UNIVERSITY OF TEXAS ARLINGTON

PHYS 5319-001: Math Methods in Physics III

Fast Fourier Transformation (FFT)

Instructor:	**Dr. Qiming Zhang**
Office:	**CPB 336**
Phone:	**817-272-2020**
Email:	**zhang@uta.edu**

DFT recap

$$Y_n \equiv \sum_{k=1}^{N} y_k \, e^{-2\pi ikn/N}$$

$$y_k \equiv \frac{1}{N} \sum_{n=1}^{N} Y_n \, e^{2\pi ikn/N}$$

Periodic: $Y_{n+N} = Y_n$ $\quad y_{k+N} = y_k$

Let $Z \equiv e^{2\pi i/N}$ $\quad e^{2\pi ikn/N} = Z^{kn}$ $\quad$ Z could be pre-calculated

Cost of DFT: for each $Y_n \equiv \sum_{k=1}^{N} y_k * Z^{kn}$, about *O(N)*

But *n=1,2,…,N*. So we have *N* Y_n's to calculate, the total operation ~ $O(N^2)$

Fast Fourier Transformation

- Idea by Gauss (1886); developed by Cooley & Tukey (1965)
- "the most important numerical algorithm of our lifetime“

For N=2^m^,

$$y_n = \sum_{k=0}^{N-1} Y_k\, e^{2\pi i k n/N}$$

$$= \sum_{k=0}^{\frac{N}{2}-1} Y_{2k} e^{i2\pi 2kn/N} + \sum_{k=0}^{\frac{N}{2}-1} Y_{2k+1} e^{i2\pi(2k+1)n/N}$$

Fast Fourier Transformation

$$y_n = \sum_{k=0}^{N-1} Y_k\, e^{2\pi ikn/N}$$

Split the FT into odd/even terms

$n = 0, 1, 2, \ldots$

$$= \sum_{k=0}^{\frac{N}{2}-1} Y_{2k} e^{i2\pi 2kn/N} + \sum_{k=0}^{\frac{N}{2}-1} Y_{2k+1} e^{i2\pi(2k+1)n/N}$$

$$= y_n^e + y_n^o e^{i2\pi n/N}$$

where $y_n^e = \sum_{k=0}^{\frac{N}{2}-1} Y_{2k} e^{i2\pi kn/\left(\frac{N}{2}\right)}$, $y_n^o = \sum_{k=0}^{\frac{N}{2}-1} Y_{2k+1} e^{i2\pi kn/\left(\frac{N}{2}\right)}$

y_n^e and y_n^o are even/odd half, respectively

Let $w = e^{i2\pi/N}$, then $\underline{y_n = y_n^e + w^n y_n^o}$

y_n^e and y_n^o has a period of *N/2*

This is very significant!

So, $y_n = y_n^e + w^n y_n^o$ y_n^e and y_n^o has a period of *N/2*

Or, $y_n \rightarrow y_n^e$ & y_n^o

Then do the same to $y_n^e \rightarrow y_n^{ee}$ & y_n^{eo} and $y_n^o \rightarrow y_n^{oe}$ & y_n^{oo}

Can be done recursively: $N \rightarrow \frac{N}{2} \rightarrow \frac{N}{4} \rightarrow \cdots \rightarrow 2$

Example for *N=8:*

$N = 1024 = 2^{10}$

DFT: $O(1M)$

FFT: $O(10k)$

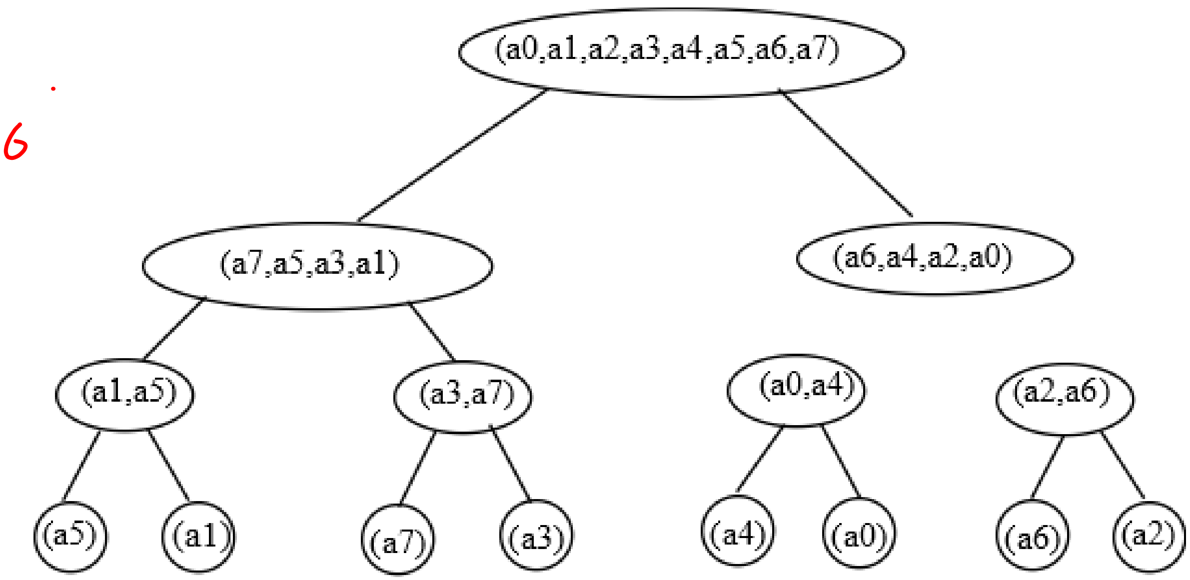


Figure 1 : FFT algorithm with the bit-reversal permutation at the end

Total cost *O(Nlog₂N)*

$$y_n = \sum_{k=0}^{N-1} Y_k \, e^{2\pi i k n/N}$$

… …

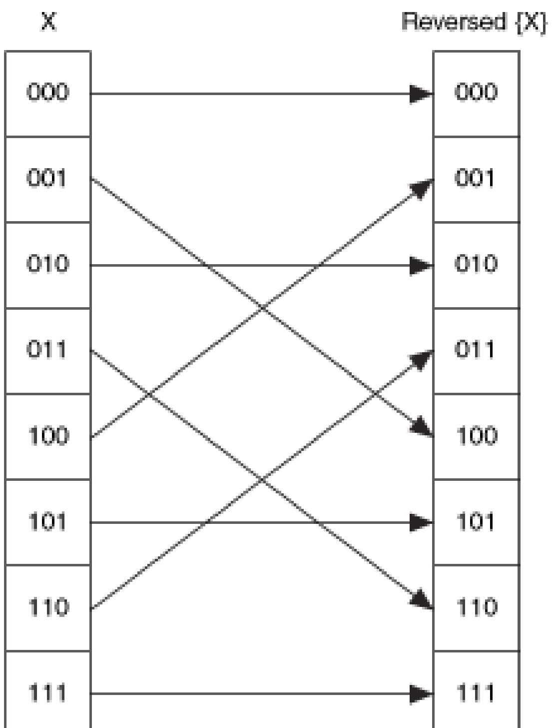
Eventually, $y_n^{eoee\dots oe} = Y_k$ for some k, at the lowest level

Question: which pattern of e's and o's $\Rightarrow k$

Bit reversal

Let $e = 0, \& \; o = 1$, express k in binary: e.g. $6 \rightarrow 110$

Store in the bit reversal order:



We combine adjacent pairs to get two-point transformation (next level). Then combine adjacent pairs of pairs to get 4-ponts transformation. ……

$$y_n = y_n^e + w^n y_n^o \quad (w = e^{i2\pi/N})$$

Now, as a user, we don't see bit reversal storage. The input and output array for FFT call is in the natural sequence.

The restriction for $N=2^m$ is relaxed to $N=2^l 3^m 5^q 7^p$ where l,m,q,p are integers, or 0 (except l).

ψ — real

$\psi^*(x) = \sum \phi^*(k) e^{-ikx}$

$\phi^*(k) = \phi(-k)$

Instead of N=8, 16, 32, 64, 128, 256,…
You may have N=18, 20, 24, …

A computer platform usually will provide 1-d FFT for

real$\rightarrow$*complex* then *complex* $\rightarrow$ *real*

Or complex $\rightarrow$ complex

e.g. ccfft(f, n, id)

f(1:n)—complex array with n components

n -- size N, usually multiple of 2, 3, 5, and 7

Id-- 0 initialize, +1 forward FFT, -1 inverse FFT

FFTW package

- https://www.fftw.org/

Download GitHub Mailing List Benchmark Features Documentation FAQ Links Feedback

Introduction

FFTW is a C subroutine library for computing the discrete Fourier transform (DFT) in one or more dimensions, of arbitrary input size, and of both real and complex data (as well as of even/odd data, i.e. the discrete cosine/sine transforms or DCT/DST). We believe that FFTW, which is free software, should become the FFT library of choice for most applications.

The latest official release of FFTW is version **3.3.9**, available from our download page. Version 3.3 introduced support for the AVX x86 extensions, a distributed-memory implementation on top of MPI, and a Fortran 2003 API. Version 3.3.1 introduced support for the ARM Neon extensions. See the release notes for more information.

The FFTW package was developed at MIT by Matteo Frigo and Steven G. Johnson.

Our benchmarks, performed on on a variety of platforms, show that FFTW's performance is typically superior to that of other publicly available FFT software, and is even competitive with vendor-tuned codes. In contrast to vendor-tuned codes, however, FFTW's performance is *portable*: the same program will perform well on most architectures without modification. Hence the name, "FFTW," which stands for the somewhat whimsical title of "**Fastest Fourier Transform in the West**."

Subscribe to the fftw-announce mailing list to receive release announcements (or use the web feed).

On MATLAB

Syntax

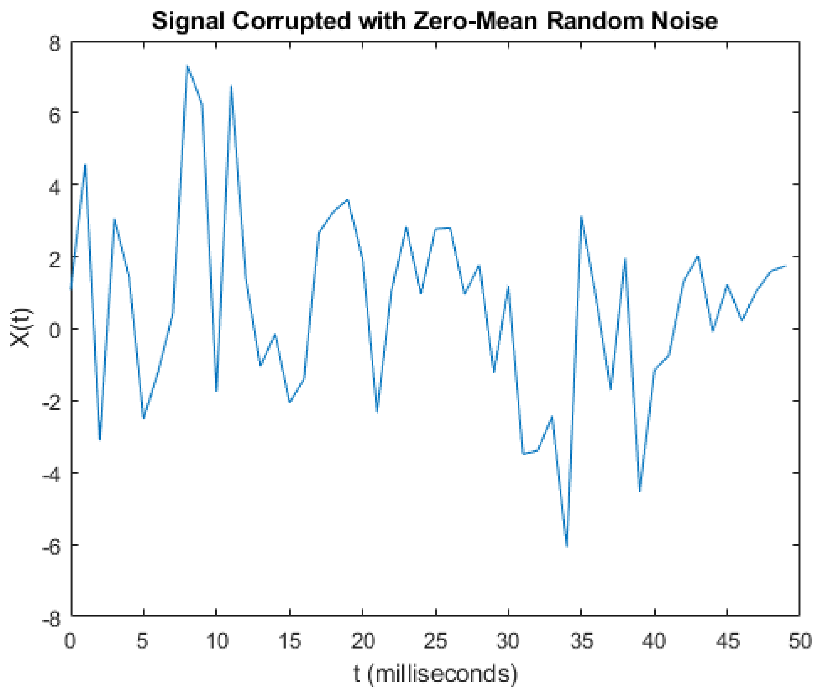
```
Y = fft(X)
Y = fft(X,n)
Y = fft(X,n,dim)
```

Description

`Y = fft(X)` computes the discrete Fourier transform (DFT) of X using a fast Fourier transform (FFT) algorithm.

Example:

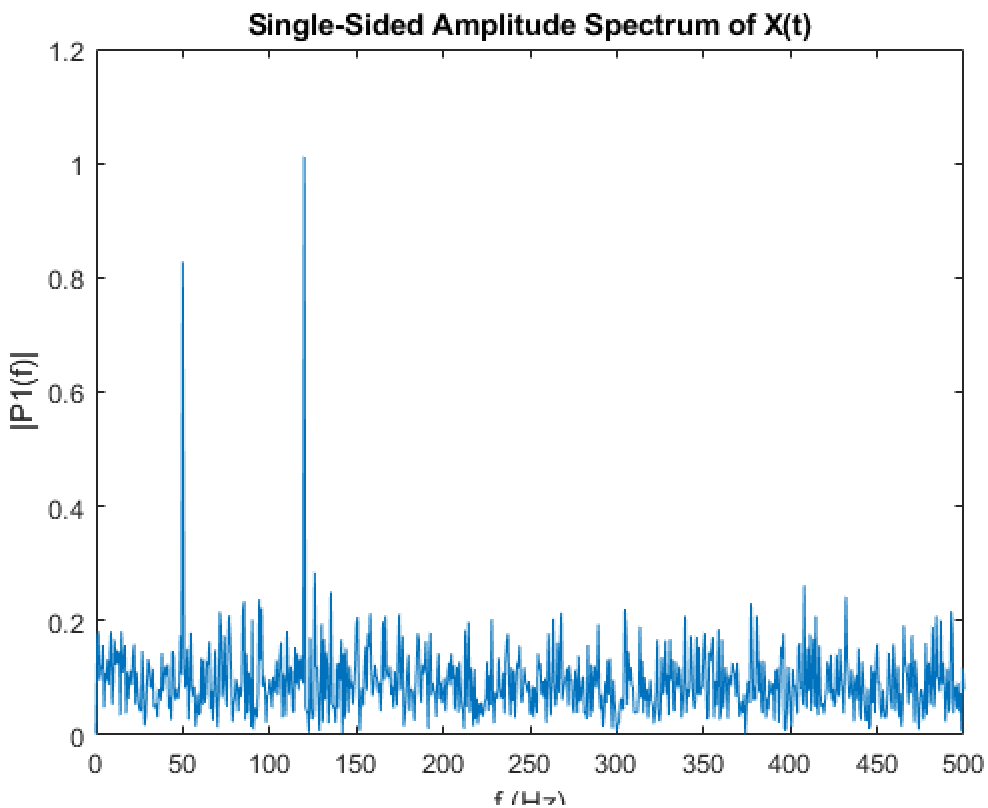
```
Fs = 1000;            % Sampling frequency
T = 1/Fs;             % Sampling period
L = 1500;             % Length of signal
t = (0:L-1)*T;        % Time vector
```



MATLAB Example (con't)

```
Y = fft(X);

P2 = abs(Y/L);
P1 = P2(1:L/2+1);
P1(2:end-1) = 2*P1(2:end-1);
```



On Python

https://docs.scipy.org/doc/scipy/reference/tutorial/

SciPy User Guide

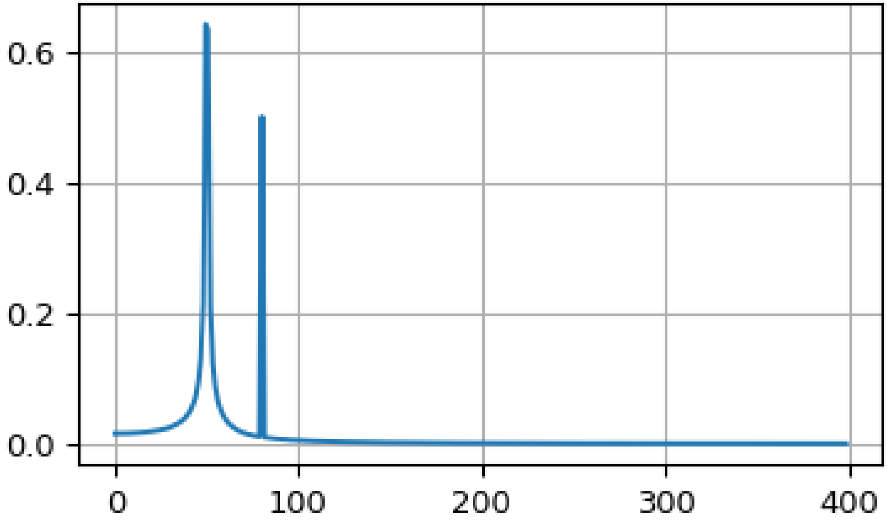
- Introduction
- Special functions (`scipy.special`)
- Integration (`scipy.integrate`)
- Optimization (`scipy.optimize`)
- Interpolation (`scipy.interpolate`)
- Fourier Transforms (`scipy.fft`)
- Signal Processing (`scipy.signal`)
- Linear Algebra (`scipy.linalg`)
- Sparse eigenvalue problems with ARPACK
- Compressed Sparse Graph Routines (`scipy.sparse.csgraph`)
- Spatial data structures and algorithms (`scipy.spatial`)
- Statistics (`scipy.stats`)
- Multidimensional image processing (`scipy.ndimage`)
- File IO (`scipy.io`)

Fourier Transforms (`scipy.fft`)

Contents

```
>>> from scipy.fft import fft, ifft
>>> x = np.array([1.0, 2.0, 1.0, -1.0, 1.5])
>>> y = fft(x)
>>> y
array([ 4.5       +0.j        ,  2.08155948-1.65109876j,
       -1.83155948+1.60822041j, -1.83155948-1.60822041j,
        2.08155948+1.65109876j])
>>> yinv = ifft(y)
>>> yinv
array([ 1.0+0.j,  2.0+0.j,  1.0+0.j, -1.0+0.j,  1.5+0.j])
```

```
>>> from scipy.fft import fft, fftfreq
>>> # Number of sample points
>>> N = 600
>>> # sample spacing
>>> T = 1.0 / 800.0
>>> x = np.linspace(0.0, N*T, N, endpoint=False)
>>> y = np.sin(50.0 * 2.0*np.pi*x) + 0.5*np.sin(80.0 * 2.0*np.pi*x)
>>> yf = fft(y)
>>> xf = fftfreq(N, T)[:N//2]
>>> import matplotlib.pyplot as plt
>>> plt.plot(xf, 2.0/N * np.abs(yf[0:N//2]))
>>> plt.grid()
>>> plt.show()
```



in 1-d real space, we have a periodic function $f(x+L) = f(x)$

Discrete Fourier Transform (DFT)

$$f_i, i = 1,2, \dots N$$

FT

$$g(k) = \frac{1}{L}\int_0^L f(x)e^{ikx}dx$$

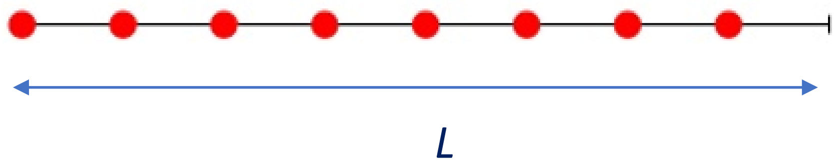
$$= \frac{1}{N}\sum_{j=1}^{N} f_j e^{ikx_i}$$

FT^{-1}

$$f(x) = \sum_{l=1}^{N} g_l e^{-ik_l x}$$

N discrete points:

$$x = 0, \frac{L}{N}, \frac{2L}{N}, \dots, \frac{L}{N}(N-1)$$



Since *f(x+L)=f(x)* $\Rightarrow e^{ikL}$=1,

$$kL = 2\pi n, n = 0,1,2, \dots, N-1$$

We have N discrete k-points:

$$k = \frac{2\pi n}{L}, n = 0, 1, 2, \dots, N-1$$

Or $n = -\frac{N}{2}, -\frac{N}{2}+1, \dots, 0, 1, \dots \frac{N}{2} - 1$

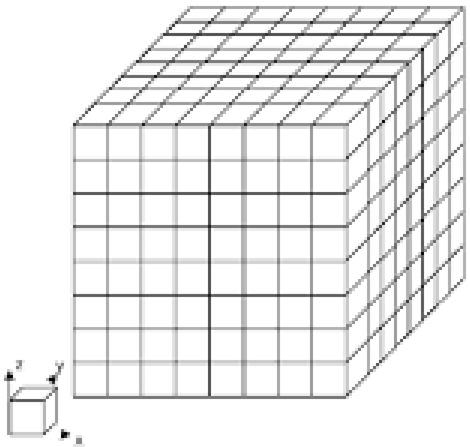
3-dimension space?

3-dimension FFT in space

- $f(\vec{r}) = f(x, y, z)$ in a supercell $L_1 \times L_2 \times L_3$ (N_1, N_2, N_3)
- FFT: $f(x, y, z) \rightarrow g(k_x, k_y, k_z)$

$\nabla^2 V = f(\vec{r})$

- If the program only provide a 1-d fft library, you have to write a "driver" to do 3-d FFT



Loop over z $(1 \rightarrow N_3)$

loop over y $(1 \rightarrow N_2)$

FFT $f(x, y, z) \rightarrow \check{f}(k_x, y, z)$

end loop

end loop

Then do loops: *FFT* $\check{f}(k_x, y, z) \rightarrow F(k_x, k_y, z)$

And: *FFT* $F(k_x, k_y, z) \rightarrow g(k_x, k_y, k_z)$

- Normalization: $\frac{1}{N_1 N_2 N_3}$ (when do FFT^{-1})