FFT in quantum physics

Maciej E. Marchwiany, Maciej Szpindler

CAPACITIES

e-infrastructure

Table of contents

Fourier transform

$$FT : \mathbb{R}^N \rightarrow \mathbb{C}^N$$

The Fourier transform is a mathematical operation with many applications in physics and engineering that expresses a mathematical **function of time** as a **function of frequency**, known as its frequency spectrum; Fourier's theorem guarantees that this can always be done.

Applications

- engineering
- signal processing
- electronics
- cosmology
- NMR spectroscopy
- quantum chemistry
- computational material science
- ...

Fourier transform

$$f \in L^1(\mathbb{R}^N)$$

$$f(\vec{k}) = \int_{\mathbb{R}^N} f(\vec{x}) e^{-2i\pi \vec{x}\vec{k}} d\vec{x}$$

$$f(\vec{k}) = \frac{1}{(2\pi)^N} \int_{\mathbb{R}^N} f(\vec{x}) e^{-2i\pi \vec{x}\vec{k}} d\vec{x}$$

$$f(\vec{k}) = \frac{1}{(2\pi)^{N/2}} \int_{\mathbb{R}^N} f(\vec{x}) e^{-2i\pi \vec{x}\vec{k}} d\vec{x}$$

Inverse Fourier transform

$$f(\vec{k}) = \int_{\mathbb{R}^N} f(\vec{x}) e^{-2i\pi \vec{x}\vec{k}} d\vec{x}$$

$$f(\vec{x}) = \frac{1}{(2\pi)^N} \int_{\mathbb{R}^N} f(\vec{k}) e^{2i\pi \vec{k}\vec{x}} d\vec{k}$$

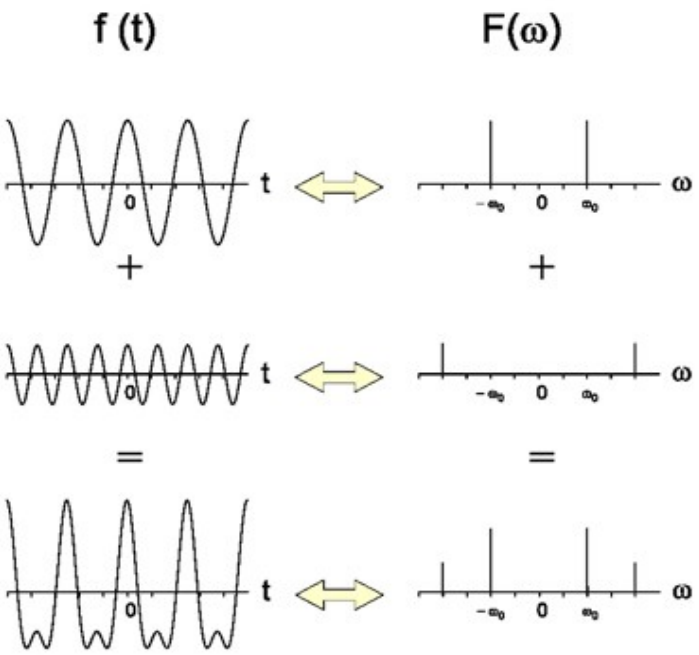
$$f(\vec{k}) = \frac{1}{(2\pi)^N} \int_{\mathbb{R}^N} f(\vec{x}) e^{-2i\pi \vec{x}\vec{k}} d\vec{x}$$

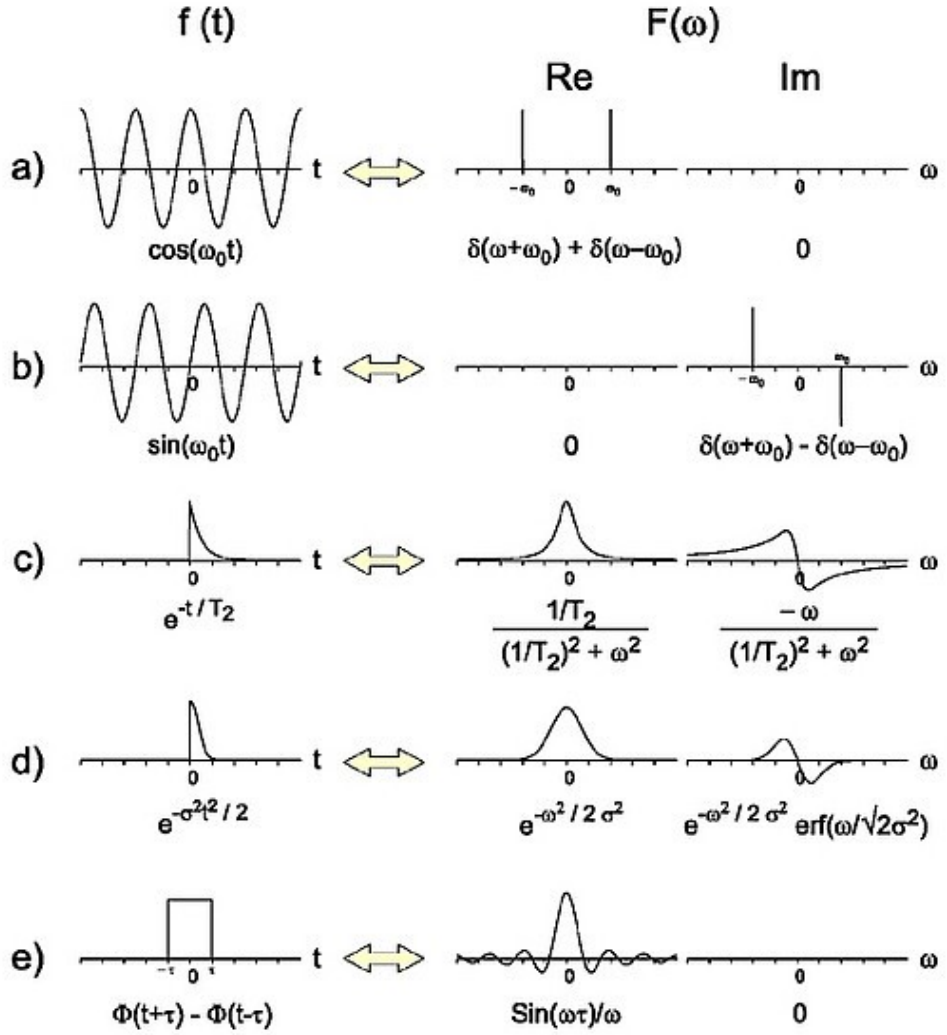
$$f(\vec{k}) = \int_{\mathbb{R}^N} f(\vec{x}) e^{2i\pi \vec{k}\vec{x}} d\vec{k}$$

$$f(\vec{k}) = \frac{1}{(2\pi)^{N/2}} \int_{\mathbb{R}^N} f(\vec{x}) e^{-2i\pi \vec{x}\vec{k}} d\vec{x}$$

$$f(\vec{k}) = \frac{1}{(2\pi)^{N/2}} \int_{\mathbb{R}^N} f(\vec{x}) e^{2i\pi \vec{k}\vec{x}} d\vec{k}$$

Interpretation





Discrete Fourier transform

$$\int dx \rightarrow \sum_n$$

$$x_k = \sum x_n e^{-2\mathrm{i}\,\pi \frac{k}{N} n}$$

$$x_n = \frac{1}{N} \sum x_k e^{2\mathrm{i}\,\pi \frac{k}{N} n}$$

$$k = k_0, k_1, \ldots, k_{N-1}$$

FFT

A fast Fourier transform (FFT) is an efficient algorithm to compute the **discrete Fourier transform** (DFT) and its inverse. There are many distinct FFT algorithms involving a wide range of mathematics, from simple complex-number arithmetic to group theory and number theory.

By far the most common FFT is the Cooley–Tukey algorithm. This is a divide and conquer algorithm that recursively breaks down a DFT of any composite size N = N1N2 into **many smaller DFTs of sizes N1 and N2**, along with O(N) multiplications by complex roots of unity traditionally called twiddle factors (after Gentleman and Sande, 1966).

FFT algorithms

- Cooley–Tukey algorithm
- Prime-factor FFT algorithm
- Bruun's FFT algorithm
- Rader's FFT algorithm
- Bluestein's FFT algorithm
- Odlyzko–Schönhage algorithm

Fourier transform on operators

$$\hat{B}(\vec{x}) \rightarrow \hat{B}(\vec{k})$$

$$\frac{\partial}{\partial x} \rightarrow ik_x$$

$$\nabla^2(x,y,z) \rightarrow -k^2 \qquad k^2 = |\vec{k}|_2$$

Poisson equation

$$V_H(\vec{r}) = \int \frac{\rho(\vec{R})}{|\vec{r}-\vec{R}|} d\vec{R}$$

$V_H(\vec{r})$ Coulomb potential

$\rho(\vec{r})$ Electrons density

$$\nabla^2 V_H(\vec{r}) = -4\pi\rho(\vec{r})$$

$$-k^2 V_H(\vec{k}) = -4\pi\rho(\vec{k})$$

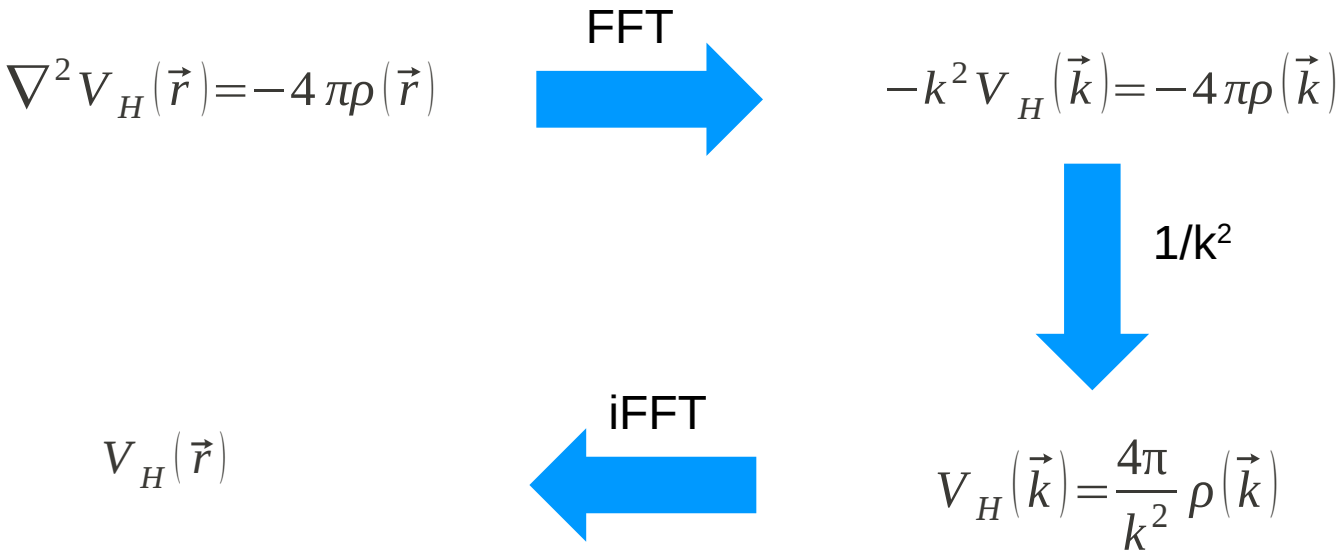
Solving Poisson equation

$$\nabla^2 V_H(\vec{r}) = -4\pi\rho(\vec{r})$$

Poisson equation solver

$$V_H(\vec{r})$$

Solving Poisson equation



k=0

$$k = 0$$

$$f(\vec{k}=0) = \int_{\mathbb{R}^N} f(\vec{x}) e^{-2i\pi \vec{x} 0} d\vec{x} = \int_{\mathbb{R}^N} f(\vec{x}) d\vec{x}$$

$$\rho(\vec{k}=0) = \int \rho(\vec{x}) d\vec{x} = \langle \rho \rangle$$

$\langle x \rangle$ mean value of x

k=0

$\langle \rho \rangle = 0$

$\rho(\vec{k}=0) = 0$

k=0

$\langle \rho \rangle = 0$ $\quad\quad$ $\rho\left(\vec{k}=0\right)=0$

$\langle \rho \rangle \neq 0$ $\quad\quad$ $\rho\left(\vec{k}=0\right)\neq 0$

$\acute{\rho} = \rho + \tilde{\rho} - \tilde{\rho}$ $\quad\quad$ $\tilde{\rho} : \langle \rho + \tilde{\rho} \rangle = 0$ and $V_H(\tilde{\rho})$ - easy to compute

$$V_H(\rho) \to V_H(\rho + \tilde{\rho} - \tilde{\rho}) = V_H(\rho + \tilde{\rho}) - V_H(\tilde{\rho})$$

Electron densinity

$$\rho(\vec{r}) = \sum |\Psi_n(\vec{r})|^2$$

Schrödinger equation

$$\hat{H}\,\Psi = E\Psi$$

$$\left(\nabla^2 + V\right)\Psi = E\Psi$$

Wave function in 1D

$$\left(\nabla^2 + V(r)\right)\Psi(r) = E\Psi(r)$$

$$\left(\frac{\partial^2}{\partial r^2} + V(r)\right)\Psi(r) = E\Psi(r)$$

$$\int d\,r\,|\Psi(\vec{r})|^2 = 1$$

$$V(r) : \Psi(0) = 0 \wedge \Psi(L) = 0$$

$$\Psi(r) = Ae^{ikr} + Be^{-ikr}$$

$$\Psi(r) = a\sin(kr) + b\cos(kr)$$

$$\Psi(r) = a\sin(kr) \qquad k = \frac{n\pi}{L} \qquad n \in \mathbb{Z}$$

Wave function in 3D

$$\left(\nabla^2 + V(r)\right)\Psi(r) = E\Psi(r)$$

$$\int d\,r\,|\Psi(\vec{r})|^2 = 1$$

$$\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} + V(\vec{r})\right)\Psi(\vec{r}) = E\Psi(\vec{r})$$

$$V(\vec{r}) : \Psi(0) = 0 \wedge \Psi(\vec{L}) = 0$$

$$\Psi(r) = \left(A_x e^{ik_x x} + B_x e^{-ik_x x}\right)\left(A_y e^{ik_y y} + B_y e^{-ik_y y}\right)\left(A_z e^{ik_z z} + B_z e^{-ik_z z}\right)$$

$$\Psi(r) = \left(a_x \sin(k_x x) + b_x \cos(k_x r_x)\right)\left(a_y \sin(k_y y) + b_y \cos(k_y r_y)\right)\left(a_z \sin(k_z z) + b_z \cos(k_z r_z)\right)$$

$$\Psi(r) = a \sin(k_x x)\sin(k_y y)\sin(k_z z)$$

$$k_i = \frac{n_i \pi}{L_i} \qquad n_i \in \mathbb{Z} \qquad i = x, y, z$$

Electron densinity

$$\rho(\vec{r}) = \sum |\Psi_n(\vec{r})|^2$$

$$\rho(\vec{r}) = a^2 \sin^2(k_x x) \sin^2(k_y y) \sin^2(k_z z)$$

Time ewolution

$$\Psi\left(\vec{r},t\right)=e^{-i\omega t}\,\Psi\left(\vec{r}\right)$$

Time ewolution in 1D

$$\Psi(r,t) = e^{-i\omega t}\,\Psi(r)$$

$$\Psi(r,t) = Ae^{ikr - i\omega t} + Be^{-ikr - i\omega t}$$

$$\Psi(r) = a\sin(kr - \omega t) + b\cos(kr - \omega t) + c\sin(kr + \omega t) + d\cos(kr + \omega t)$$

$$\Psi(0,0) = 0$$

$$\Psi\left(\frac{1}{k\omega},1\right) = 0$$

$$\Psi(r,t) = a\sin(kr - \omega t)$$

Time ewolution in 3D

$$\rho(\vec{r},t)=\sum|\Psi_n(\vec{r},t)|^2$$

$$\Psi(\vec{r},t)=a\sin(k_x x-\omega t)\sin(k_y y)\sin(k_z z)$$

$$\rho(\vec{r},t)=|a|^2\sin^2(k_x x-\omega t)\sin^2(k_y y)\sin^2(k_z z)$$

DFT library

- **FFTW**
- **P3DFFT**
- GSL
- ACML
- **ESSL**
- FFTPACK
- FFTEASY
- JTransforms
- GPUFFTW
- ...

Deklaracja biblioteki

```
#include "p3dfft.h"
#include <mpi.h>
```

P3DFFT: initialization

```
p3dfft_setup(proc_dims,nx,ny,nz,overwrite,memsize)
```

`proc_dims` - An array of two integers, specifying how the processor grid should be decomposed. Either 1D or 2D decomposition can be specified.

`nx,ny,nz` - Dimensions of the 3D transform

`overwrite` - When set to .true. (or 1 in C) this argument indicates that itis safe to overwrite the input of the btran (backward transform)routine.

`memsize` – An array of tree integers, specifying memoru used

P3DFFT: Array Decomposition

```
p3dfft_get_dims(start,end,size,ip)
```

`start` - An array containing 3 integers, defining the beginning indices of the local array for the given task within the global grid

`end` - An array containing 3 integers, defining the ending indices of the local array within the global grid

`size` - An array containing 3 integers, defining the local array's dimensions

`ip` - An integer argument specifying one of the two choices for array types (1 for forward, 2 for backward, 3 for in-place transform)

P3DFFT: Array Decomposition

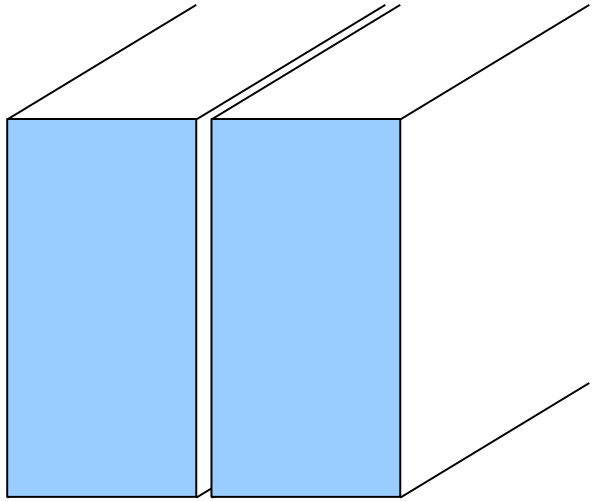
P3DFFT uses 2D block decomposition to assign local arrays for each task. Let P_1 and P_2 be processor grid dimensions defined in the call to `p3dfft_setup`.

For `ip=1` Array is distributed among P_1 tasks in Y dimension and P_2 tasks in Z dimension

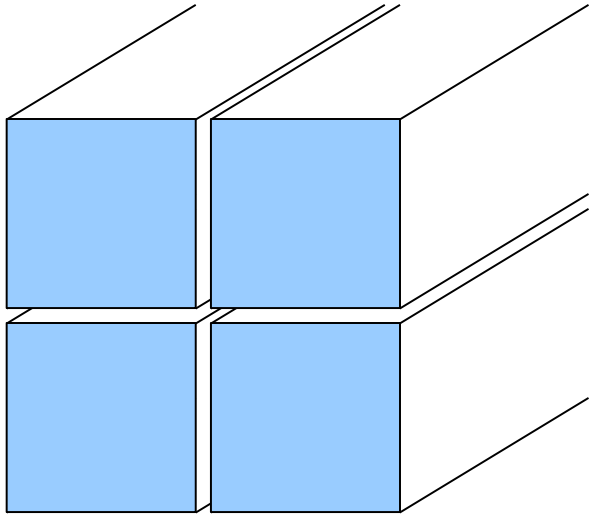
For `ip=2` Array is distributed among P_1 tasks in X dimension and P_2 tasks in Y dimension

Array Decomposition

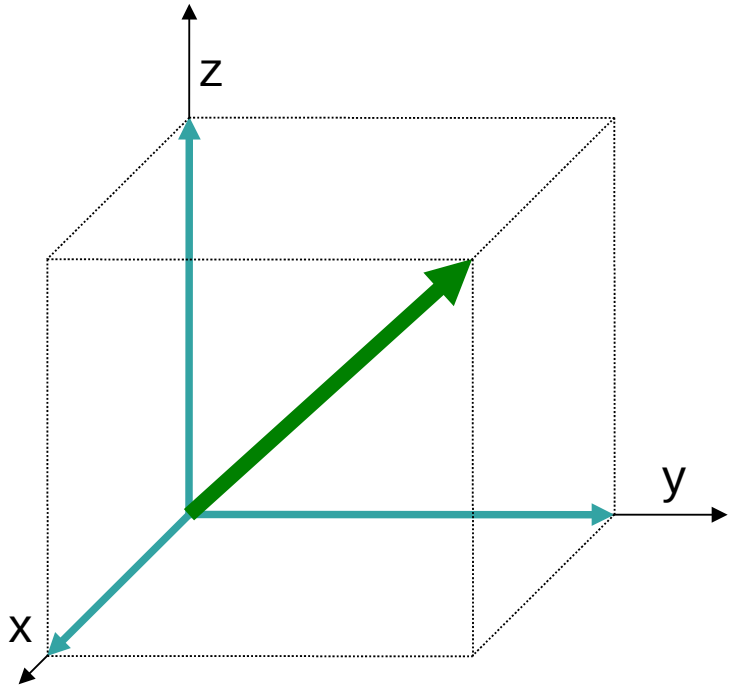
1D (FFTW)

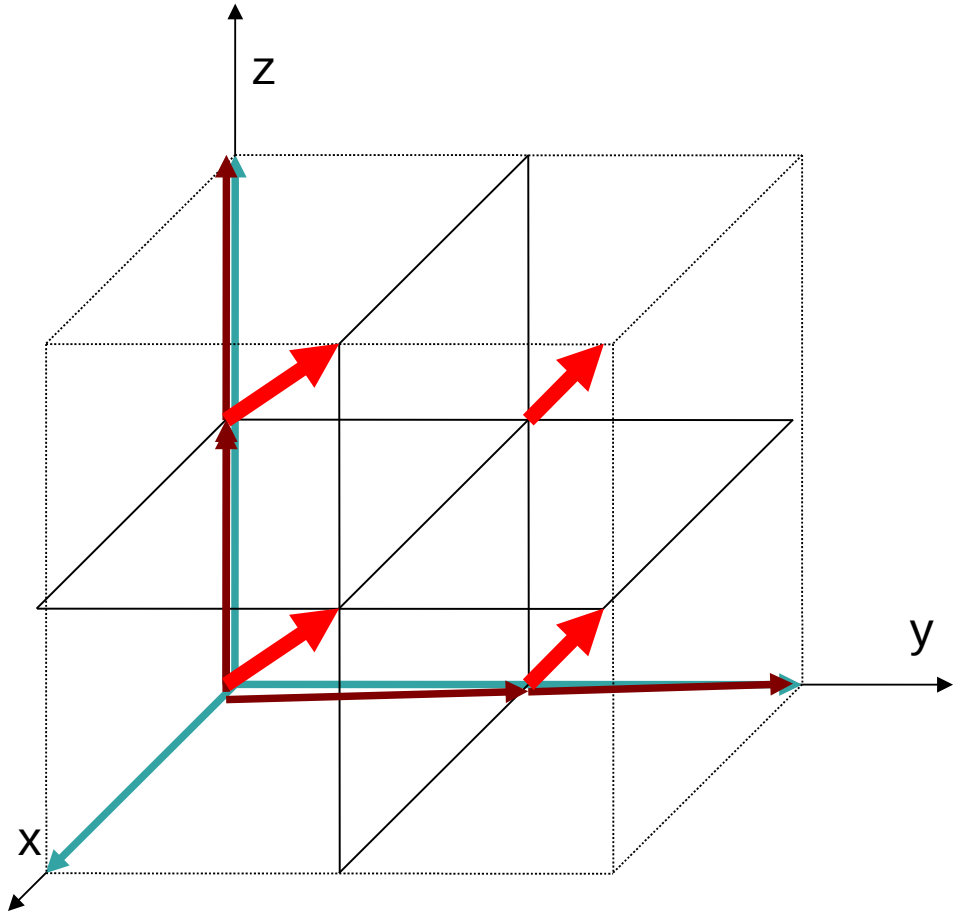


2D (P3DFFT)



Array Decomposition





Array Decomposition

$$\vec{X} = \hat{X} = X \begin{pmatrix} 0 & \dots & N_x - 1 \\ 0 & \dots & N_y - 1 \\ 0 & \dots & N_z - 1 \end{pmatrix}$$

$$\hat{X} = \sum X_p \begin{pmatrix} start[0] & \dots & end[0] \\ start[1] & \dots & end[1] \\ start[2] & \dots & end[2] \end{pmatrix}$$

$$\hat{X} = \sum X_p \begin{pmatrix} 0 & \dots & \text{size}[0] \\ 0 & \dots & \text{size}[1] \\ 0 & \dots & \text{size}[2] \end{pmatrix}$$

P3DFFT: Forward transform

```
p3dfft_ftran_r2c(IN,OUT,op)
```

`IN` - an array of real numbers with dimensions defined by array type with ip=1

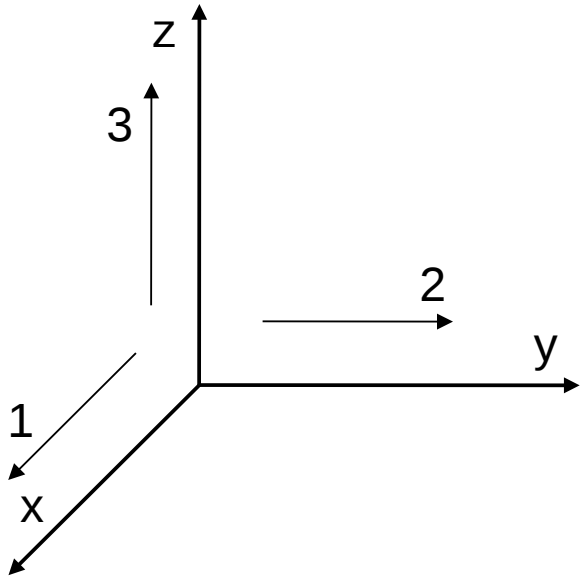
`OUT` - an array of complex numbers with dimensions defined by array type with ip=2

`op` - 3-letter character string indicating the type of transform desired in X, Y, Z direction

- `t` Fourier transform
- `c` Cosine transform
- `s` Sine transform
- `n` or `0` Empty transform (no operation, output is identical to input)

P3DFFT: Forward transform

`IN` - an array of double numbers in order:



P3DFFT: Forward transform

`IN` - an array of double numbers in order:

$$\vec{I}\,N = \begin{matrix} \mathrm{IN}(0,0,0) \\ \mathrm{IN}(1,0,0) \\ \cdots \\ \mathrm{IN}(N_x-1,0,0) \\ \mathrm{IN}(0,1,0) \\ \mathrm{IN}(1,1,0) \\ \cdots \\ \mathrm{IN}(N_x-1,N_y-1,0) \\ \mathrm{IN}(0,0,1) \\ \cdots \\ \mathrm{IN}(N_x-1,N_y-1,N_z-1) \end{matrix}$$

P3DFFT: Forward transform

OUT - an array of complex numbers in order:

$$\vec{OUT} = \begin{bmatrix} \Re(OUT_1) \\ \Im(OUT_1) \\ \Re(OUT_2) \\ \Im(OUT_2) \\ \cdots \\ \Re(OUT_{N-1}) \\ \Im(OUT_{N-1}) \end{bmatrix}$$

What is compute

$$x_k = \sum x_n e^{-2i\pi \frac{k}{N} n}$$

$$x_n = \frac{1}{2\pi} \sum x_k e^{2i\pi \frac{k}{N} n}$$

What is compute

$$x_k = \sum x_n e^{-2i\pi \frac{k}{N} n}$$

$$x_n = \frac{1}{2\pi} \sum x_k e^{2i\pi \frac{k}{N} n}$$

$$x_n \rightarrow^{FFT} x_k \rightarrow^{iFFT} N x_n$$

P3DFFT: Backward transform

```
p3dfft_ftran_c2r(IN,OUT,op)
```

`IN` - an array of real numbers with dimensions defined by array type with ip=2

`OUT` - an array of complex numbers with dimensions defined by array type with ip=1

`op` - 3-letter character string indicating the type of transform desired in X, Y, Z direction

- `t` Fourier transform
- `c` Cosine transform
- `s` Sine transform
- `n` or `0` Empty transform (no operation, output is identical to input)

Terminates P3DFFT

`p3dfft_clean()` - Terminates P3DFFT