Fourier Transforms: Discrete, Fast, and Practical

PHYS 250 (Autumn 2024) – Lecture 12

David Miller

Department of Physics and the Enrico Fermi Institute
University of Chicago

November 7, 2024

Outline

Reminders from last time

We left off discussing details of our discrete Fourier transform and how we might speed it up.

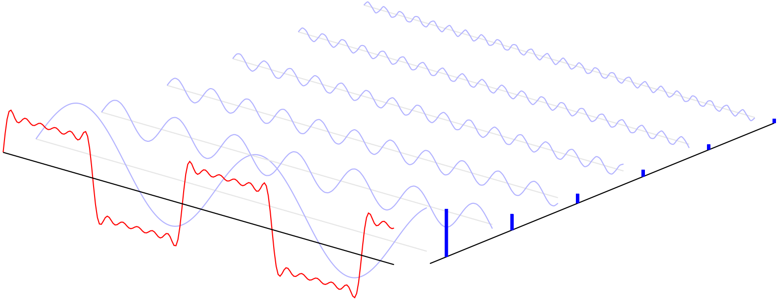
PDEs and Fourier Series

- **Fourier Series → Fourier Transforms**
  - We discussed how we can move to a continuous function definition of the expansion over a basis of functions
  - We then broke this down into discrete steps and obtained the **Discrete Fourier Transform**
- **Issues encountered:**
  - We realized that there is an issue related to the finite sampling of a function: **aliasing**
  - Began to break down the Fourier transform even further for a **fast** implementation

Today we will discuss the evolution towards the **FFT**, some of the practical limitations, and specific real-world (scientific and otherwise!) examples of using FFT's!

Square wave Fourier series

We already saw how we can break down a "simple" function into its components:



So let's figure out how to use this to its full capacity!

Outline

Discrete Fourier Transforms (I)

If $\hat{f}(\omega)$ or $f(t)$ are known analytically or numerically, the Fourier transform integrals can be evaluated using the integration techniques studied earlier.

Discrete Fourier Transforms (I)

If $\hat{f}(\omega)$ or $f(t)$ are known analytically or numerically, the Fourier transform integrals can be evaluated using the integration techniques studied earlier.

In practice, the signal $f(t)$ is measured, or **sampled** at just a finite number N of times t, and these are what we must use to approximate the transform.

Discrete Fourier Transforms (I)

If $\hat{f}(\omega)$ or $f(t)$ are known analytically or numerically, the Fourier transform integrals can be evaluated using the integration techniques studied earlier.

In practice, the signal $f(t)$ is measured, or **sampled** at just a finite number N of times t, and these are what we must use to approximate the transform.

The resultant **discrete Fourier transform (DFT)** is an approximation both because the signal is not known for all times and because we integrate numerically.

Discrete Fourier Transforms (I)

If $\hat{f}(\omega)$ or $f(t)$ are known analytically or numerically, the Fourier transform integrals can be evaluated using the integration techniques studied earlier.

In practice, the signal $f(t)$ is measured, or **sampled** at just a finite number N of times t, and these are what we must use to approximate the transform.

The resultant **discrete Fourier transform (DFT)** is an approximation both because the signal is not known for all times and because we integrate numerically.

Once we have a discrete set of transforms, they can be used to reconstruct the signal for any value of the time.

Discrete Fourier Transforms (I)

If $\hat{f}(\omega)$ or $f(t)$ are known analytically or numerically, the Fourier transform integrals can be evaluated using the integration techniques studied earlier.

In practice, the signal $f(t)$ is measured, or **sampled** at just a finite number N of times t, and these are what we must use to approximate the transform.

The resultant **discrete Fourier transform (DFT)** is an approximation both because the signal is not known for all times and because we integrate numerically.

Once we have a discrete set of transforms, they can be used to reconstruct the signal for any value of the time.

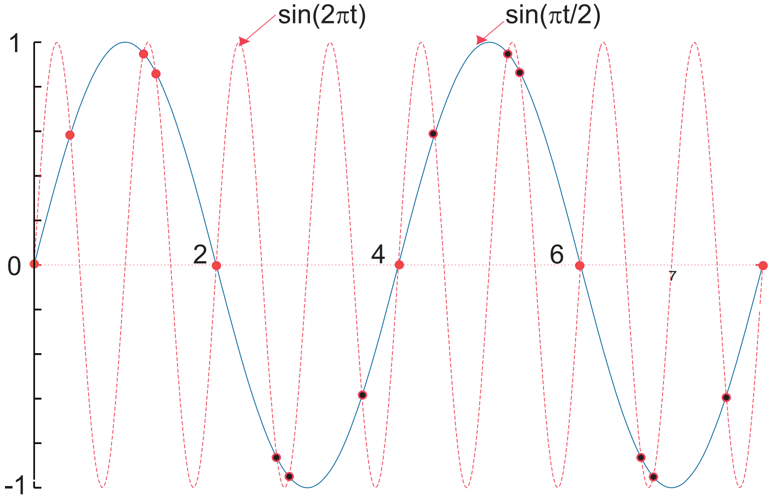
In this way the **DFT can be thought of as a technique for interpolating, compressing, and extrapolating data**.

Discussion

Do you see any issues with this “sampling”?

Discussion

Do you see any issues with this "sampling"?



Discrete Fourier Transforms (II)

The DFT algorithm results from evaluating the integral not from $-\infty$ to $+\infty$ but rather from time 0 to time T over which the signal is measured, and from approximating the integration of the integral by computing a discrete sum:

$$
\begin{aligned}
\hat{f}(\omega_n) &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(t) e^{-i\omega_n t} dt && (1) \\
&\simeq \frac{1}{\sqrt{2\pi}} \int_{0}^{T} f(t) e^{-i\omega_n t} dt && (2) \\
&\simeq \frac{1}{\sqrt{2\pi}} \sum_{k=1}^{N} h f(t_k) e^{-i\omega_n t_k} \qquad (h \equiv \text{stepsize}) && (3) \\
&\simeq \frac{h}{\sqrt{2\pi}} \sum_{k=1}^{N} f_k e^{-2\pi i k n/N} && (4) \\
\hat{f}_n \equiv \frac{\hat{f}(\omega_n)}{h} &= \frac{1}{\sqrt{2\pi}} \sum_{k=1}^{N} f_k e^{-2\pi i k n/N} && (5)
\end{aligned}
$$

Discrete Fourier Transforms (III)

We then need the inverse as well, which we can obtain with $d\omega \rightarrow 2\pi/Nh$ we invert the $\hat{f}_n$

$$f_k = \frac{1}{\sqrt{2\pi}} \sum_{n=1}^{N} \frac{2\pi}{Nh} \hat{f}_n e^{i\omega_n t} \tag{6}$$

Once we know the N values of the transform $\hat{f}_n$, we can use this expression to evaluate $f(t)$ for any time t. The frequencies ω_n are determined by the number of samples taken and by the total sampling time $T = Nh$ as

$$\omega_n = n\frac{2\pi}{Nh} \tag{7}$$

Clearly, the larger we make the time $T = Nh$ over which we sample the function, the smaller will be the frequency steps or resolution. Accordingly,if you want a smooth frequency spectrum, you need to have a small frequency step $2\pi/T$.

Discrete Fourier Transforms (IV)

Lastly, we can simplify this expression to yield a clear computational approach:

$$f_k = \frac{\sqrt{2\pi}}{N}\sum_{n=1}^{N} Z^{-nk}\hat{f}_n \qquad (Z = e^{-2\pi i/N}) \tag{8}$$

$$\hat{f}_n = \frac{1}{\sqrt{2\pi}}\sum_{k=1}^{N} Z^{nk} f_k \qquad (n = 0, 1, \cdots, N) \tag{9}$$

With this formulation, the computer needs to compute only powers of Z.

Recap of the Discrete Fourier Transform (DFT)

This is where we are at with the discretization of the Fourier Transform:

$$\hat{f}(\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(t) e^{-i\omega t} dt \quad \xrightarrow{DFT} \quad \hat{f}_n = \frac{1}{\sqrt{2\pi}} \sum_{k=1}^{N} f_k e^{-2\pi i k n/N} \tag{10}$$

$$f(t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \hat{f}(\omega) e^{i\omega t} d\omega \quad \xrightarrow{DFT} \quad f_k = \frac{\sqrt{2\pi}}{N} \sum_{n=1}^{N} \hat{f}_n e^{-i\omega_n t} \tag{11}$$

This has certain drawbacks which we will discuss shortly, but it also has huge advantages. Namely, we can re-write this to see some amazing computational properties.

$$\hat{f}_n = \frac{1}{\sqrt{2\pi}} \sum_{k=1}^{N} Z_N^{nk} f_k \qquad (Z_N = e^{-2\pi i/N}) \tag{12}$$

$$f_k = \frac{\sqrt{2\pi}}{N} \sum_{n=1}^{N} Z_N^{-nk} \hat{f}_n \qquad (n = 0, 1, \cdots, N) \tag{13}$$

*Moving to a **fast** DFT, aka Fast Fourier Transform*

We're saying that with this formulation, the computer needs to compute only powers of $Z \to Z_N^{nk}$.

What does this buy us, though???

*Moving to a **fast** DFT, aka Fast Fourier Transform*

We're saying that with this formulation, the computer needs to compute only powers of $Z \to Z_N^{nk}$.

What does this buy us, though???

Well, evaluating Eqs. **??**–**??** definition directly requires $\mathcal{O}(N^2)$ operations: there are N outputs f_k, and each output requires a sum of N terms.

*Moving to a **fast** DFT, aka Fast Fourier Transform*

We're saying that with this formulation, the computer needs to compute only powers of $Z \to Z_N^{nk}$.

What does this buy us, though???

Well, evaluating Eqs. **??**–**??** definition directly requires $\mathcal{O}(N^2)$ operations: there are N outputs f_k, and each output requires a sum of N terms.

What if we can make this scale as $N \ln N$???

*Moving to a **fast** DFT, aka Fast Fourier Transform*

We're saying that with this formulation, the computer needs to compute only powers of $Z \to Z_N^{nk}$.

What does this buy us, though???

Well, evaluating Eqs. **??**–**??** definition directly requires $\mathcal{O}(N^2)$ operations: there are N outputs f_k, and each output requires a sum of N terms.

What if we can make this scale as $N \ln N$???

This may not seem like much of a difference, for $N = 10^{2-3}$, the difference of 10^{3-5} is the difference between a minute and a week.

This is what the FFT buys us!

Cooley-Tukey algorithm

Let's start with the simplified form of the DFT:

$$\hat{f}_n = \frac{1}{\sqrt{2\pi}} \sum_{k=1}^{N} f_k e^{-2\pi i k n/N} = \frac{1}{\sqrt{2\pi}} \sum_{k=1}^{N} f_k Z_N^{nk} \tag{14}$$

First, **STOP AND LOOK**. **What are some features of this expression? What operations are required?**

- There are **imaginary components**
  - Even if the signal elements f_k to be transformed are real, Z_N is always complex, and therefore we must process both real and imaginary parts when computing transforms
- We have to add and/or multiply N^2 times unless we break this down further
  - Both n and k range over N integer values, the $(Z_N^n)^k f_k$ multiplications require N^2 multiplications and additions of complex numbers.

Cooley-Tukey algorithm

Let's start with the simplified form of the DFT:

$$\hat{f}_n = \frac{1}{\sqrt{2\pi}} \sum_{k=1}^{N} f_k e^{-2\pi i k n/N} = \frac{1}{\sqrt{2\pi}} \sum_{k=1}^{N} f_k Z_N^{nk} \tag{14}$$

First, **STOP AND LOOK**. **What are some features of this expression? What operations are required?**

- There are **imaginary components**
  - Even if the signal elements f_k to be transformed are real, Z_N is always complex, and therefore we must process both real and imaginary parts when computing transforms
- We have to add and/or multiply N^2 times unless we break this down further
  - Both n and k range over N integer values, the Z_N^{nk} multiplications require N^2 multiplications and additions of complex numbers

Cooley-Tukey algorithm

Let's start with the simplified form of the DFT:

$$\hat{f}_n = \frac{1}{\sqrt{2\pi}} \sum_{k=1}^{N} f_k e^{-2\pi ikn/N} = \frac{1}{\sqrt{2\pi}} \sum_{k=1}^{N} f_k Z_N^{nk} \tag{14}$$

First, **STOP AND LOOK**. **What are some features of this expression? What operations are required?**

- There are **imaginary components**
  - Even if the signal elements f_k to be transformed are real, Z_N is always complex, and therefore we must process both real and imaginary parts when computing transforms.
- We have to add and/or multiply N^2 times unless we break this down further
  - Both n and k range over N integer values, the $(Z_N^n)^k f_k$ multiplications require N^2 multiplications and additions of complex numbers.

Cooley-Tukey algorithm

Let's start with the simplified form of the DFT:

$$\hat{f}_n = \frac{1}{\sqrt{2\pi}} \sum_{k=1}^{N} f_k e^{-2\pi ikn/N} = \frac{1}{\sqrt{2\pi}} \sum_{k=1}^{N} f_k Z_N^{nk} \tag{14}$$

First, **STOP AND LOOK**. **What are some features of this expression? What operations are required?**

- There are **imaginary components**
  - Even if the signal elements f_k to be transformed are real, Z_N is always complex, and therefore we must process both real and imaginary parts when computing transforms.
- We have to add and/or multiply N^2 times unless we break this down further
  - Both n and k range over N integer values, the $(Z_N^n)^k f_k$ multiplications require N^2 multiplications and additions of complex numbers.

Cooley-Tukey algorithm

Let's start with the simplified form of the DFT:

$$\hat{f}_n = \frac{1}{\sqrt{2\pi}} \sum_{k=1}^{N} f_k e^{-2\pi ikn/N} = \frac{1}{\sqrt{2\pi}} \sum_{k=1}^{N} f_k Z_N^{nk} \tag{14}$$

First, **STOP AND LOOK**. **What are some features of this expression? What operations are required?**

- There are **imaginary components**
  - Even if the signal elements f_k to be transformed are real, Z_N is always complex, and therefore we must process both real and imaginary parts when computing transforms.
- We have to add and/or multiply N^2 times unless we break this down further
  - Both n and k range over N integer values, the $(Z_N^n)^k f_k$ multiplications require N^2 multiplications and additions of complex numbers.

Cooley-Tukey algorithm

Let's start with the simplified form of the DFT:

$$\hat{f}_n = \frac{1}{\sqrt{2\pi}} \sum_{k=1}^{N} f_k e^{-2\pi ikn/N} = \frac{1}{\sqrt{2\pi}} \sum_{k=1}^{N} f_k Z_N^{nk} \tag{14}$$

First, **STOP AND LOOK**. **What are some features of this expression? What operations are required?**

- There are **imaginary components**
  - Even if the signal elements f_k to be transformed are real, Z_N is always complex, and therefore we must process both real and imaginary parts when computing transforms.
- We have to add and/or multiply N^2 times unless we break this down further
  - Both n and k range over N integer values, the $(Z_N^n)^k f_k$ multiplications require N^2 multiplications and additions of complex numbers.

Example Cooley-Tukey algorithm (I)

The time savings comes from taking advantage of **periodicity**.

The values of the **computationally expensive complex factor** $Z^{nk} = ((Z)^n)^k$ **are repeated** as the integers n and k vary sequentially.

For instance, for $N = 8$ (*leaving out subscript and* $1/\sqrt{2\pi}$, *but note this is* Z_8^{nk}):

Example Cooley-Tukey algorithm (I)

The time savings comes from taking advantage of **periodicity**.

The values of the **computationally expensive complex factor** $Z^{nk} = ((Z)^n)^k$ **are repeated** as the integers n and k vary sequentially.

For instance, for $N = 8$ (*leaving out subscript and* $1/\sqrt{2\pi}$, *but note this is* Z_8^{nk}):

$$\hat{f}_{n=0} \quad = \quad Z^{nk=0} f_{k=0} + Z^{nk=0} f_1 + Z^0 f_2 + Z^0 f_3 + Z^0 f_4 + Z^0 f_5 + Z^0 f_6 + Z^0 f_7$$

Example Cooley-Tukey algorithm (I)

The time savings comes from taking advantage of **periodicity**.

The values of the **computationally expensive complex factor** $Z^{nk} = ((Z)^n)^k$ **are repeated** as the integers n and k vary sequentially.

For instance, for $N = 8$ (*leaving out subscript and* $1/\sqrt{2\pi}$, *but note this is* Z_8^{nk}):

$$\begin{aligned}
\hat{f}_{n=0} &= Z^{nk=0} f_{k=0} + Z^{nk=0} f_1 + Z^0 f_2 + Z^0 f_3 + Z^0 f_4 + Z^0 f_5 + Z^0 f_6 + Z^0 f_7 \\
\hat{f}_{n=1} &= Z^{nk=0} f_{k=0} + Z^{nk=1} f_1 + Z^2 f_2 + Z^3 f_3 + Z^4 f_4 + Z^5 f_5 + Z^6 f_6 + Z^7 f_7
\end{aligned}$$

Example Cooley-Tukey algorithm (I)

The time savings comes from taking advantage of **periodicity**.

The values of the **computationally expensive complex factor** $Z^{nk} = ((Z)^n)^k$ **are repeated** as the integers n and k vary sequentially.

For instance, for $N = 8$ (*leaving out subscript and* $1/\sqrt{2\pi}$, *but note this is* Z_8^{nk}):

$$\begin{aligned}
\hat{f}_{n=0} &= Z^{nk=0} f_{k=0} + Z^{nk=0} f_1 + Z^0 f_2 + Z^0 f_3 + Z^0 f_4 + Z^0 f_5 + Z^0 f_6 + Z^0 f_7 \\
\hat{f}_{n=1} &= Z^{nk=0} f_{k=0} + Z^{nk=1} f_1 + Z^2 f_2 + Z^3 f_3 + Z^4 f_4 + Z^5 f_5 + Z^6 f_6 + Z^7 f_7 \\
\hat{f}_{n=2} &= Z^{nk=0} f_{k=0} + Z^{nk=2} f_1 + Z^4 f_2 + Z^6 f_3 + Z^8 f_4 + Z^{10} f_5 + Z^{12} f_6 + Z^{14} f_7
\end{aligned}$$

Example Cooley-Tukey algorithm (I)

The time savings comes from taking advantage of **periodicity**.

The values of the **computationally expensive complex factor $Z^{nk} = ((Z)^n)^k$ are repeated** as the integers n and k vary sequentially.

For instance, for $N = 8$ (*leaving out subscript and $1/\sqrt{2\pi}$, but note this is Z_8^{nk}*):

$$\begin{aligned}
\hat{f}_{n=0} &= Z^{nk=0}f_{k=0} + Z^{nk=0}f_1 + Z^0f_2 + Z^0f_3 + Z^0f_4 + Z^0f_5 + Z^0f_6 + Z^0f_7 \\
\hat{f}_{n=1} &= Z^{nk=0}f_{k=0} + Z^{nk=1}f_1 + Z^2f_2 + Z^3f_3 + Z^4f_4 + Z^5f_5 + Z^6f_6 + Z^7f_7 \\
\hat{f}_{n=2} &= Z^{nk=0}f_{k=0} + Z^{nk=2}f_1 + Z^4f_2 + Z^6f_3 + Z^8f_4 + Z^{10}f_5 + Z^{12}f_6 + Z^{14}f_7 \\
\hat{f}_{n=3} &= Z^{nk=0}f_{k=0} + Z^{nk=3}f_1 + Z^6f_2 + Z^9f_3 + Z^{12}f_4 + Z^{15}f_5 + Z^{18}f_6 + Z^{21}f_7 \\
\hat{f}_{n=4} &= Z^{nk=0}f_{k=0} + Z^{nk=4}f_1 + Z^8f_2 + Z^{12}f_3 + Z^{16}f_4 + Z^{20}f_5 + Z^{24}f_6 + Z^{28}f_7 \\
\hat{f}_{n=5} &= Z^{nk=0}f_{k=0} + Z^{nk=5}f_1 + Z^{10}f_2 + Z^{15}f_3 + Z^{20}f_4 + Z^{25}f_5 + Z^{30}f_6 + Z^{35}f_7 \\
\hat{f}_{n=6} &= Z^{nk=0}f_{k=0} + Z^{nk=6}f_1 + Z^{12}f_2 + Z^{18}f_3 + Z^{24}f_4 + Z^{30}f_5 + Z^{36}f_6 + Z^{42}f_7 \\
\hat{f}_{n=7} &= Z^{nk=0}f_{k=0} + Z^{nk=7}f_1 + Z^{14}f_2 + Z^{21}f_3 + Z^{28}f_4 + Z^{35}f_5 + Z^{42}f_6 + Z^{49}f_7
\end{aligned}$$

Example Cooley-Tukey algorithm (II)

There are actually only 4 independent values! Z^0, Z^1, Z^2, Z^3

$$Z^0 = \exp(0) = +1, \qquad Z^1 = \exp(-\frac{2\pi}{8}i) = +\frac{\sqrt{2}}{2} - i\frac{\sqrt{2}}{2}$$

$$Z^2 = \exp(-\frac{2\pi}{8}2i) = -i, \qquad Z^3 = \exp(-\frac{2\pi}{8}3i) = -\frac{\sqrt{2}}{2} - i\frac{\sqrt{2}}{2}$$

$$Z^4 = \exp(-\frac{2\pi}{8}4i) = -Z^0, \qquad Z^5 = \exp(-\frac{2\pi}{8}5i) = -Z^1$$

$$Z^6 = \exp(-\frac{2\pi}{8}6i) = -Z^2, \qquad Z^7 = \exp(-\frac{2\pi}{8}7i) = -Z^3$$

$$Z^8 = \exp(-\frac{2\pi}{8}8i) = +Z^0, \qquad Z^9 = \exp(-\frac{2\pi}{8}9i) = +Z^1$$

$$Z^{10} = \exp(-\frac{2\pi}{8}10i) = +Z^2, \qquad Z^{11} = \exp(-\frac{2\pi}{8}11i) = +Z^3$$

$$Z^{12} = \exp(-\frac{2\pi}{8}11i) = -Z^0, \qquad \cdots$$

Example Cooley-Tukey algorithm (III)

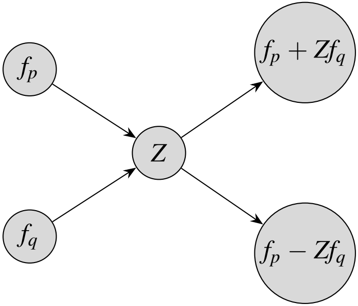
We can now put these equations in an appropriate form for computing by regrouping the terms into sums and differences of the f's:

$$
\begin{aligned}
\hat{f}^0 &= Z^0(f_0+f_4)+Z^0(f_1+f_5)+Z^0(f_2+f_6)+Z^0(f_3+f_7)\\
\hat{f}^1 &= Z^0(f_0-f_4)+Z^1(f_1-f_5)+Z^2(f_2-f_6)+Z^3(f_3-f_7)\\
\hat{f}^2 &= Z^0(f_0+f_4)+Z^2(f_1+f_5)-Z^0(f_2+f_6)-Z^2(f_3+f_7)\\
\hat{f}^3 &= Z^0(f_0-f_4)+Z^3(f_1-f_5)-Z^2(f_2-f_6)+Z^1(f_3-f_7)\\
\hat{f}^4 &= Z^0(f_0+f_4)-Z^0(f_1+f_5)+Z^0(f_2+f_6)-Z^0(f_3+f_7)\\
\hat{f}^5 &= Z^0(f_0-f_4)-Z^1(f_1-f_5)+Z^2(f_2-f_6)-Z^3(f_3-f_7)\\
\hat{f}^6 &= Z^0(f_0+f_4)-Z^2(f_1+f_5)-Z^0(f_2+f_6)+Z^2(f_3+f_7)\\
\hat{f}^7 &= Z^0(f_0-f_4)-Z^3(f_1-f_5)-Z^2(f_2-f_6)-Z^1(f_3-f_7)\\
\hat{f}^8 &= \hat{f}^0.
\end{aligned}
\tag{15}
$$

Butterfly calculations (I)

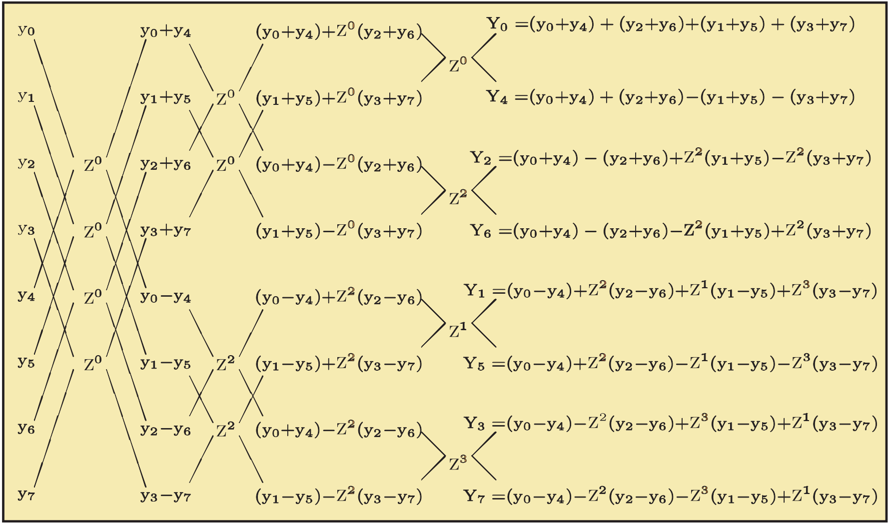
Now comes the real magic, and something that is used all over the place in fast, hardware-based calculations:

→ notice the **repeating factors inside the parentheses**, they have the form $f_p \pm f_q$. These symmetries are systematized by introducing the **butterfly operation**.



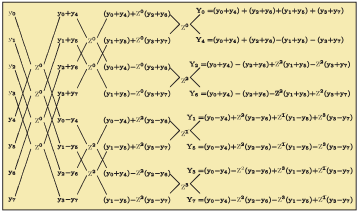
Butterfly calculations (II)

With the mapping $y \to f, Y \to \hat{f}$, this looks like a **network of complex additions and multiplications** for our $N = 8$ FFT:



Butterfly calculations (II)

With the mapping $y \to f, Y \to \hat{f}$, this looks like a **network of complex additions and multiplications** for our $N = 8$ FFT:



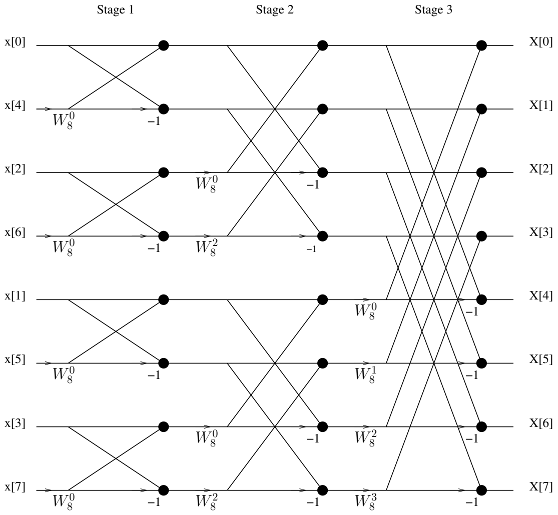
Notice how the number of multiplications of complex numbers has been reduced:

- For the first butterfly operation there are 8 multiplications by Z^0
- For the second butterfly operation there are 8 multiplications
- A total of 24 multiplications is made in four butterfly operations

Butterfly calculations (III)

This is often written in a slightly different form **(notice anything?)**:

$N = 8$-point decimation-in-time FFT algorithm



$W_N = \mathrm{e}^{-\jmath \frac{2\pi}{N}}$

Danielson-Lanczos Lemma

The discrete Fourier transform of length N (where N is even) can be rewritten as the **sum of two discrete Fourier transforms**, each of length $N/2$, one for **even-numbered** points and the other for **odd-numbered** points.

$$\begin{aligned}
\hat{f}_n &= \frac{1}{\sqrt{2\pi}} \sum_{k=1}^{N} f_k Z_N^{nk} &(16)\\
&= \frac{1}{\sqrt{2\pi}} \sum_{k=1}^{N/2} f_{2k}\, Z_{N/2}^{nk} + Z_N^n \sum_{k=1}^{N/2} Z_{N/2}^{nk} f_{2k+1} &(17)\\
&= \hat{f}_n^{\text{even}} + Z_N^n \hat{f}_n^{\text{odd}}, &(18)
\end{aligned}$$

In fact, this procedure can be **applied recursively** to break up the $N/2$ even and odd points to their $N/4$ even and odd points.

If N is a power of 2, this procedure breaks up the original transform into $\ln N$ transforms of length 1.