The Fourier series

A large class of phenomena can be described as periodic in nature:
waves, sounds, light, radio, water waves etc.

It is natural to attempt to describe these phenomena by means of *expansions in periodic functions.*

The Fourier series is an expansion of a function in terms of trigonometric sines and cosines:

Suppose *f(x)* is defined over a finite range $-L \le x \le L$, i.e. f(x) is periodic with period $2L$. The trigonometric functions are periodic with period $2L = 2\pi$, so it is natural to expand these functions in terms of trigonometric functions with an argument $[(x / 2L)\, 2\pi n]$, $n \in \mathrm{N}$:

$$\boxed{f(x) = \frac{1}{2}a_0 + \sum_{m=1}^{\infty} a_m \cos\left(\frac{n\pi}{L}x\right) + \sum_{m=1}^{\infty} b_m \sin\left(\frac{n\pi}{L}x\right)}$$

$$\boxed{a_n = \frac{1}{L}\int_{-L}^{L} f(x)\cos\left(\frac{n\pi}{L}x\right)dx} \quad \boxed{b_n = \frac{1}{L}\int_{-L}^{L} f(x)\sin\left(\frac{n\pi}{L}x\right)dx}$$

Example for Fourier series

Let's take the following function:

$$f(x) = \begin{cases} |x| & ; -8 <= x <= 8 \\ f(x \pm 16n) \ , \ n \in \mathbb{N} & ; \text{ otherwise} \end{cases}$$

- $f(x)$ is even in x while $sin(x)$ is odd => b_n's must be zero.

$$a_n = \frac{2}{L}\int_0^L x\cos(\frac{n\pi}{L}x)dx = \frac{2L}{n^2\pi^2}[\cos(n\pi)-1] = -\frac{4L}{n^2\pi^2} \qquad ; \text{ if } n \text{ is odd}$$

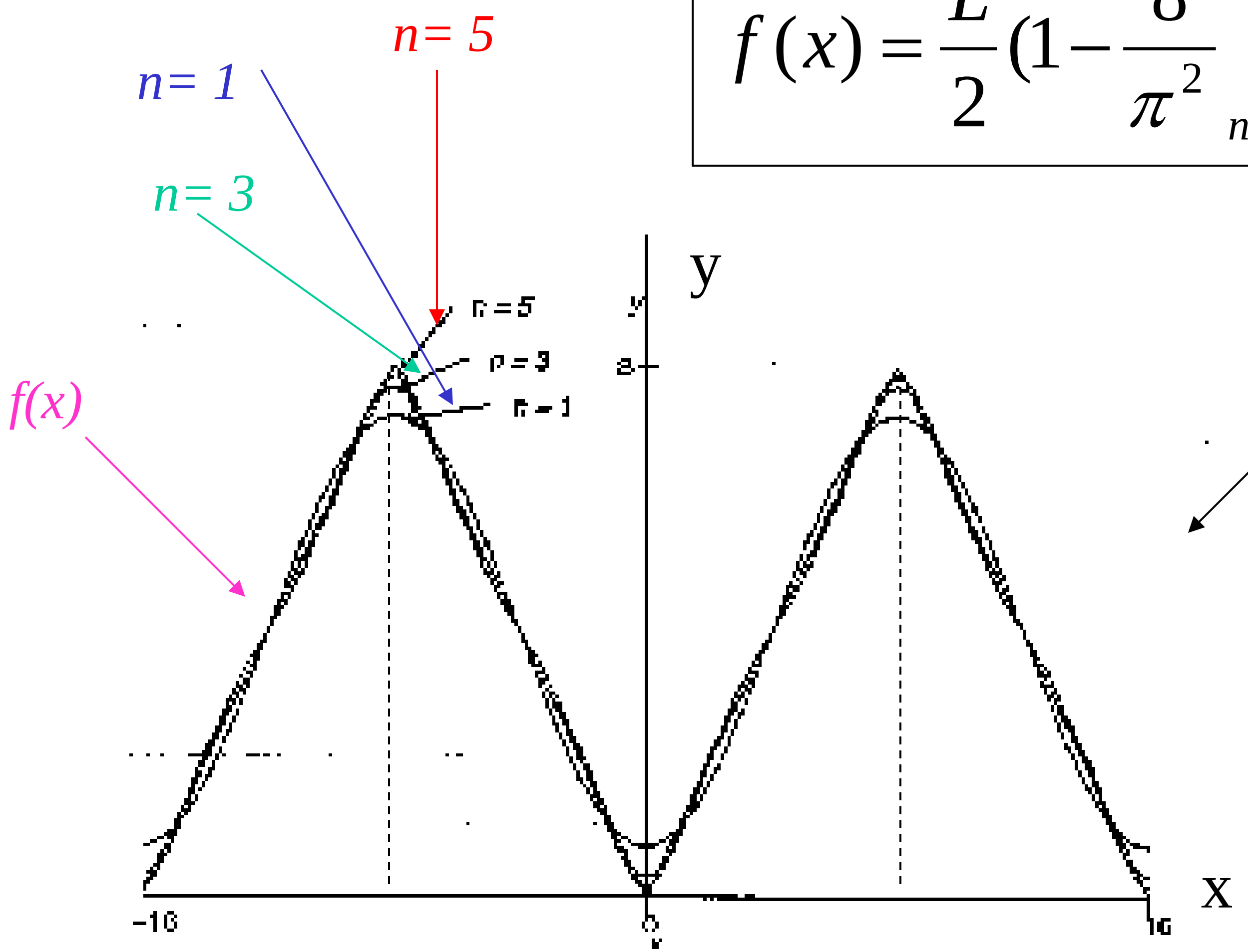
$$a_n = \frac{2}{L}\int_0^L x\cos(\frac{n\pi}{L}x)dx = \frac{2L}{n^2\pi^2}[\cos(n\pi)-1] = 0 \qquad ; \text{ if } n \text{ is even}$$

=> the expansion is :

$$\boxed{f(x) = \frac{L}{2}(1-\frac{8}{\pi^2}\sum_{n=odd}^{\infty}\frac{1}{n^2}\cos(\frac{n\pi}{L}x))}$$

Example for Fourier series(2)

source: $f(x) = \|x\|$, $-8 <= x <= 8$
$f(x) = \frac{L}{2}\left(1 - \frac{8}{\pi^2}\sum_{n=odd}^{\infty}\frac{1}{n^2}\cos\left(\frac{n\pi}{L}x\right)\right)$



Successive approximations of *f(x)*

Complex version of the Fourier expansion

The Euler identity: $e^{i\Theta} = \cos\Theta + i\sin\Theta$

The inverse equations:

$$\sin\Theta = \tfrac{1}{2i}(e^{i\Theta} - e^{-i\Theta}) \quad , \quad \cos\Theta = \tfrac{1}{2}(e^{i\Theta} + e^{-i\Theta})$$

Using the formulas above and some properties of exponential function, the Fourier series can also be written as an expansion in terms of complex exponentials as:

1

$$f(x) = \sum_{n=-\infty}^{\infty} c_n e^{(in\pi/L)x} \quad , \quad c_n = \frac{1}{2L}\int_{-L}^{L} f(x) e^{-i(n\pi/L)x} dx$$

Note: you can read the full explanation here.

The Fourier transform

Let's define $w = {}^{n\pi}\!/_{L}$ => $dw = {}^{\pi}\!/_{L}$.

The **Fourier transform** is a generalization of the complex Fourier series in the limit as $L \to \infty$ on the formula we got on the previous slide:

1
$$f(x) = \sum_{n=-\infty}^{\infty} c_n e^{(in\pi/L)x} \quad , \quad c_n = \frac{1}{2L}\int_{-L}^{L} f(x)e^{-i(n\pi/L)x}dx$$

2
$$\hat{f}(x) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} f(x)e^{-iwx}dx$$

The inverse Fourier transform

By placing the formula $\boxed{2}$ in the formula $\boxed{1}$ we get the **inverse Fourier transform** formula:

1

$$f(x) = \sum_{n=-\infty}^{\infty} c_n e^{(in\pi/L)x} \quad , \quad c_n = \frac{1}{2L}\int_{-L}^{L} f(x)e^{-i(n\pi/L)x}dx$$

$$\Downarrow$$

3

$$f(x) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} \hat{f}(x)e^{iwx}dw$$

Note: you can read the full explanation here

The discrete Fourier transform

Motivation: computer applications of the Fourier transform require that all of the definitions and properties of Fourier transforms be translated into analogous statements appropriate to functions represented by a discrete set of sampling points rather than by continuous functions.

Let $f(x)$ be a function.
Let $\{f_k = f(x_k)\}$ be a set of N function values, $x_k = k\Delta x$ $\quad k = 0, 1, \ldots, N\text{-}1$.
Let Δx be the separation of the equidistant sampling points.
Assumption: N is even.

The **discrete Fourier transform** is:

$$\boxed{\hat{f}_n = \sum_{k=0}^{N} (e^{2\pi i/N})^{nk} f_k, n = 0,1,\ldots, N-1}$$

The **inverse discrete transform** is:

$$\boxed{f_k = \frac{1}{N+1} \sum_{n=0}^{N} (e^{2\pi i/N})^{-nk} \hat{f}_n, k = 0,1,\ldots, N-1}$$

The discrete Fourier transform(2)

Let's examine more closely the formula of the **discrete Fourier transform**:

$$\boxed{\hat{f}_n = \sum_{k=0}^{N} (e^{2\pi i/N})^{nk} f_k, n = 0,1,\ldots, N-1}$$

We know that $w_N = \frac{2\pi i}{N}$ (it's called n-th root of unity), so the formula above can be rewritten as:

$$\boxed{\hat{f}_n = \sum_{k=0}^{N} w_n{}^k f_k, n = 0,1,\ldots, N-1 \Longrightarrow \hat{f}_n = \hat{f}_n(w_n)}$$

Usage of the Fast Fourier Transform - motivation

Let $A(x) = \sum_{k=0}^{n-1} a_k x^k$, $B(x) = \sum_{k=0}^{n-1} b_k x^k$ be two polynomials.

We want to multiply them: $C(x) = A(x)*B(x)$.

Two ways to do this:

1. $C(x) = \sum_{j=0}^{2n-2} c_j x^j$, where $c_j = \sum_{k=0}^{j} a_k b_{j-k}$ **- takes time** $\Theta(n^2)$

2. (a) calculate $A(x)$ and $B(x)$ values in $2n-1$ distinct points $x_0, \ldots, x_{2n-2}$;
   we get two vectors $\{(x_0, y_0),\ldots,(x_{2n-2}, y_{2n-2})\}$, $\{(x_0,z_0),\ldots,(x_{2n-2}, z_{2n-2})\}$;
   (b) multiply these vectors: $\{(x_0, y_0 z_0),\ldots,(x_{2n-2}, y_{2n-2} z_{2n-2})\}$
   (c) calculate the polynomial $C(x)$ that passes through the result vector (interpolation)

   - takes time $\Theta(n \log n)$ **if use** FFT

Uniqueness of *C(x)*

Theorem1: for any set $\{(x_0, y_0),\ldots,(x_{n-\backslash}, y_{n-1})\}$ on n distinct points there is a unique polynomial $C(x)$ with degree less than n such that $y_i = C(x_i)$ for $i = 0,1,\ldots, n-1$.

Proof: we can write $y_i = C(x_i) = \sum_{k=0}^{n-1} c_k x_i^{\,k}$ for $i = 0,1,\ldots, n-1$ as matrices multiplication:

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n\text{-}1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n\text{-}1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n\text{-}1} & x_{n\text{-}1}^2 & \cdots & x_{n\text{-}1}^{n\text{-}1} \end{bmatrix} * \begin{pmatrix} c_0 \\ c_1 \\ \ldots \\ c_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \ldots \\ y_{n-1} \end{pmatrix}$$

The matrix $V(x_0,\ldots, x_{n-\backslash})$ is called a Vandermonde matrix.
Since all $x_0,\ldots, x_{n-\backslash}$ are distinct, a discriminate of V isn't zero, so it is reversible.
=> we can calculate $c_i = V(x_0,\ldots, x_{n-\backslash})^{-1}\, y_i$. ■

Discrete FFT and FFTinverse

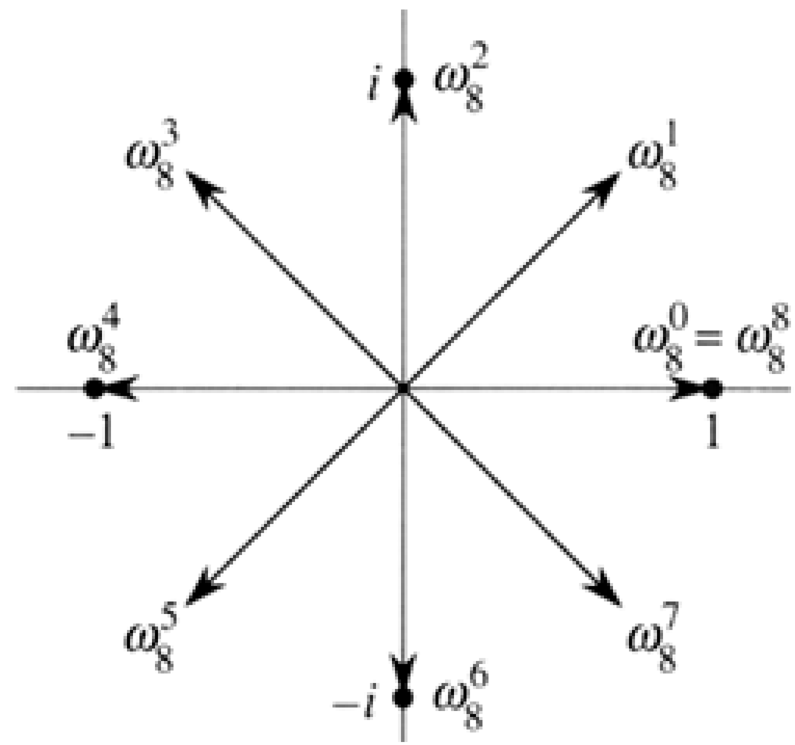
We will use FFT to execute step (a) and FFTinv to execute step (c).

Let $C(x) = \sum_{k=0}^{n-1} c_k x^k$ be a polynomial.

We want to execute a step (a): calculate $C(x)$ in n distinct points $x_0, \ldots, x_{n-1}$.
FFT uses special n points – $w_n^{\,0}, w_n^{\,1}, w_n^{\,2}, \ldots, w_n^{\,n-1}$.

w_n is called the n-th (complex) root of unity.
That means that $w_n^{\,n} = 1$.

Figure 32.2 The values of $\omega_8^0, \omega_8^1, \ldots, \omega_8^7$ in the complex plane, where $\omega_8 = e^{2\pi i/8}$ is the principal 8th root of unity.



Properties of n-th root of unity

Let's examine some properties of w_n :

1. There are exactly n n-th roots of unity: $w_n^0, w_n^1, w_n^2, \ldots, w_n^{n-1}$.

Each one of them can be presented as $e^{2\pi i/n}$.

w_n is called the principal n-th root of unity.

2. The inverse of w_n is $w_n^{-1} = w_n^{n-1}$: $w_n * w_n^{-1} = w_n^0 = 1;$

$$w_n * w_n^{n-1} = w_n^n = 1.$$

3. $w_{dn}^{dk} = w_n^k$: $w_{dn}^{dk} = \left(e^{2\pi i/dn}\right)^{dk} = \left(e^{2\pi i/n}\right)^k = w_n^k$.

4. $w_n^{n/2} = w_2 = -1$: $w_n^{n/2} = w_{(n/2)*2}^{n/2}$ = (by property 3) $w_2 = e^{2\pi i/2} = -1$.

Properties of n-th root of unity(2)

5. If n > 0 and n is even => $(w_n^k)^2 = w_{n/2}^k$, $k = 0, 1, \ldots, n-1$:

. $(w_n^k)^2 = (e^{2\pi i/n})^{2k} = (e^{2\pi i/n/2})^k = w_{n/2}^k$.

. $(w_n^{k+n/2})^2 = w_n^{2k+n} = w_n^{2k} w_n^n = w_n^{2k} = (w_n^k)^2 = w_{n/2}^k$.

6. $\boxed{\sum_{j=0}^{n-1} (w_n^k)^j = 0}$: $\sum_{j=0}^{n-1} (w_n^k)^j = \frac{(w_n^k)^n - 1}{w_n^k - 1} = \frac{(w_n^n)^k - 1}{w_n^k - 1} = \frac{(1)^k - 1}{w_n^k - 1} = 0$.

geometry series formula

DFFT preview

Let $p_c(x) = \sum_{k=0}^{n-1} c_k x^k$ be a polynomial.

We want to execute a step (a): calculate $p_c(x)$ in w_n^{0}, w_n^{1}, w_n^{2}, …, w_n^{n-1}.

Observation:

Let $c = (c_0, c_1, \ldots, c_{n-1})$ be an n-tuple of the coefficients of $p_c(x)$.

Assume that *n* is power of 2

Let $a = (c_0, c_2, \ldots, c_{n-2})$ and $b = (c_1, c_3, \ldots, c_{n-1})$

=> $p_c(x) = p_a(x^2) + x p_b(x^2)$, where p_a is the polynomial that is defined by the vector a, and p_b is the polynomial that is defined by the vector b.

=> $p_c(w_n^i) = p_a(w_n^{2i}) + w_n^i p_b(w_n^{2i})$

Let $t = n/2$ => $w_n^t = w_n^{n/2} = -1$.

=> for $i < n/2$: $p_c(w_n^i) = p_a(w_n^{2i}) + w_n^i p_b(w_n^{2i})$

for $i >= n/2$: $p_c(w_n^i) = p_a(w_n^{2i}) + w_n^{j+t} p_b(w_n^{2i}) = p_a(w_n^{2i}) - w_n^j p_b(w_n^{2i})$

DFFT algorithm

```
DFFT ( c = (c_0, c_1, ..., c_{n-1}) , w_n) :
    // n is a power of 2 and w_n^{n/2} = -1 //
    array C[0, …, n – 1]  // for the answer
    if n = = 1 then C[0] ← c[0]
    else
        t ← n / 2
        arrays a, b, A, B [0,…,t – 1] // intermediate arrays
        for i ← 0 to t – 1 do:
            a[i] ← c[2i]
            b[i] ← c[2i + 1]
        // recursive Fourier transform computation
        A ← DFFT (a, w_n^2 )
        B ← DFFT (b, w_n^2 )
        // Fourier transform computation of the vector c
        for i ← 0 to t – 1 do:
            temp = w_n^i
            C[i] ← A[i] + temp * B[i]
            C[t + i] ← A[i] – temp * B[i]
            temp ← temp * w_n
    return C  // return the answer
```

DFFT algorithm - example

Let $\boldsymbol{n = 8}$ and $\boldsymbol{c = (255, 8, 0, 226, 37, 240, 3, 0)}$.

Let $\boldsymbol{F_{257}}$ be a finite field => $\boldsymbol{w_8 = 4}$ $(4^8 \bmod 257 = 1)$.

1) $a = (255, 0, 37, 3)$, $b = (8, 226, 240, 0)$

2) recursive call with $t = 8 / 2 = 4$ and $w_8^2 = 4^2 = 16$:

$$A = [38, 170, 32, 9], \quad B = [217, 43, 22, 7]$$

3) $C[0] \leftarrow 38 + 217 = 255$
$C[1] \leftarrow 170 + 43 * w_8 = 85$
$C[2] \leftarrow 32 + 22 * w_8^2 = 127$
$C[3] \leftarrow 9 + 7 * w_8^3 = 200$
$C[4] \leftarrow 38 - 217$
$C[5] \leftarrow 170 - 43 * w_8 = 255$
$C[6] \leftarrow 32 - 22 * w_8^2 = 194$
$C[7] \leftarrow 9 - 7 * w_8^3 = 75$

4) The final result is: $\boldsymbol{C = (255, 85, 127, 200, 78, 255, 194, 75)}$.

DFFT algorithm – execution time

We have *log(n)* recursive calls.

For each call:

- n multiplications $w_8^{i+1} \leftarrow w_8^i * w_8$
- n multiplications of const * w_8^i , $i = 0, 1, \ldots, n / 2 - 1$
- $n / 2$ additions, $n / 2$ subtractions

=> $\Theta(n)$ arithmetical instructions, each costs $\Theta(1)$

=> $\boxed{\text{DFFT algorithm execution time is } \Theta(n \log n)}$

DFFTinv algorithm - preview

We want to execute a step (c) calculate the polynomial $C(x)$ that passes through the result vector (interpolation)

We can write $y_i = C(w_n^{\,i}) = \sum_{k=0}^{n-1} c_k w_n^{\,ik}$ for $i = 0,1,\ldots, n-1$ as matrices multiplication:

$$\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & w_n & w_n^2 & \cdots & w_n^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & w_n^{n-1} & w_n^{2\,n-1} & \cdots & w^{n-1\,n-1} \end{bmatrix} * \begin{pmatrix} c_0 \\ c_1 \\ \ldots \\ c_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \ldots \\ y_{n-1} \end{pmatrix}$$

The matrix $V_{ij}(x_0, \ldots, x_{n-1})$ is called a Vandermonde matrix.
Since all $x_0, \ldots, x_{n-1}$ are distinct, a discriminate of V_{ij} isn't zero, so it is reversible.
=> we can calculate $c_i = V_{ij}(x_0, \ldots, x_{n-1})^{-1} y_i$.

DFFTinv algorithm – preview(2)

Theorem: let V_{ij} (x_0 ,..., $x_{n-\backslash}$) be the matrix as above.
Then the inverse matrix of V_{ij} is $V_{ij}^{-1} = n^{-1} w_n^{-ij}$.

Proof: we already saw that V_{ij}^{-1} exists.

$$V_{ij} * V_{ij}^{-1} = \sum_{k=0}^{n-1} V_{ik} V_{kj} = n^{-1} \sum_{k=0}^{n-1} w_n^{(i-j)k}$$

i. if i = j, then $w_n^{(i-j)k} = w_n^{0} = 1$, and so $n^{-1} \sum_{k=0}^{n-1} 1 = n * -n = 1$

ii. else, then $n^{-1} \sum_{k=0}^{n-1} w_n^{(i-j)k} = n^{-1} * 0 = 0$ (by property 6)

So we get that $V_{ij} * V_{ij}^{-1} = I_n$. ■

DFFTinv algorithm

```
DFFTinv (C[0, …, n – 1] , w_n) :
    // n is a power of 2 and w_n^{n/2} = -1 //
    array c[0, …, n – 1]  // for the answer
    c ← DFFT (C[0, …, n – 1] , w_n^{n-1} ) // remember that w_n^{-1} = w_n^{n-1}
    for i ← 0 to n – 1 do:
        c[i] ← n^{-1} * c[i]
    return c  // return the answer
```

DFFTinv algorithm - example

Let ***n* = 8** and ***C* = (255, 85, 127, 200, 78, 255, 194, 75).**
Let F_{257} be a finite field => $w_8 = 4$ $(4^8 \bmod 257 = 1)$.

1) $w_8^{-1} = w_8^7 = 193$.

2) calculate DFFT(C, 193):

$$c = [241, 64, 0, 9, 39, 121, 24, 0]$$

3) multiply c by $n^{-1} = 225$ $(225 * 8 \bmod 257 = 1)$

4) The final result is: ***c* = (255, 8, 0, 226, 37, 240, 3, 0).**

DFFTinv algorithm – execution time

DFFT takes $\Theta(n \log n)$ time .

n multiplications c[i] $\leftarrow n^{-1}$ * c[i]

=> $\Theta(n)$ arithmetical instructions, each costs $\Theta(1)$

=> $\boxed{\text{DFFTinv algorithm execution time is } \Theta(n \log n)}$