Lecture 11 Fast Fourier Transform (FFT)

Weinan E\textsuperscript{1,2} and Tiejun Li\textsuperscript{2}

\textsuperscript{1}Department of Mathematics, 
Princeton University, 
weinan@princeton.edu

\textsuperscript{2}School of Mathematical Sciences, 
Peking University, 
tieli@pku.edu.cn 
No.1 Science Building, 1575
Outline

Examples

Fast Fourier Transform

Applications
Signal processing

- Filtering: a polluted signal

- High pass and low pass filter (signal and noise)

- How to obtain the high frequency and low frequency quickly?
Solving PDEs on rectangular mesh

- Solving the Poisson equations

\[-\Delta u = f \text{ in } \Omega\]

\[u = 0 \text{ on } \partial\Omega\]

in the rectangular domain

- After discretization we will obtain the linear system with about \(N^2\) unknowns

\[-\frac{u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j}}{4h^2} = f_{ij}\]

- The FFT would give a fast algorithm to solve the system above with computational efforts \(O(N^2 \log_2 N)\).
Computing convolution (卷积)

- Suppose

\[ h(x) = \int_{0}^{2\pi} f(x - y)g(y)\,dy \]

is the convolution of \( f \) and \( g \), where \( f(x), g(x) \in C_{2\pi} \) are period \( 2\pi \) functions.

- Take \( x_j = j\delta, j = 0, 1, \ldots, N - 1, \delta = \frac{2\pi}{N} \) and apply simple rectangular discretization

\[ h(x_i) \approx \sum_{j=0}^{N-1} f(x_i - x_j)g(x_j) \cdot \delta \quad i = 0, 1, \ldots, N - 1 \]

- Define \( f_i = f(x_i), g_i = g(x_i) \), and let \( f_i \) is period \( N \) respect to the subscript \( i \), define

\[ h_i = \sum_{j=0}^{N-1} f_{i-j}g_j \cdot \delta \quad i = 0, 1, \ldots, N - 1 \]

- The direct computation is \( O(N^2) \).
Fast Fourier Transform is one of the top 10 algorithms in 20th century.

But its idea is quite simple, even for a high school student!
Examples

Fast Fourier Transform

Applications
Suppose $f(x)$ is absolutely integrable in $(-\infty, +\infty)$, then the Fourier transform of $f(x)$ is

$$\hat{f}(k) = \int_{-\infty}^{+\infty} f(x)e^{-ikx} dx.$$ 

Moreover if $f(x)$ is square integrable, then the inverse Fourier transform of $\hat{f}(k)$ is

$$f(x) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} \hat{f}(k)e^{ikx} dk.$$
Properties of Fourier transform

Some important properties of Fourier transform:

1. Derivative to coefficient:

\[
(f'(x))(k) = ik \hat{f}(k);
\]

2. Translation property:

\[
(f(x - a))(k) = e^{-ika} \hat{f}(k);
\]

3. Convolution to multiplication:

\[
(f * g)(k) = \hat{f}(k) \hat{g}(k);
\]

where \((f * g)(x) = \int_{-\infty}^{+\infty} f(x - y)g(y)dy\).

4. Parseval’s identity:

\[
\int_{-\infty}^{+\infty} |f(x)|^2 dx = \frac{1}{2\pi} \int_{-\infty}^{+\infty} |\hat{f}(k)|^2 dk.
\]
Discrete Fourier transform (DFT)

- Suppose we have \( \mathbf{a} = (a_0, a_1, \cdots, a_{N-1})^T \), define DFT of \( \mathbf{a} \) as
  \[
  \mathbf{c} = (c_0, c_1, \cdots, c_{N-1})^T \triangleq \hat{\mathbf{a}},
  \]
  where
  \[
  c_k = \sum_{j=0}^{N-1} a_j e^{-jk \frac{2 \pi i}{N}}, \quad k = 0, 1, \ldots, N - 1.
  \]

  Here \( i \) is the imaginary unit, \( e^{-\frac{2 \pi i}{N}} \triangleq \omega \) is the \( N \)-th root of unity.

- \( \mathbf{a} \) is the inverse discrete Fourier transform of \( \mathbf{c} \) defined as
  \[
  a_j = \frac{1}{N} \sum_{k=0}^{N-1} c_k e^{jk \frac{2 \pi i}{N}}, \quad j = 0, 1, \ldots, N - 1.
  \]

- DFT is closely related to the trigonometric interpolation for \( 2\pi \)-periodic function
  \[
  T(x) = \sum_{k=-\frac{N}{2}+1}^{\frac{N}{2}} b_k e^{ikx}.
  \]

  such that at \( x_j = \frac{2j\pi}{N} \), \( T(x_j) = a_j \), \( j = 0, 1, \ldots, N - 1 \). The readers may find the relation between \( c_k \) and \( b_k \).
Remark on DFT

- DFT can be considered as a linear transformation.
- Define Fourier matrix

\[
F = \begin{pmatrix}
1 & 1 & \ldots & 1 \\
1 & \omega & \ldots & \omega^{N-1} \\
\vdots & \vdots & \ddots & \vdots \\
1 & \omega^{N-1} & \ldots & \omega^{(N-1)^2}
\end{pmatrix} = (\omega^{jk})_{j,k=0}^{N-1}
\]

where \( \omega \) is the \( N \)-th root of unity.

- \( c \) is the Fourier transform of \( a \) can be represented as

\[
c = F a
\]
Remark on DFT

- Inverse DFT can also be considered as a linear transformation.
- Define inverse Fourier matrix

\[
F^{-1} = G = \frac{1}{N} \begin{pmatrix}
1 & 1 & \cdots & 1 \\
1 & \omega^{-1} & \cdots & \omega^{-(N-1)} \\
\vdots & \vdots & \ddots & \vdots \\
1 & \omega^{-(N-1)} & \cdots & \omega^{-(N-1)^2}
\end{pmatrix} = (\omega^{-jk})_{j,k=0}^{N-1}
\]

where \( \omega \) is the \( N \)-th root of unity.

- \( a \) is the inverse Fourier transform of \( c \) can be represented as

\[
a = Gc
\]
Properties of DFT

- Convolution to multiplication:

\[
(f \ast g)_k = \hat{f}_k \hat{g}_k \quad k = 0, 1, \ldots, N - 1
\]

where

\[
(f \ast g)_l = \sum_{j=0}^{N-1} f_{l-j}g_j \quad l = 0, 1, \ldots, N - 1,
\]

and \(f_l\) is period \(N\) with respect to index \(l\), i.e.

\[
f_{-1} = f_{N-1}, f_{-2} = f_{N-2}, \ldots
\]

- Parseval’s identity:

\[
N \sum_{j=0}^{N-1} |a_j|^2 = \sum_{k=0}^{N-1} |c_k|^2
\]
FFT idea

- FFT is proposed by J.W. Cooley and J.W. Tukey in 1960s, but the idea may be traced back to Gauss.
- The basic motivation is if we compute DFT directly, i.e.
  \[
  c = Fa
  \]
  we need \(N^2\) multiplications and \(N(N-1)\) additions. Is it possible to reduce the computation effort?
- First consider the case \(N = 4\)

\[
F = \begin{pmatrix}
1 & 1 & 1 & 1 \\
1 & -i & -1 & i \\
1 & -1 & 1 & -1 \\
1 & i & -1 & -i \\
\end{pmatrix}, \quad Fa = \begin{pmatrix}
(a_0 + a_2) + (a_1 + a_3) \\
(a_0 - a_2) - i(a_1 - a_3) \\
(a_0 + a_2) - (a_1 + a_3) \\
(a_0 - a_2) + i(a_1 - a_3) \\
\end{pmatrix}
\]
FFT idea

- From the concrete form of DFT, we actually need 2 multiplications (timing \( \pm i \)) and 8 additions \((a_0 + a_2, a_1 + a_3, a_0 - a_2, a_1 - a_3 \) and the additions in the middle).
- This observation may reduce the computational effort from \( O(N^2) \) into \( O(N \log_2 N) \)

- Because

\[
\lim_{N \to \infty} \frac{\log_2 N}{N} = 0
\]

It is a typical fast algorithm.

- Fast algorithms of this type of recursive halving are very typical in scientific computing.
Construction of FFT

- Consider $N = 2^m$ and denote

$$p(x) = a_0 + a_1 x + \cdots + a_{N-1} x^{N-1},$$

divide $p(x)$ into odd (奇) and even (偶) power parts

$$p(x) = (a_0 + a_2 x^2 + \cdots) + x(a_1 + a_3 x^2 + \cdots)$$

$$= p_e(x^2) + x p_o(x^2)$$

where

$$p_e(t) = a_0 + a_2 t + \cdots + a_{N-2} t^{\frac{N}{2} - 1},
\quad p_o(t) = a_1 + a_3 t + \cdots + a_{N-1} t^{\frac{N}{2} - 1}$$

- Define $\omega_k = e^{-\frac{2\pi i}{k}}$ (k-th root of unity), then when $j = 0, 1, \ldots, \frac{N}{2} - 1$

$$\begin{cases} 
    c_j &= p_e(\omega_N^{2j}) + \omega_N^{j} p_o(\omega_N^{2j}) \\
    c_{\frac{N}{2} + j} &= p_e(\omega_N^{2(\frac{N}{2} + j)}) + \omega_N^{\frac{N}{2} + j} p_o(\omega_N^{2(\frac{N}{2} + j)})
\end{cases}$$
Construction of FFT

▶ Pay attention that

\[ \omega_{2j}^N = \omega_{N/2}^j, \quad \omega_{N/2}^j + j = -\omega_{N}^j, \quad \omega_{N}^{N+2j} = \omega_{N/2}^j \]

then

\[ c_j = v_j + \omega_{N}^j u_j, \quad c_{j+N/2} = v_j - \omega_{N}^j u_j \quad \text{for } j = 0, 1, ..., \frac{N}{2} - 1 \]

where

\[ v_j = p_e(\omega_{N/2}^j), \quad u_j = p_o(\omega_{N/2}^j) \]

▶ The formula above show that the DFT of \( N \) components vector \( a \) could be converted to compute the DFT of two \( \frac{N}{2} \) components vectors \( a_e, a_o \) and some simple additions and multiplications. This is called Danielson-Lanczos algorithm. The recursive application of this idea gives FFT.
A simple example: \( N = 8 \)

Suppose the array

\[
a = (a_0, a_1, \cdots, a_7)^T
\]

Step A: Splitting (reordering) (odd parts and even parts):

- Step 1

\[
a_e = (a_0, a_2, a_4, a_6)^T, \quad a_o = (a_1, a_3, a_5, a_7)^T;
\]

- Step 2

\[
a_{ee} = (a_0, a_4)^T, \quad a_{eo} = (a_2, a_6)^T, \\
a_{oe} = (a_1, a_5)^T, \quad a_{oo} = (a_3, a_7)^T;
\]

- Step 3

\[
\begin{array}{cccccccc}
a_{eee} & a_{eoo} & a_{oeo} & a_{eoo} & a_{oee} & a_{ooe} & a_{ooe} & a_{ooo} \\
a_0 & a_4 & a_2 & a_6 & a_1 & a_5 & a_3 & a_7
\end{array}
\]
**A simple example: \( N = 8 \)**

Step B: Combination:

- **Step 1**

\[
\begin{align*}
\mathbf{c}_{ee} &= (a_0 + \omega_2^0 a_4, a_0 - \omega_2^0 a_4)^T, \\
\mathbf{c}_{eo} &= (a_2 + \omega_2^0 a_6, a_2 - \omega_2^0 a_6)^T, \\
\mathbf{c}_{oe} &= (a_1 + \omega_2^0 a_5, a_1 - \omega_2^0 a_5)^T, \\
\mathbf{c}_{oo} &= (a_3 + \omega_2^0 a_7, a_3 - \omega_2^0 a_7)^T,
\end{align*}
\]

- Define the notations

\[
\begin{align*}
\mathbf{w}_4 &= (w_4^0, w_4^1)^T, \\
\mathbf{w}_8 &= (w_8^0, w_8^1, w_8^2, w_8^3)^T,
\end{align*}
\]

and

\[
X \circ Y \triangleq (x_j y_j)_j
\]

as the vector product through multiplication by components.
A simple example: \( N = 8 \)

Step B: Combination:

- **Step 2**

\[
\begin{align*}
\mathbf{c}_e &= \begin{bmatrix} c_{ee} + w_4 \circ c_{eo} \\ c_{ee} - w_4 \circ c_{eo} \end{bmatrix}, \\
\mathbf{c}_o &= \begin{bmatrix} c_{oe} + w_4 \circ c_{oo} \\ c_{oe} - w_4 \circ c_{oo} \end{bmatrix},
\end{align*}
\]

- **Step 3**

\[
\mathbf{c} = \begin{bmatrix} c_e + w_8 \circ c_0 \\ c_e - w_8 \circ c_0 \end{bmatrix}
\]
A simple sketch of FFT ($N = 8$)
A remark on the reordering

If we map \( e \) to 0, and \( o \) to 1, we can find the binary representation of the indices after reordering is just the bit reversal before reordering.

\[
\begin{align*}
0 &= 000_2 & 000_2 &= 0 \\
1 &= 001_2 & 100_2 &= 4 \\
2 &= 010_2 & 010_2 &= 2 \\
3 &= 011_2 & 110_2 &= 6 \\
4 &= 100_2 & 001_2 &= 1 \\
5 &= 101_2 & 101_2 &= 5 \\
6 &= 110_2 & 011_2 &= 3 \\
7 &= 111_2 & 111_2 &= 7
\end{align*}
\]
Outline

Examples

Fast Fourier Transform

Applications
Compute the convolution

- From the discretization at the beginning, we have

\[ h_i = \sum_{j=0}^{N-1} f_{i-j} g_j \cdot \delta \quad i = 0, 1, \ldots, N - 1 \]

thus

\[ h = (\hat{h})^\vee = (\delta \cdot \hat{f} \circ \hat{g})^\vee \]

- After using FFT, \( N^2 + N \) multiplications and \( N(N - 1) \) additions are reduced to \( \frac{3}{2}N \log_2 N + 2N \) multiplications and \( 3N \log_2 N \) additions.
Solving the linear system with loop matrix

Let

\[ L = \begin{pmatrix} c_0 & c_{N-1} & \cdots & c_1 \\ c_1 & c_0 & \cdots & c_2 \\ \vdots & \vdots & \ddots & \vdots \\ c_{N-1} & c_{N-2} & \cdots & c_0 \end{pmatrix} \]

Solving \( Lx = b \). \( L \) is a loop matrix.

We have

\[
(Lx)_i = \sum_{j=0}^{N-1} c_{i-j}x_j
\]

where we assume \( c \) is period \( N \) with respect to the subscripts, and \( x = (x_0, x_1, \ldots, x_{N-1})^T \).
Solving the linear system with loop matrix

First consider the Jordan form of $L$. From the formula before

$$Lx = c * x = \lambda x$$

Take DFT we have

$$\hat{c} \circ \hat{x} = \lambda \hat{x}$$

then eigenvalues

$$\lambda_k = \hat{c}_k$$

The eigenvectors

$$\hat{x}_j^{(k)} = \delta_{kj}, \quad (j, k = 0, 1, \ldots, N - 1)$$

where $\delta_{kj}$ is Kronecker’s $\delta$.

Take inverse transform we obtain

$$x^{(0)} = (1, 1, \ldots, 1)^T,$$

$$x^{(1)} = (1, \omega^{-1}, \ldots, \omega^{-(N-1)})^T,$$

$$\ldots \ldots$$

$$x^{(N-1)} = (1, \omega^{-(N-1)}, \ldots, \omega^{-(N-1)^2})^T$$
Solving the linear system with loop matrix

- Spectral decomposition of $L$

$$L = \begin{pmatrix} x^{(0)} & x^{(1)} & \cdots & x^{(N-1)} \end{pmatrix} \begin{pmatrix} \lambda_0 & & & \\ & \lambda_1 & & \\ & & \ddots & \\ & & & \lambda_{N-1} \end{pmatrix} = (NF^{-1})\Lambda (NF^{-1})^{-1} = F^{-1}\Lambda F$$

- Solving $Lx = b$ is equivalent to $F^{-1}\Lambda Fx = b$, i.e. $\Lambda(Fx) = Fb$. Then it is composed of three steps:
  - Step 1: Compute $Fb$ i.e. apply FFT to $b$ to obtain $\hat{b}$;
  - Step 2: Compute $\Lambda$ i.e. apply FFT to $c$ to obtain $\hat{c}$;
  - Step 3: Compute $\hat{x}_k = \hat{b}_k / \hat{c}_k$, and then compute $(\hat{x})^\vee$ to obtain $x$. 

Examples

Fast Fourier Transform

Applications
Homework assignment

- Familiarize the “FFT” and “IFFT” command in MATLAB;
- Compute the convolution for

\[ h(x) = \int_0^{2\pi} \sin(x - y)e^{\cos y} dy \]