_n \cos(k_n x) + b_n \sin(k_n x)\right]$$

$$k_n = \frac{2\pi n}{L}$$

$$a_0 = \frac{1}{L}\int_0^L f(x)dx$$

$$a_n = \frac{2}{L}\int_0^L f(x)\cos(k_n x)dx, \quad n > 1$$

$$b_n = \frac{2}{L}\int_0^L f(x)\sin(k_n x)dx$$

alternative $\longleftrightarrow$

$$f(x) = \sum_{n=-\infty}^{\infty} c_n e^{ik_n x}$$

$$c_n = \frac{1}{L}\int_0^L f(x)e^{-ik_n x}dx = \frac{1}{L}\int_{-L/2}^{L/2} f(x)e^{-ik_n x}dx$$

$$c_0 = a_0$$

$$c_n = \tfrac{1}{2}(a_n - ib_n)\,, \quad n > 0$$

$$c_{-n} = \tfrac{1}{2}(a_n + ib_n)\,, \quad n > 0$$

Fourier Transform

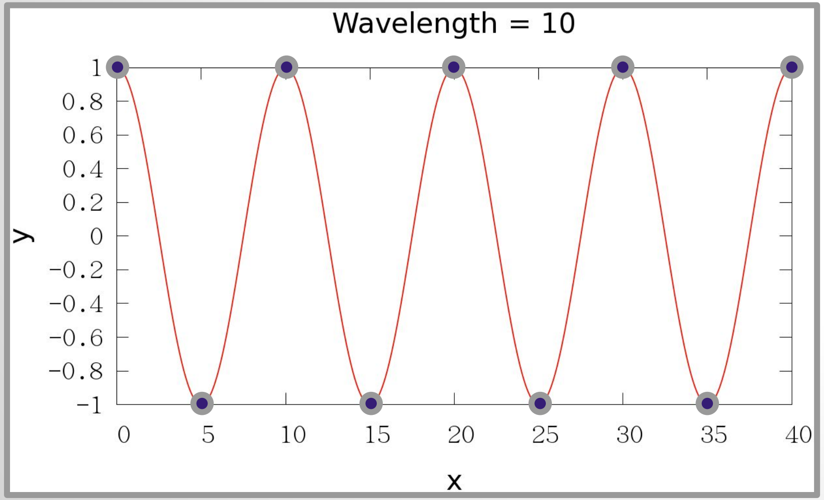
- **Let** $L \rightarrow \infty$, $k_n \rightarrow k$:

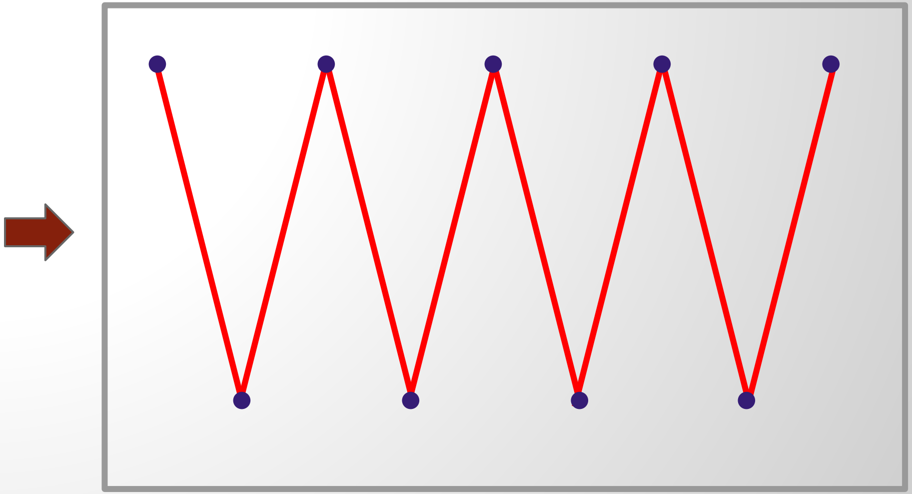
$$\boxed{\begin{aligned} f(x) &= \frac{1}{2\pi}\int_{-\infty}^{\infty} F(k)e^{ikx}\,dk \\ F(k) &= \int_{-\infty}^{\infty} f(x)e^{-ikx}\,dx \end{aligned}}$$

  - $|F(k)|$ **represents the amplitudes of different Fourier modes**
    - $|F(k)|^2 \rightarrow$ **power spectrum**
  - **Properties**
    - **If** $f(x)$ **is real** $\rightarrow F(-k) = F(k)^*$ **(Hermitian)** $\rightarrow |F(-k)| = |F(k)|$
    - **If** $f(x)$ **is even/odd** $\rightarrow F(k)$ **is even/odd**
    - **Linearity:** $af(x) + bg(x) \rightarrow aF(k) + bG(k)$
    - **Translation:** $f'(x)=f(x-\Delta x) \rightarrow F'(k)=F(k)e^{-ik\Delta x} \rightarrow |F'(k)|=|F(k)|$
    - **DC (average):** $F(0) = \int_{-\infty}^{\infty} f(x)dx$
    - **Parseval's theorem:** $\int_{-\infty}^{\infty} |f(x)|^2 dx = \int_{-\infty}^{\infty} |F(k)|^2 dk$

Nyquist Frequency

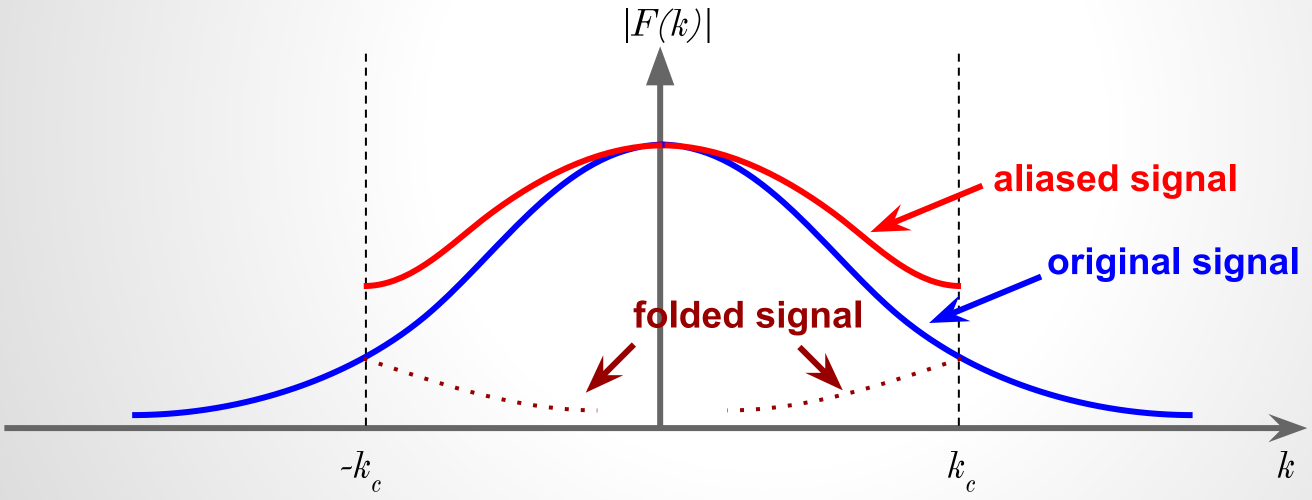
- Assuming a finite and fixed sampling interval Δ with N consecutive samples: $x_m = m\Delta,\ m=0,\ 1,\ 2,\ \dots,\ (N\text{-}1)$
- Nyquist critical frequency: $f_c=1/(2\Delta) \rightarrow k_c=2\pi f_c=\pi/\Delta$
  - The shortest wavelength resolved by a sampling interval Δ is 2Δ
  - k_c is the corresponding (highest) Fourier mode





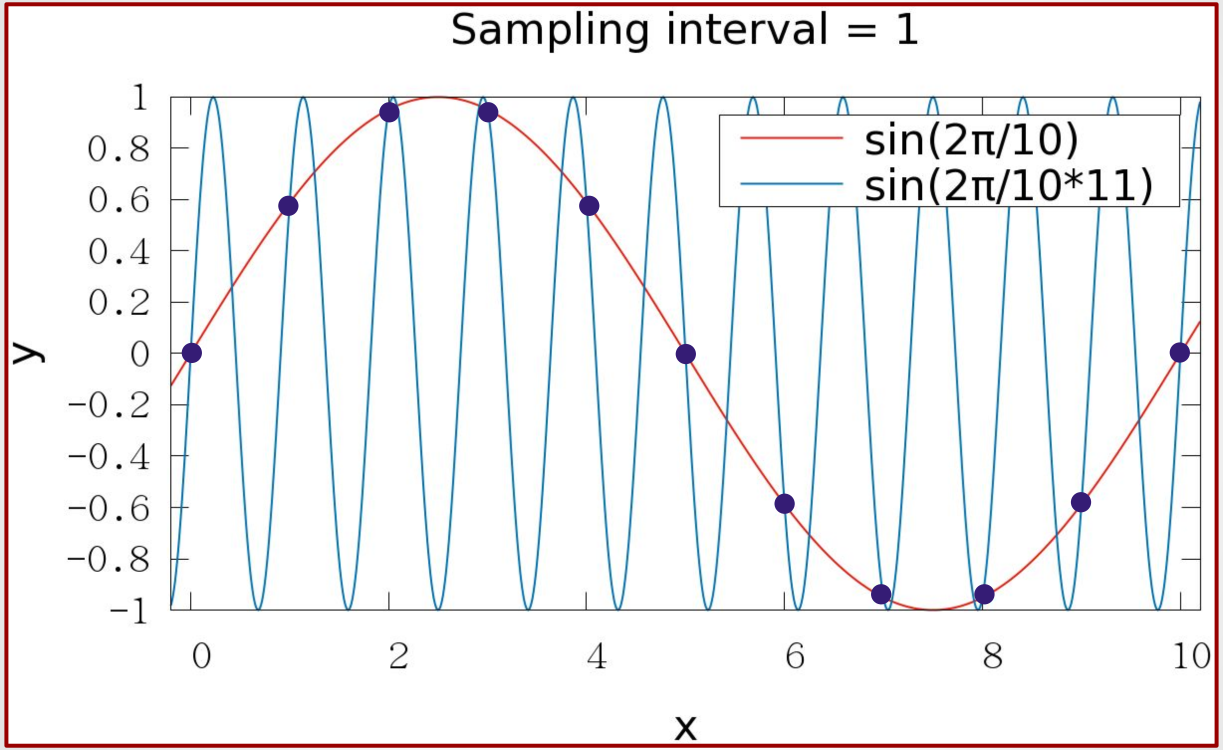
Nyquist Frequency

- **Aliasing: For a sampling interval Δ, the two Fourier modes give exactly the same samples if** $k_1 - k_2 = (2\pi/\Delta)m = 2k_c m,\ m \in Z$
  - **Solution: ensure power spectrum around k_c is negligible**



Assuming signal $f(x)$ is real here $\rightarrow$ $|F(k)|$ is symmetric

Nyquist Frequency Example



From the given sampling points, one CANNOT determine whether the real signals are red (low-k) or blue (high-k) lines!

Discrete Fourier Transform (DFT)

- **Only includes frequencies within the Nyquist frequency:** $-k_c \leqslant k \leqslant k_c$

$$f(x_m) = \sum_{n=-N/2+1}^{N/2} c_n e^{ik_n x_m} = \sum_{n=0}^{N-1} c_n e^{ik_n x_m}$$

$c_{-N/2} = c_{N/2}$ $\qquad$ $c_{-n} = c_{-n+N}$

$$\sum_{m=0}^{N-1} e^{ik_{n1}x_m} e^{ik_{n2}x_m} = N\delta_{n1,n2}, \quad x_m = m\Delta, k_n = 2\pi n/(N\Delta), F_n \equiv c_n$$

derive it!

$$\boxed{\begin{cases} f_m = \sum_{n=0}^{N-1} F_n e^{i2\pi mn/N} \\ F_n = \frac{1}{N}\sum_{m=0}^{N-1} f_m e^{-i2\pi mn/N} \end{cases}}$$

DFT

Discrete Fourier Transform

- **Convention:** F_n **with** $n=0, 1, 2, ..., N\text{-}1$. **But remember** $F_n = F_{n-N}$
  - $n=0$: **DC mode**
  - $1 \leqslant n \leqslant N/2\text{-}1$: **positive frequency**
  - $N/2+1 \leqslant n \leqslant N\text{-}1 \rightarrow \text{-}N/2+1 \leqslant n \leqslant \text{-}1$: **negative frequency**
  - $n=N/2$: **Nyquist frequency**
- **Normalization** $1/N$: $DFT^{-1}(\ DFT(f_m)\ ) = f_m$
- **Discrete power spectrum:** $|F_n|^2$
- **Symmetry properties: same as Fourier transform**
  - **If** f_m **is real** $\rightarrow F_{-n} = F_n^* \rightarrow |F_{-n}| = |F_n| \rightarrow$ **power spectrum is symmetric**
  - **If** f_m **is even/odd** $\rightarrow F_n$ **is even/odd (here even/odd means** $f_m = \pm f_{N-m}$**)**
- **Parseval's theorem:** $\displaystyle\sum_{m=0}^{N-1} |f_m|^2 = N \sum_{n=0}^{N-1} |F_n|^2$

DFT on Real Data

- If f_m are <u>real</u> instead of complex data
  - Only N degree of freedom (DoF)
  - Half of the data in F_n, $n=0, ..., N\text{-}1$ must be redundant
- Recall that (i) $F_n=F_{n\text{-}N}$ and (ii) $F_{\text{-}n} = F_n{}^*$ if f_m is real
  - $F_0=F_0{}^* \rightarrow Im[F_0]=0 \rightarrow$ DoF: 1
    - DC element is purely real
  - If N is even
    - $F_{N/2+t}=F_{\text{-}N/2+t}=F_{N/2\text{-}t}{}^*,\ t=1, ..., N/2\text{-}1 \rightarrow$ DoF: N-2
    - $F_{N/2}=F_{\text{-}N/2}=F_{N/2}{}^* \rightarrow Im[F_{N/2}]=0 \rightarrow$ DoF: 1
      - Nyquist frequency element is purely real
  - If N is odd
    - Let $N/2$ be the integer (truncated) division $\rightarrow 7/2=3$
    - $F_{N/2+t}=F_{N/2+t\text{-}N}=F_{\text{-}N/2\text{-}1+t}=F_{N/2+1\text{-}t}{}^*,\ t=1, ..., N/2 \rightarrow$ DoF: N-1
  - So there are exactly N redundant data in F_n

DFT on Real Data

- **Example**
  - **N=8:** $Re[F_0], F_1, F_2, F_3, Re[F_4], \underbrace{(F_5, F_6, F_7)}_{= F_3^*, F_2^*, F_1^*}$
  - **N=7:** $Re[F_0], F_1, F_2, F_3, \underbrace{(F_4, F_5, F_6)}_{= F_3^*, F_2^*, F_1^*}$
- **In practice, DFT libraries usually do not store these redundant data**
  - **But be aware that libraries such as *numpy FFT* and *FFTW* (see the Reference page) still store** $Im[F_0]$ **and** $Im[F_{N/2}]$
    - **Advantage: complex-to-complex and complex-to-real transforms have similar data structure**
    - **Disadvantage: input and output arrays are of different sizes**
      - **Be careful about the *in-place* transform**

Multi-Dimensional DFT

- d-dimensional DFT

$$
\begin{cases}
f_{m_1,m_2,\ldots m_d} = \displaystyle\sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1}\cdots\sum_{n_d=0}^{N_d-1} F_{n_1,n_2,\ldots,n_d}\, e^{i2\pi m_1 n_1/N_1}\, e^{i2\pi m_2 n_2/N_2}\ldots e^{i2\pi m_d n_d/N_d} \\[2em]
F_{n_1,n_2,\ldots,n_d} = \displaystyle\sum_{m_1=0}^{N_1-1}\sum_{m_2=0}^{N_2-1}\cdots\sum_{m_d=0}^{N_d-1} f_{m_1,m_2,\ldots,m_d}\, e^{-i2\pi m_1 n_1/N_1}\, e^{-i2\pi m_2 n_2/N_2}\ldots e^{-i2\pi m_d n_d/N_d} \\
\qquad\qquad \cdot \left[\dfrac{1}{N_1 N_2 \ldots N_d}\right]
\end{cases}
$$

- **If f is real** $\rightarrow$ $F_{-n_1,-n_2,\ldots,-n_d} = F^*_{n_1,n_2,\ldots,n_d}$

  $= F_{N_1-n_1,N_2-n_2,\ldots,N_d-n_d}$ (Hermitian)

  - Again, half of the data are redundant

Fast Fourier Transform (FFT)

- Computational complexity of the brute-force DFT: $O(N^2)$
  - For each k mode, simply perform an integration over N sampling points
- FFT: reduce the complexity to $O(N\ log_2 N)$
  - For example, check out the Danielson-Lanczos lemma
- Example libraries
  - Fastest Fourier Transform in the West (FFTW): http://www.fftw.org
  - Intel Math Kernel Library (MKL): https://software.intel.com/en-us/mkl-developer-reference-c-fft-functions
  - numpy FFT: https://docs.scipy.org/doc/numpy/reference/generated/numpy.fft.fft.html
  - cuFFT (FFT on GPU): https://developer.nvidia.com/cufft

DFT for Diffusion Eq.

- **Diffusion eq.** $\dfrac{\partial u(x,t)}{\partial t} = D\dfrac{\partial^2 u(x,t)}{\partial x^2}$
- **Fourier transform:** $\nabla \to i\boldsymbol{k}, u(x,t) \to U(k,t)$

$$\frac{\partial U(k,t)}{\partial t} = -Dk^2 U(k,t)$$

$$U(k,t+\Delta t) \approx U(k,t) - \Delta t D k^2 U(k,t)$$

$$u(x,t+\Delta t) = FT^{-1}(U(k,t+\Delta t))$$

```
#     DFT
      uk = np.fft.rfft( u )
#     compute the wavenumber (actually it needs to be computed just once)
      k  = 2.0*np.pi*np.fft.rfftfreq( N, dx )
#     inverse DFT
      u  = np.fft.irfft( uk - dt*D*k**2.0*uk )
```

lec05-demo01.py

Exercise

- **Use DFT to determine the amplitudes of different Fourier modes**
  - $f(x) = A + Bsin(k_1x) + Ccos(k_2x) \rightarrow$ **Given** $f(x)$**, determine** (A, B, C, k_1, k_2)
  - **Refer to the slide of "Fourier Series"**

```
# set the input data
x  = np.arange( 0.0, L, dx )   # x=L/N*n, n=0,1,...,N-1
u1 = amp1*np.cos( k1*x )       # cosine component
u2 = amp2*np.sin( k2*x )       # sine component
dc = np.ones(x.size)*dc        # DC component
u  = u1 + u2 + dc              # overall

# compute the coefficients of all sin() and cos() using np.fft.rfft
...
```

lec05-exercise01-template.py

DFT for Convolution

- Convolution of two functions f and g

$$\boxed{(f * g)(t) \equiv \int_{-\infty}^{\infty} f(t-\tau)g(\tau)d\tau}$$

  - Weighted average of f at t, where g is the weighting function
  - Convolution can be used for various purposes by choosing different weighting functions
    - E.g., smoothing (blurring), sharpening, edge detection, embossing
    - Google “convolution matrix/filter/kernel”
  - Commutativity: $f * g = g * f$
- Convolution theorem

$$\boxed{f * g = FT^{-1}(FT(f) \cdot FT(g))}$$ ← *derive it*

  - Reduce the computational complexity from $O(N^2)$ to $O(N\ log2N)$ using FFT

Image Processing with Convolution

Original

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Sharpening

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Blurring

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Edge detection

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

credit: wikipedia

Convolution Demo

```python
# set the input data
x = np.arange( 0.0, L, dx ) + 0.5*dx   # cell-centered coordinates
u = amp1*np.sin( 2.0*np.pi/lambda1*x ) + amp2*np.sin( 2.0*np.pi/lambda2*x )

# define a convolution filter
f  = np.array( [1.0, 2.0, 4.0, 2.0, 1.0] )
f /= f.sum()                          # normalization
f_pad0 = np.zeros( u.size )   # zero-padded filter
f_pad0[ 0:f.size ] = f
f_pad0 = np.roll( f_pad0, -(f.size//2) )# [0.4, 0.2, 0.1, 0.0, ..., 0.0, 0.1, 0.2]

# convolution
uk = np.fft.rfft( u )
fk = np.fft.rfft( f_pad0 )
u_con = np.fft.irfft( uk*fk )
```

lec05-demo02.py

Exercise

- Apply a sharpening convolution filter to lec05-demo02.py
- Apply an edge detection convolution filter to a step function

DFT for Self-Gravity with Periodic BC

- **Poisson eq. in 1D:** $\dfrac{\partial^2 \phi}{\partial x^2} = \rho$
- **Fourier transform:** $\partial/\partial x \to ik, \phi(x) \to \Phi(k), \rho(x) \to D(k)$

$$\Phi(k) = -\frac{D(k)}{k^2} \quad \to \quad \phi(x) = FT^{-1}(\Phi(k))$$

- **Alternatively, we can discretize the Poisson eq. first and then use DFT to compute its *exact* solution**

$$(\phi_{m+1} - 2\phi_m + \phi_{m-1}) = \Delta^2 \rho_m$$

$$\phi_m = \sum_{n=0}^{N-1} \Phi_n e^{i2\pi mn/N}, \quad \rho_m = \sum_{n=0}^{N-1} D_n e^{i2\pi mn/N}$$

For mode n : $\Phi_n e^{i2\pi mn/N}(e^{i2\pi n/N} - 2 + e^{-i2\pi n/N}) = \Delta^2 D_n e^{i2\pi mn/N}$

$$\Phi_n = \frac{-\Delta^2 D_n}{4\sin^2(\pi n/N)} \quad \to \quad \phi_m = FT^{-1}(\Phi_n)$$

  - **Pros: consistent discretization on root and refinement levels in AMR**

DFT for Self-Gravity with Isolated BC

- **Continuous case:**

$\nabla^2\phi = 4\pi G\rho$ with $\phi(r \to \infty) \to 0$

$$\phi(r) = -G\int \frac{\rho(\boldsymbol{r}')}{|\boldsymbol{r}-\boldsymbol{r}'|}d^3\boldsymbol{r}' = -G\,\underline{\left(\rho * r^{-1}\right)(r)}$$

convolution of $\rho(r)$ and the Green's function r^{-1}

- **Discrete case:**
  - Let $M_{ijk}=\rho_{ijk}\Delta h^3$ be the mass of cell (i,j,k) and treat each cell as a point mass

$$\phi_{i,j,k} = -G\sum_{m=0}^{N_x-1}\sum_{n=0}^{N_y-1}\sum_{l=0}^{N_z-1}\frac{M_{m,n,l}}{|\boldsymbol{r}_{i,j,k}-\boldsymbol{r}_{m,n,l}|}$$

~ exclude (m,n,l)=(i,j,k)

  - Question: how to use the discrete (periodic) convolution to solve it?

$$(f*g)_m = \sum_{i=0}^{N-1} f_i g_{m-i} = N\cdot DFT^{-1}(DFT(f)\cdot DFT(g))$$

~ 1D example

assuming periodicity: $g_{m\text{-}i} = g_{m\text{-}i+N}$

depending on the DFT normalization

DFT for Self-Gravity with Isolated BC

- **Zero padding** on the mass array: $M_m \rightarrow M'_m$ $(m=0, \ldots, 2N\text{-}1)$

m	0	1	2	...	N-1	N	N+1	...	2N-2	2N-1
M_m'	M_0	M_1	M_2	...	M_{N-1}	0	0	...	0	0

- Define a **symmetric discrete Green's function**: R_m

m	0	1	2	...	N-1	N	N+1	...	2N-2	2N-1
R_m	0	-1/1	-1/2	...	-1/(N-1)	x	-1/(N-1)	...	-1/2	-1/1

self force

arbitrary

$$\phi_m = \frac{G}{\Delta h} \sum_{i=0}^{2N-1} R_i M'_{m-i} = \frac{G}{\Delta h} (R * M')_m = \frac{G}{\Delta h} 2N \cdot DFT^{-1}(DFT(R) \cdot DFT(M'))$$

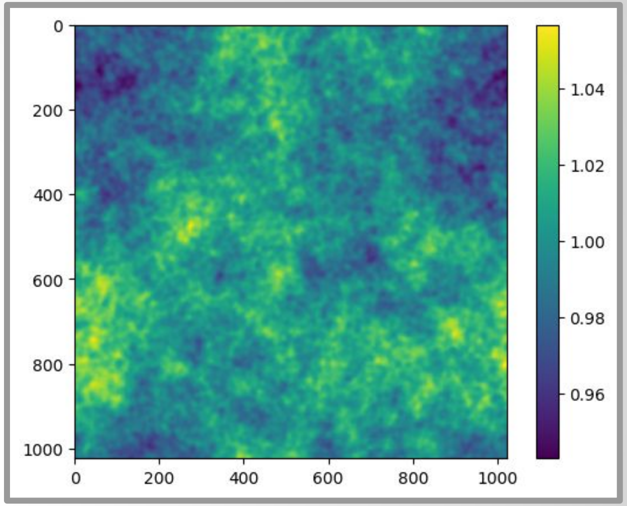
DFT for Self-Gravity with Isolated BC

- **Properties**
  - R_N **can be set arbitrarily since the greatest distance between any two cells is** $N\text{-}1$ **cells**
  - **Results should be exactly the same as computing the pairwise potential among all cells directly (i.e., direct N-body)**
  - $DFT(R)$ **only needs to be computed once**
  - R_0 **is related to the potential from the mass <u>within the same cell</u>**
    - $R_0=0 \rightarrow$ **ignore self force** $\rightarrow$ **consistent with point mass**
    - $R_0\neq 0 \rightarrow$ **take into account the mass distribution within each cell**
  - **Generalization to 3D is straightforward**

Homework

1. **Power spectrum and convolution**
   a. Download the data "density.dat", which is a 1024x1024 array. The script "plot__density.py" can be used to load and plot the data.
   b. Apply 2D convolution using a Gaussian filter with σ = 10 cells. Show the resulting image.
   c. Same as (b) but with σ = 100 cells.
   d. Compare the power spectra of the original data, (b), and (c). Discuss the results.

Deadline: May 11 at 11 PM



Cosmological large-scale structure at high-z

Reference

1. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*

2. *Numpy FFT:* https://docs.scipy.org/doc/numpy/reference/routines.fft.html

3. *FFTW:* http://www.fftw.org