FFTs in Graphics and Vision

The Fast Fourier Transform

Outline

The FFT Algorithm

Applications in 1D

Multi-Dimensional FFTs

More Applications

Real FFTs

Computational Complexity

What do we need to do in order to compute the moving dot-product of two periodic, *n*-dimensional arrays *f*[ ] and *g*[ ]?

1. We need to express *f*[ ] and *g*[ ] in terms of the basis v_k[ ]:

$$f[\,] = \sum_{k=0}^{n-1} \hat{f}[k] v_k[\,] \quad \text{and} \quad g[\,] = \sum_{k=0}^{n-1} \hat{g}[k] v_k[\,] \qquad \boxed{O(n^2)}$$

2. We need to multiply the coefficients:

$$\left(f[\,] * g[\,]\right)[\,] = \sqrt{n} \sum_{k=0}^{n-1} \hat{f}[k] \overline{\hat{g}[k]} v_k[\,] \qquad \boxed{O(n)}$$

3. We need to evaluate the moving dot-product at every index α:

$$\left(f[\,] * g[\,]\right)[\alpha] = \sqrt{n} \sum_{k=0}^{n-1} \hat{f}[k] \overline{\hat{g}[k]} v_k[\alpha] \qquad \boxed{O(n^2)}$$

Goal

Given an *n*-dimensional array of complex values *f*[ ], we would like to compute the Fourier coefficients of *f*[ ]:

$$f[\,] = \sum_{k=0}^{n-1} \hat{f}[k] v_k[\,]$$

where *v*$_k$[ ] are the discrete samples of the complex exponentials at *n* regularly spaced positions:

$$v_k[\,] = \sqrt{\frac{1}{n}}\left(e^{ik2\pi 0/n}, e^{ik2\pi 1/n}, \ldots, e^{ik2\pi(n-2)/n}, e^{ik2\pi(n-1)/n}\right)$$

Challenge

How can we compute all n Fourier coefficients efficiently?

Challenge

How can we compute all *n* Fourier coefficients?

Challenge

How can we compute all *n* Fourier coefficients?

Since the vectors $v_k[\,]$ are orthonormal, we can compute the *k*-th Fourier coefficient of $f[\,]$ by computing the dot-product:

$$\hat{f}[k] = \langle f[\,], v_k[\,] \rangle$$

Challenge

How can we compute all *n* Fourier coefficients?

Since the vectors $v_k[\,]$ are orthonormal, we can compute the *k*-th Fourier coefficient of $f[\,]$ by computing the dot-product:

$$\begin{aligned}\hat{f}[k] &= \left\langle f[\,], v_k[\,] \right\rangle \\ &= \sum_{j=0}^{n-1} f[j] \cdot \overline{v_k[j]}\end{aligned}$$

Challenge

How can we compute all *n* Fourier coefficients?

Since the vectors $v_k[\,]$ are orthonormal, we can compute the *k*-th Fourier coefficient of *f*[ ] by computing the dot-product:

$$
\begin{aligned}
\hat{f}[k] &= \left\langle f[\,], v_k[\,] \right\rangle \\
&= \sum_{j=0}^{n-1} f[j] \cdot \overline{v_k[j]} \\
&= \sqrt{\frac{1}{n}} \sum_{j=0}^{n-1} f[j] \cdot e^{-ik2\pi j/n}
\end{aligned}
$$

Challenge

How can we compute all *n* Fourier coefficients?

Since the vectors $v_k[\ ]$ are orthonormal, we can compute the *k*-th Fourier coefficient of *f*[ ] by computing

Computing one coefficient is: O(*n*)

$$\hat{f}[k] = \langle f[\ ], v_k[\ ] \rangle$$

$$= \sum_{j=0}^{n-1} f[j] \cdot \overline{v_k[j]}$$

$$= \sqrt{\frac{1}{n}} \sum_{j=0}^{n-1} f[j] \cdot e^{-ik2\pi j/n}$$

Challenge

How can we compute all *n* Fourier coefficients?

Since the vectors $v_k[\,]$ are orthonormal, we can compute the *k*-th Fourier coefficient of $f[\,]$ by computin

Computing one coefficient is: O(*n*)

Computing all the coefficient is: O(n^2)

$$= \sqrt{\frac{1}{n}} \sum_{j=0}^{n-1} f[j] \cdot e^{-ik2\pi j/n}$$

Challenge

How can we compute all n Fourier coefficients efficiently?

Approach (Divide and Conquer)

Key Idea:

If we decompose the array into the even and odd halves, we can solve smaller problems and then combine.

Approach (Divide and Conquer)

Key Idea:

Consider the 8-dimensional array:

$$f[\,] = \{f_0, f_1, f_2, f_3, f_4, f_5, f_6, f_7\}$$

And consider its even/odd decomposition:

$$f_0[\,] = \{f_0, f_2, f_4, f_6\}$$
$$f_1[\,] = \{f_1, f_3, f_5, f_7\}$$

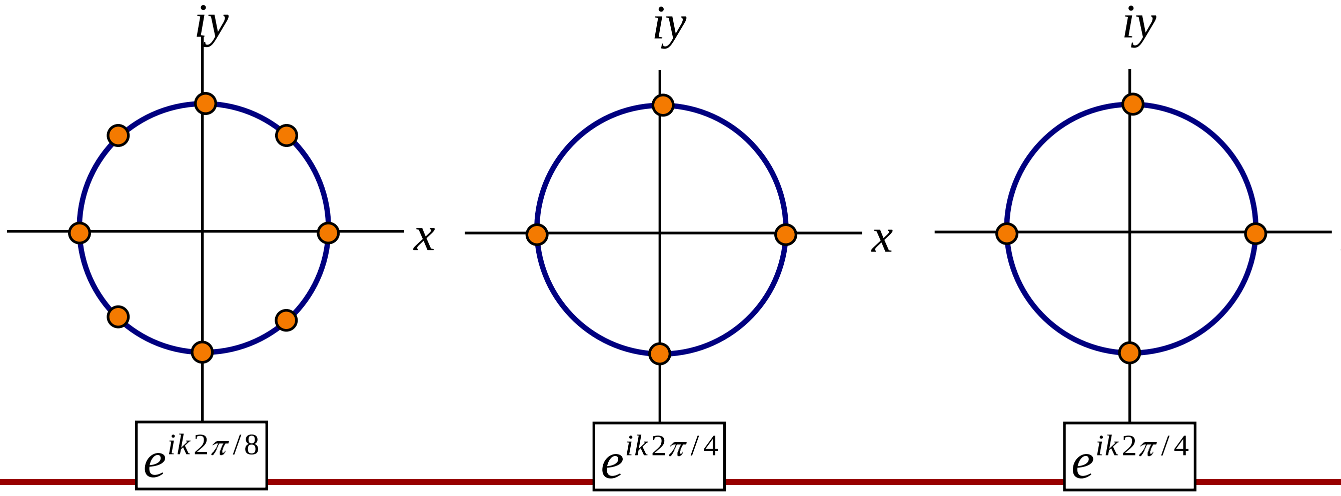
Approach (Divide and Conquer)

Key Idea:

$$f[\,] = \left(f_0, f_1, f_2, f_3, f_4, f_5, f_6, f_7 \right)$$
$$f_0[\,] = \left(f_0, f_2, f_4, f_6 \right)$$
$$f_1[\,] = \left(f_1, f_3, f_5, f_7 \right)$$

Now let's consider the Fourier coefficients:

$$\hat{f}[k] = \sqrt{\frac{1}{n}} \sum_{j=0}^{n-1} f[j] \cdot e^{-ik2\pi j/n}$$



Approach (Divide and Conquer)

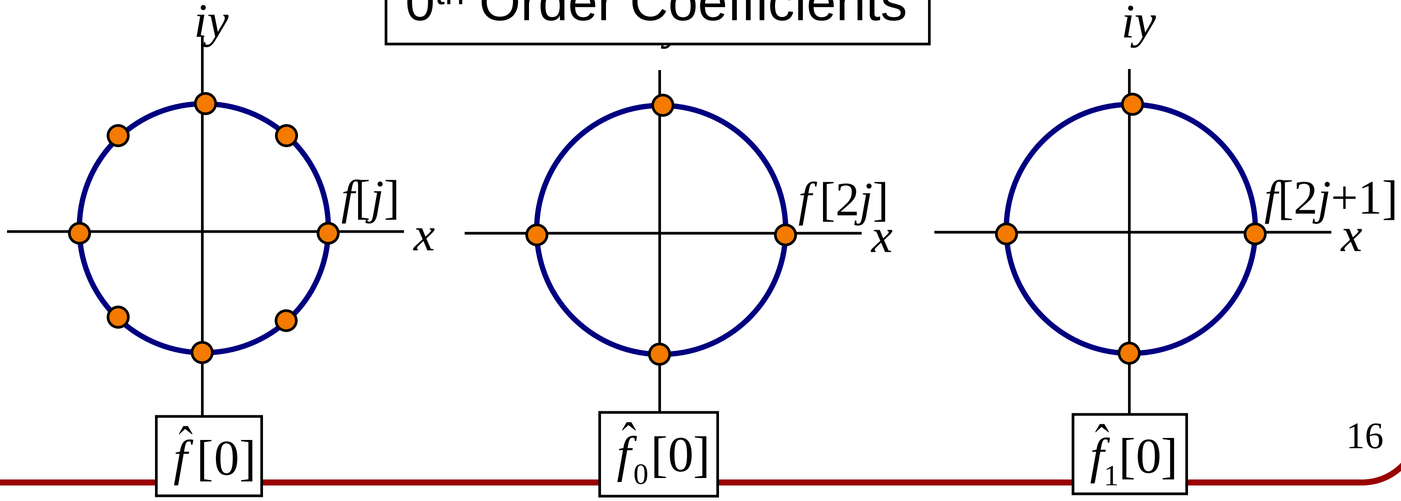
Key Idea:

$$f[\,] = \{f_0, f_1, f_2, f_3, f_4, f_5, f_6, f_7\}$$
$$f_0[\,] = \{f_0, f_2, f_4, f_6\}$$
$$f_1[\,] = \{f_1, f_3, f_5, f_7\}$$

Now let's consider the Fourier coefficients:

$$\hat{f}[k] = \sqrt{\frac{1}{n}} \sum_{j=0}^{n-1} f[j] \cdot e^{-ik2\pi j/n}$$

0th Order Coefficients



Approach (Divide and Conquer)

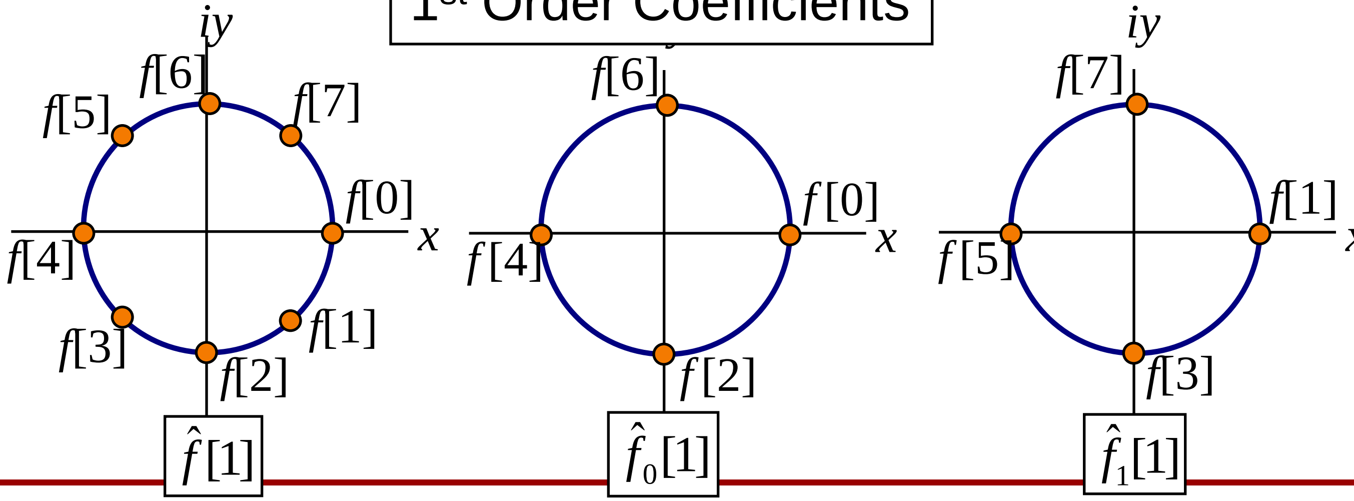
Key Idea:

$$f[\,] = \left(f_0, f_1, f_2, f_3, f_4, f_5, f_6, f_7\right)$$
$$f_0[\,] = \left(f_0, f_2, f_4, f_6\right)$$
$$f_1[\,] = \left(f_1, f_3, f_5, f_7\right)$$

Now let's consider the Fourier coefficients:

$$\hat{f}[k] = \sqrt{\frac{1}{n}} \sum_{j=0}^{n-1} f[j] \cdot e^{-ik2\pi j/n}$$

1st Order Coefficients



Approach (Divide and Conquer)

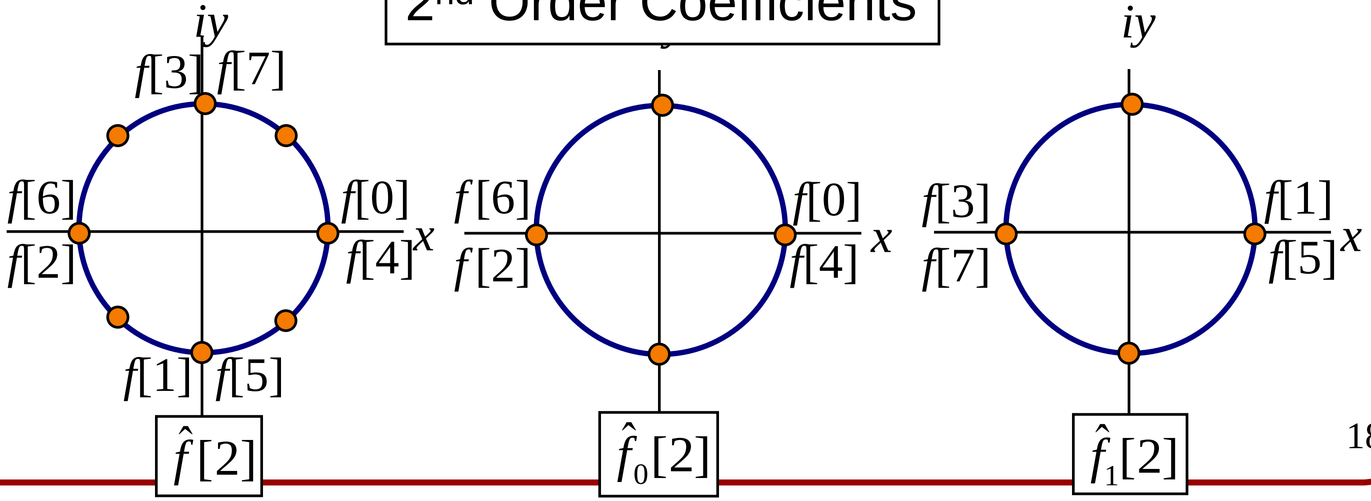
Key Idea:

$$f[\,] = (f_0, f_1, f_2, f_3, f_4, f_5, f_6, f_7)$$
$$f_0[\,] = (f_0, f_2, f_4, f_6)$$
$$f_1[\,] = (f_1, f_3, f_5, f_7)$$

Now let's consider the Fourier coefficients:

$$\hat{f}[k] = \sqrt{\frac{1}{n}} \sum_{j=0}^{n-1} f[j] \cdot e^{-ik2\pi j/n}$$

2nd Order Coefficients



Approach (Divide and Conquer)

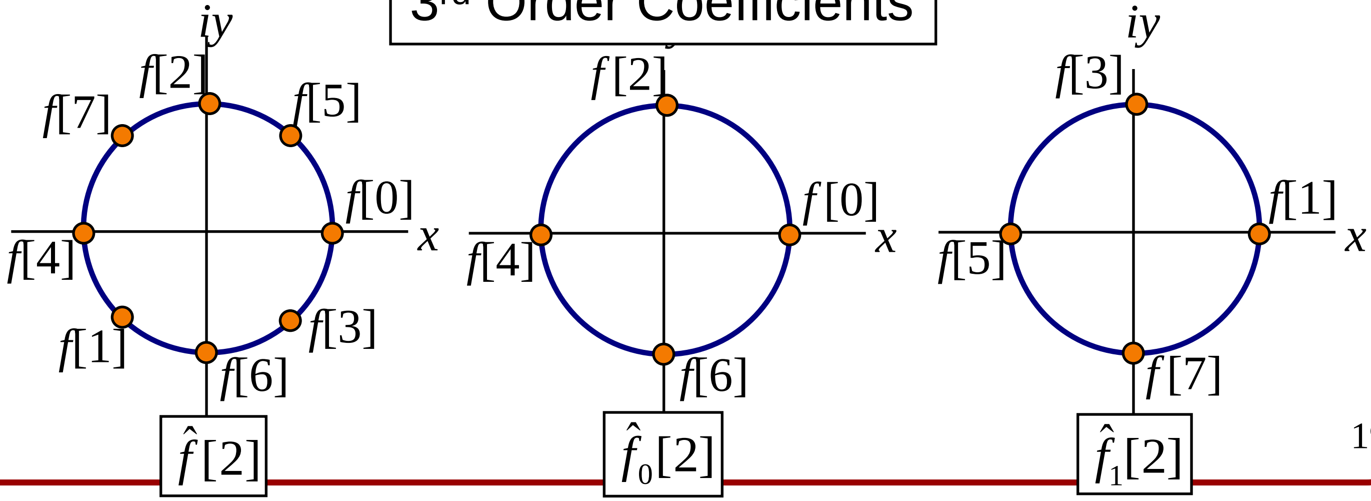
Key Idea:

$$f[\,] = \left(f_0, f_1, f_2, f_3, f_4, f_5, f_6, f_7\right)$$
$$f_0[\,] = \left(f_0, f_2, f_4, f_6\right)$$
$$f_1[\,] = \left(f_1, f_3, f_5, f_7\right)$$

Now let's consider the Fourier coefficients:

$$\hat{f}[k] = \sqrt{\frac{1}{n}} \sum_{j=0}^{n-1} f[j] \cdot e^{-ik2\pi j/n}$$

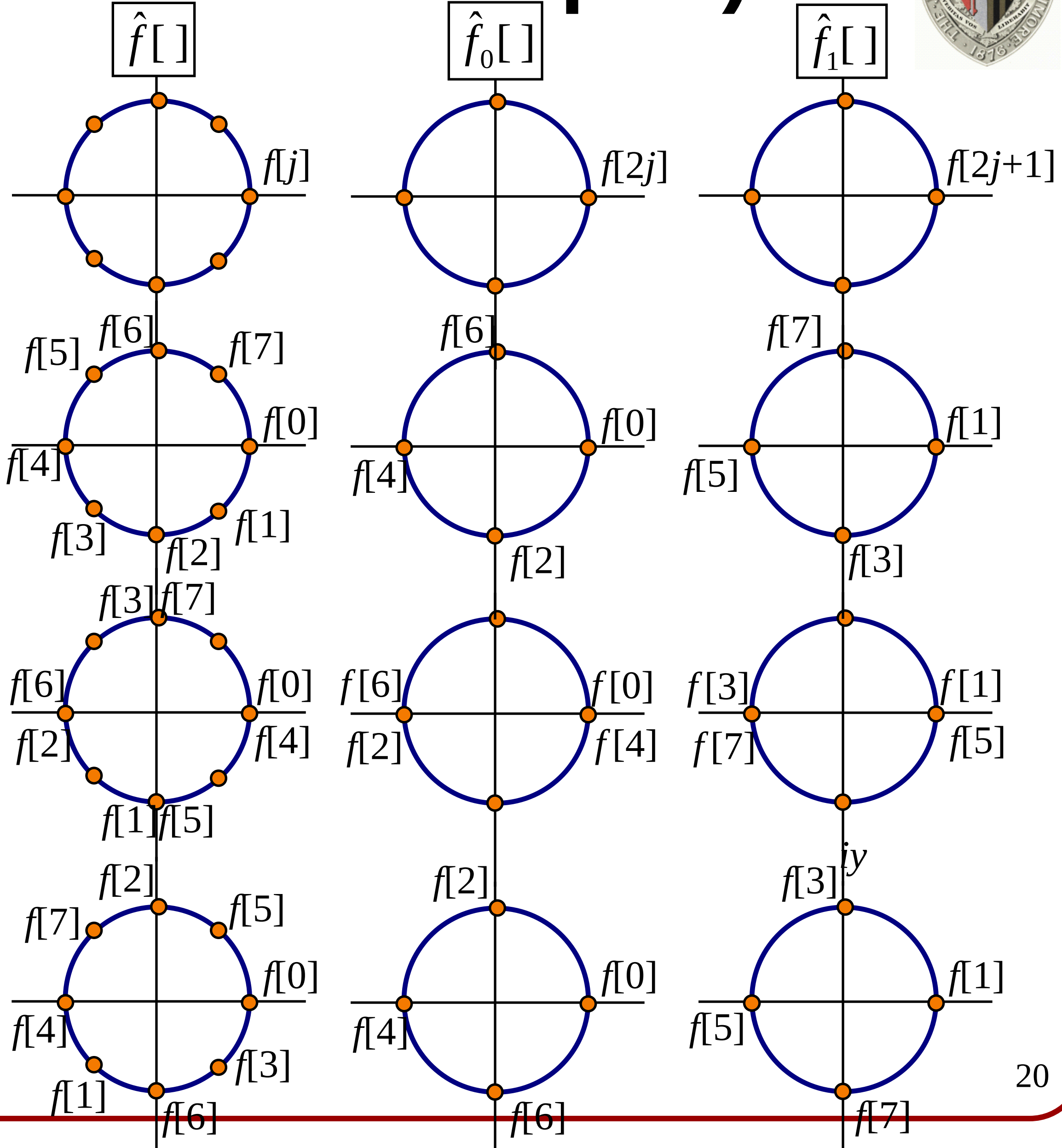
3rd Order Coefficients



Approach (Divide and Conquer)

For every frequency:

- The even components are in the same position as the original
- The odd ones are off by a fixed angle.



$\hat{f}[\,]$
$\hat{f}_0[\,]$
$\hat{f}_1[\,]$
$f[j]$
$f[2j]$
$f[2j+1]$
$f[6]$ $f[5]$ $f[7]$ $f[0]$ $f[4]$ $f[3]$ $f[1]$ $f[2]$
$f[6]$ $f[0]$ $f[4]$ $f[2]$
$f[7]$ $f[1]$ $f[5]$ $f[3]$
$f[3]$ $f[7]$ $f[6]$ $f[0]$ $f[2]$ $f[4]$ $f[1]$ $f[5]$
$f[6]$ $f[0]$ $f[2]$ $f[4]$
$f[3]$ $f[1]$ $f[7]$ $f[5]$
iy
$f[2]$ $f[7]$ $f[5]$ $f[0]$ $f[4]$ $f[3]$ $f[1]$ $f[6]$
$f[2]$ $f[0]$ $f[4]$ $f[6]$
$f[3]$ $f[1]$ $f[5]$ $f[7]$

Approach (Divide and Conquer)

Assume that n is even ($n=2m$) and lets consider the even and odd entries separately. That is, let $f_0[\,]$ and $f_1[\,]$ be the m-dimensional arrays:

$$f_0[k] = f[2k] \qquad\qquad f_1[k] = f[2k+1]$$

Approach (Divide and Conquer)

$$\boxed{f_0[k] = f[2k] \qquad\qquad f_1[k] = f[2k+1]}$$

The *k*-th Fourier coefficient of *f* is defined as:

$$\hat{f}[k] = \sqrt{\frac{1}{n}} \sum_{j=0}^{n-1} f[j] \cdot e^{-ik2\pi j/n}$$

Approach (Divide and Conquer)

$$\boxed{f_0[k] = f[2k] \qquad\qquad f_1[k] = f[2k+1]}$$

The *k*-th Fourier coefficient of *f* is defined as:

$$\hat{f}[k] = \sqrt{\frac{1}{n}} \sum_{j=0}^{n-1} f[j] \cdot e^{-ik2\pi j/n}$$

Splitting this summation into the even and odd terms gives:

$$\hat{f}[k] = \sqrt{\frac{1}{n}} \sum_{j=0}^{m-1} \left(f[2j] \cdot e^{-ik2\pi(2j)/(2m)} + f[2j+1] \cdot e^{-ik2\pi(2j+1)/(2m)} \right)$$

Approach (Divide and Conquer)

$$\boxed{f_0[k] = f[2k] \qquad\qquad f_1[k] = f[2k+1]}$$

Splitting the summation we got:

$$\hat{f}[k] = \sqrt{\frac{1}{n}} \sum_{j=0}^{m-1} \left(f[2j] \cdot e^{-ik2\pi(2j)/(2m)} + f[2j+1] \cdot e^{-ik2\pi(2j+1)/(2m)} \right)$$

Re-writing the exponent, we get:

$$\hat{f}[k] = \sqrt{\frac{1}{n}} \sum_{j=0}^{m-1} \left(f[2j] \cdot e^{-ik2\pi j/m} + f[2j+1] \cdot e^{-ik2\pi j/m} \cdot e^{-ik2\pi/n} \right)$$

Approach (Divide and Conquer)

$$\boxed{f_0[k] = f[2k] \qquad\qquad f_1[k] = f[2k+1]}$$

Simplifying the exponent gave:

$$\hat{f}[k] = \sqrt{\frac{1}{n}} \sum_{j=0}^{m-1} \left(f[2j] \cdot e^{-ik2\pi j/m} + f[2j+1] \cdot e^{-ik2\pi j/m} \cdot e^{-ik2\pi/n} \right)$$

Plugging in our expression for the even and odd entries gives:

$$\hat{f}[k] = \sqrt{\frac{1}{n}} \sum_{j=0}^{m-1} \left(f_0[j] \cdot e^{-ik2\pi j/m} + f_1[j] \cdot e^{-ik2\pi j/m} \cdot e^{-ik2\pi/n} \right)$$

Approach (Divide and Conquer)

$$\boxed{f_0[k] = f[2k] \qquad\qquad f_1[k] = f[2k+1]}$$

Plugging in our expression for the even and odd entries gives:

$$\hat{f}[k] = \sqrt{\frac{1}{n}} \sum_{j=0}^{m-1} \left(f_0[j] \cdot e^{-ik2\pi j/m} + f_1[j] \cdot e^{-ik2\pi j/m} \cdot e^{-ik2\pi/n} \right)$$

Re-writing this equation gives:

$$\hat{f}[k] = \sqrt{\frac{1}{n}} \sum_{j=0}^{m-1} f_0[j] \cdot e^{-ik2\pi j/m} + e^{-ik2\pi/n} \cdot \sqrt{\frac{1}{n}} \sum_{j=0}^{m-1} f_1[j] \cdot e^{-ik2\pi j/m}$$

Approach (Divide and Conquer)

$$\boxed{f_0[k] = f[2k] \qquad\qquad f_1[k] = f[2k+1]}$$

Re-writing this equation gave:

$$\hat{f}[k] = \sqrt{\frac{1}{n}}\sum_{j=0}^{m-1} f_0[j]\cdot e^{-ik2\pi j/m} + e^{-ik2\pi/n}\cdot\sqrt{\frac{1}{n}}\sum_{j=0}^{m-1} f_1[j]\cdot e^{-ik2\pi j/m}$$

But the interior summations resemble Fourier coefficients:

$$\hat{f}[k] = \sqrt{\frac{m}{n}}\left(\hat{f}_0[k] + \hat{f}_1[k]\cdot e^{-ik2\pi/n}\right)$$

Approach (Divide and Conquer)

$$\boxed{f_0[k] = f[2k] \qquad\qquad f_1[k] = f[2k+1]}$$

$$\boxed{\hat{f}[k] = \sqrt{\frac{m}{n}}\left(\hat{f}_0[k] + \hat{f}_1[k]\cdot e^{-ik2\pi/n}\right)}$$

So, if we can figure out the Fourier coefficients of $f_0[\ ]$ and $f_1[\ ]$, we can combine them to get the Fourier coefficients of $f[\ ]$.

Approach (Divide and Conquer)

$$\boxed{f_0[k] = f[2k] \qquad\qquad f_1[k] = f[2k+1]}$$

$$\boxed{\hat{f}[k] = \sqrt{\frac{m}{n}}\left(\hat{f}_0[k] + \hat{f}_1[k]\cdot e^{-ik2\pi/n}\right)}$$

So, if we can figure out the Fourier coefficients of f_0[ ] and f_1[ ], we can combine them to get the Fourier coefficients of *f*[ ].

Assuming that *m* is also even, we can repeat for f_0[ ] and f_1[ ] to get their Fourier coefficients.

Approach (Divide and Conquer)

$$\boxed{f_0[k] = f[2k] \qquad\qquad f_1[k] = f[2k+1]}$$

$$\boxed{\hat{f}[k] = \sqrt{\frac{m}{n}}\left(\hat{f}_0[k] + \hat{f}_1[k]\cdot e^{-ik2\pi/n}\right)}$$

So, if we can figure out the Fourier coefficients of $f_0[\,]$ and $f_1[\,]$, we can combine them to get the Fourier coefficients of $f[\,]$.

Assuming that m is also even, we can repeat for $f_0[\,]$ and $f_1[\,]$ to get their Fourier coefficients.

Thus, if n is a power of 2, we can keep separating, giving an O(n log n) algorithm.

Outline

The FFT Algorithm

Applications in 1D

Multi-Dimensional FFTs

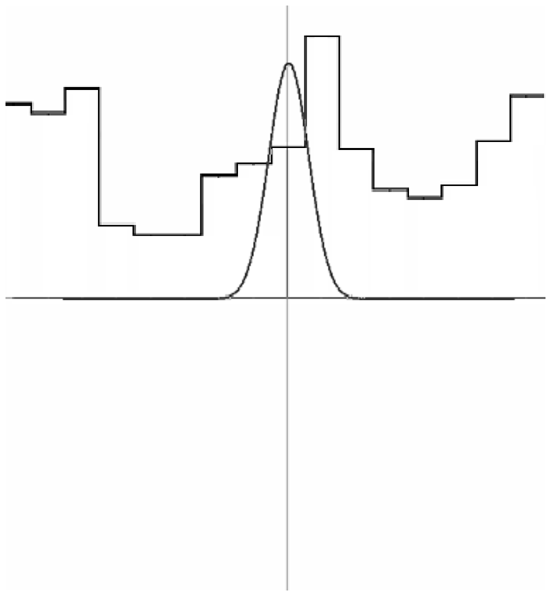
More Applications

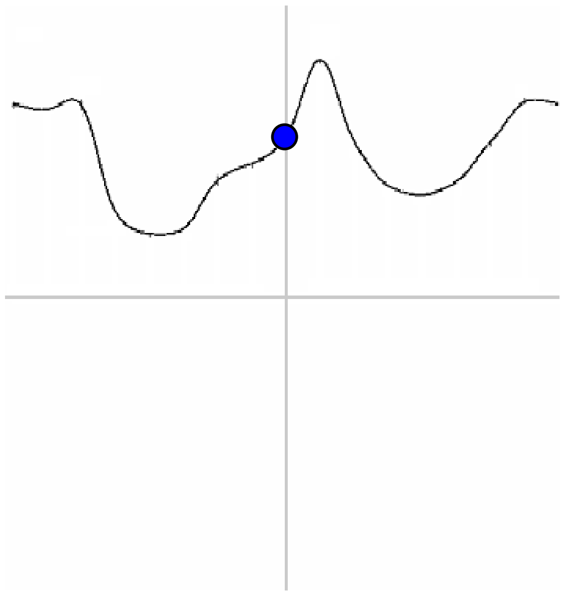
Real FFTs

Applications of the FFT

- Moving Dot Products / Cross Correlation

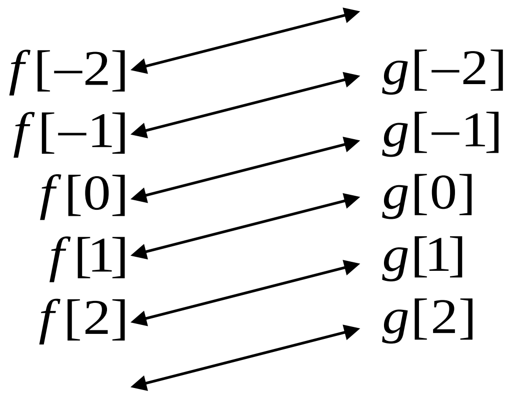
$$\begin{tikzcd}
f[-2] \arrow[r, leftrightarrow] & g[-2] \\
f[-1] \arrow[r, leftrightarrow] & g[-1] \\
f[0] \arrow[r, leftrightarrow] & g[0] \\
f[1] \arrow[r, leftrightarrow] & g[1] \\
f[2] \arrow[r, leftrightarrow] & g[2]
\end{tikzcd}$$

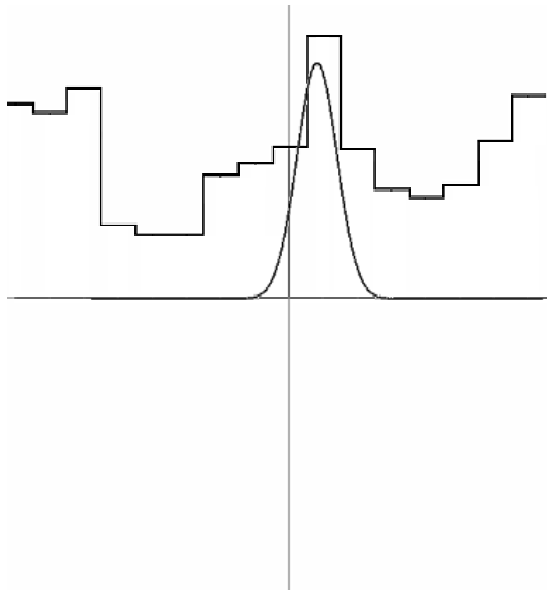


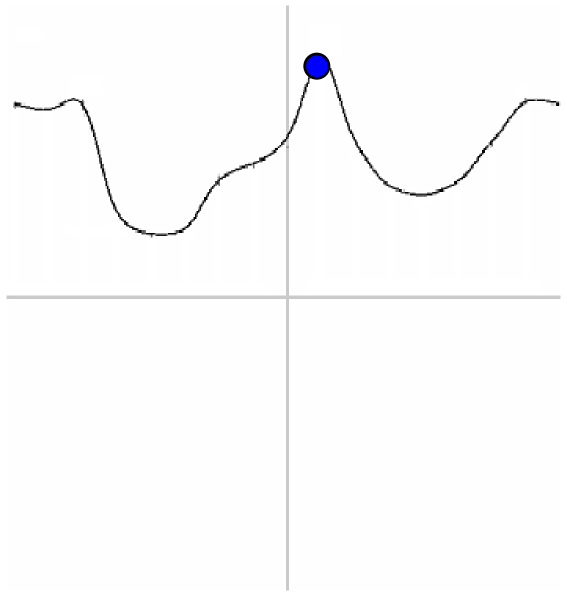


Applications of the FFT

- Moving Dot Products / Cross Correlation



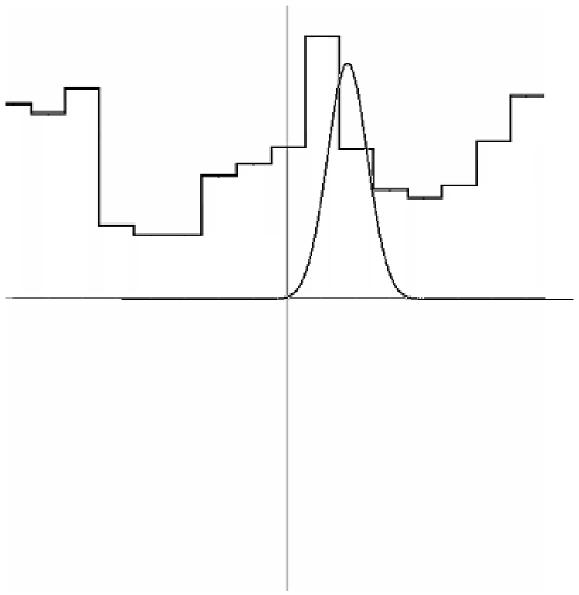


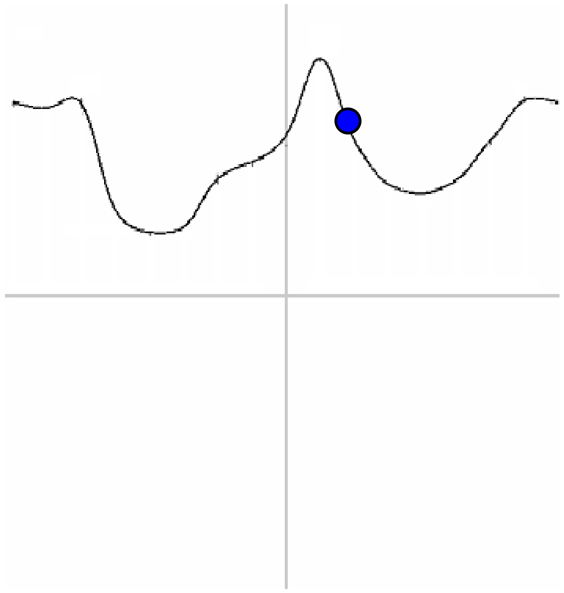


Applications of the FFT

- Moving Dot Products / Cross Correlation

$$\begin{tikzcd}
 & {} \\
 & {} \\
f[-2] \arrow[uur, leftrightarrow] & g[-2] \\
f[-1] \arrow[uur, leftrightarrow] & g[-1] \\
f[0] \arrow[uur, leftrightarrow] & g[0] \\
f[1] \arrow[uur, leftrightarrow] & g[1] \\
f[2] \arrow[uur, leftrightarrow] & g[2] \\
{} \arrow[uur, leftrightarrow] & \\
{} \arrow[uur, leftrightarrow] &
\end{tikzcd}$$





Applications of the FFT

- Moving Dot Products / Cross Correlation
- Convolution

  This is like cross-correlation, except that flip the entries of the array *g*[ ] before cross-correlating.

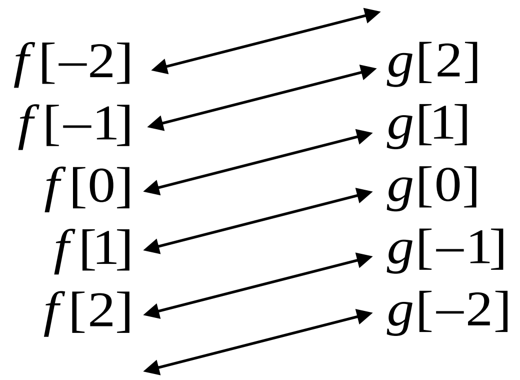
$$\begin{tikzcd}
f[-2] \arrow[r, leftrightarrow] & g[2] \\
f[-1] \arrow[r, leftrightarrow] & g[1] \\
f[0] \arrow[r, leftrightarrow] & g[0] \\
f[1] \arrow[r, leftrightarrow] & g[-1] \\
f[2] \arrow[r, leftrightarrow] & g[-2]
\end{tikzcd}$$

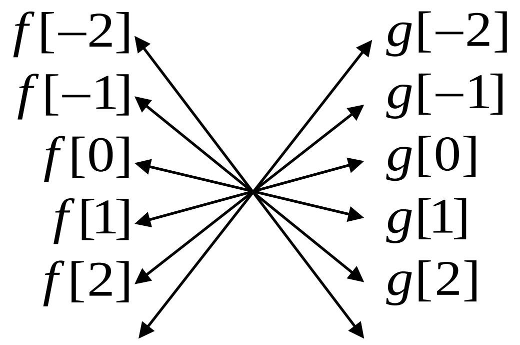
$$\begin{tikzcd}
f[-2] \arrow[ddddr, leftrightarrow] & g[-2] \\
f[-1] \arrow[ddr, leftrightarrow] & g[-1] \\
f[0] \arrow[r, leftrightarrow] & g[0] \\
f[1] \arrow[uur, leftrightarrow] & g[1] \\
f[2] \arrow[uuuur, leftrightarrow] & g[2]
\end{tikzcd}$$

Applications of the FFT

- Moving Dot Products / Cross Correlation
- Convolution

  This is like cross-correlation, except that flip the entries of the array *g*[ ] before cross-correlating.

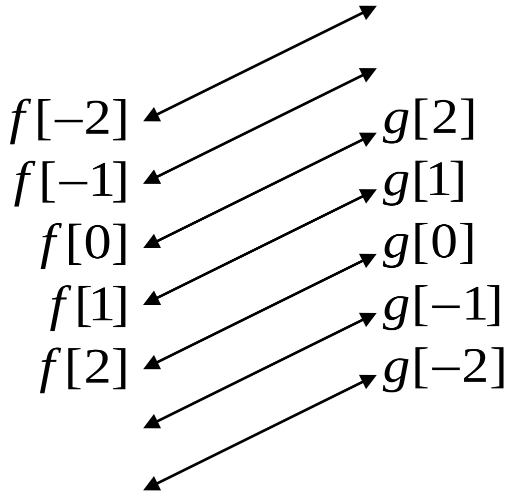


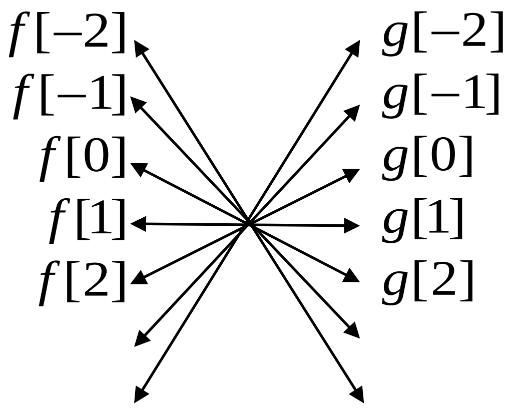


Applications of the FFT

- Moving Dot Products / Cross Correlation
- Convolution

  This is like cross-correlation, except that flip the entries of the array *g*[ ] before cross-correlating.



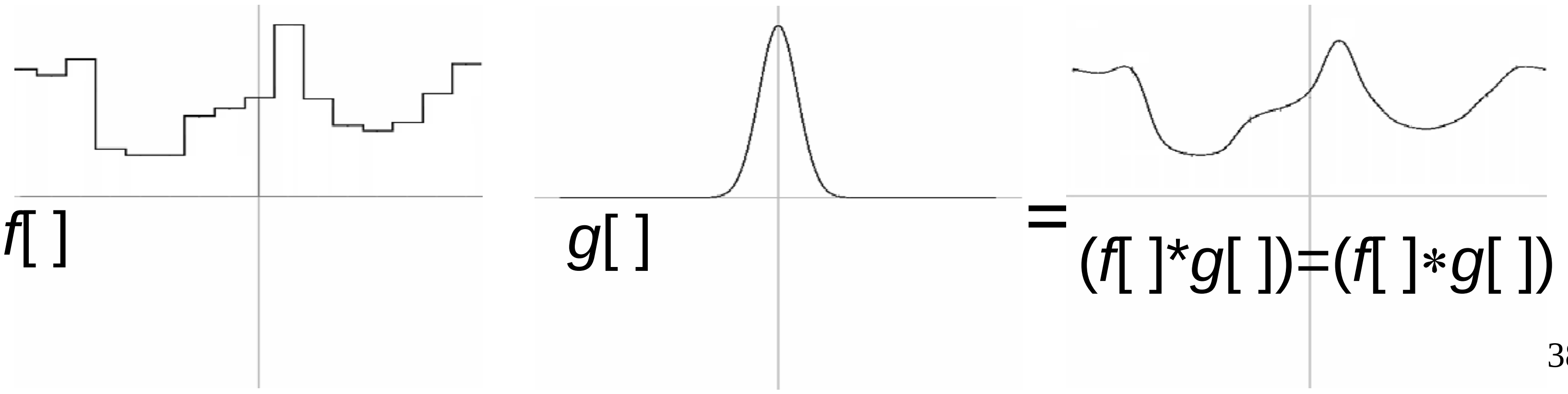


Applications of the FFT

- Moving Dot Products / Cross Correlation
- Convolution

  This is like cross-correlation, except that flip the entries of the array *g*[ ] before cross-correlating.

  Note: If *g*[ ] is symmetric (i.e. *g*[-*k*]=*g*[*k*]) then the convolution of *f*[ ] with *g*[ ] is equal to the cross-correlation of *f*[ ] with *g*[ ].



Applications of the FFT

- Moving Dot Products / Cross Correlation
- Convolution
- Polynomial Multiplication

Applications of the FFT

- Moving Dot Products / Cross Correlation
- Convolution
- Polynomial Multiplication

Given two polynomials:

$$p(x) = a_0 + a_1 x + \cdots + a_n x^n$$
$$q(x) = b_0 + b_1 x + \cdots + b_n x^n$$

we can represent the polynomials *p*(x) and *q*(*x*) by (2*n*+1)-dimensional arrays:

$$p(x) \rightarrow [a_{-n}, \ldots, a_{-1}, a_0, a_1, \ldots, a_n]$$
$$q(x) \rightarrow [b_{-n}, \ldots, b_{-1}, b_0, b_1, \ldots, b_n]$$

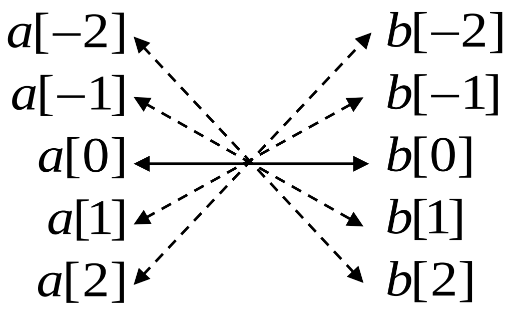
with

$$a_{-n} = \cdots = a_{-1} = b_{-n} = \cdots = b_{-1} = 0$$

Applications of the FFT

- Moving Dot Products / Cross Correlation
- Convolution
- Polynomial Multiplication

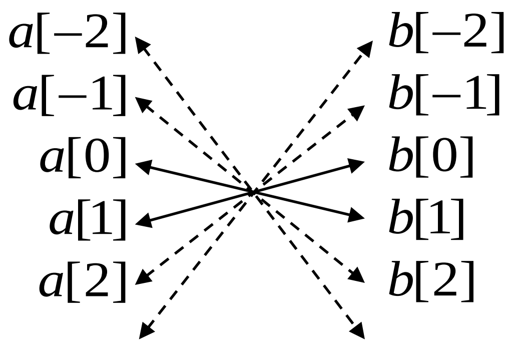
  The 0th order coefficient of the product is:



Applications of the FFT

- Moving Dot Products / Cross Correlation
- Convolution
- Polynomial Multiplication

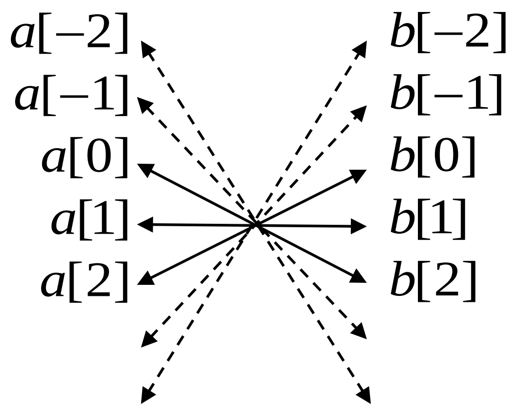
  The 1st order coefficient of the product is:



Applications of the FFT

- Moving Dot Products / Cross Correlation
- Convolution
- Polynomial Multiplication

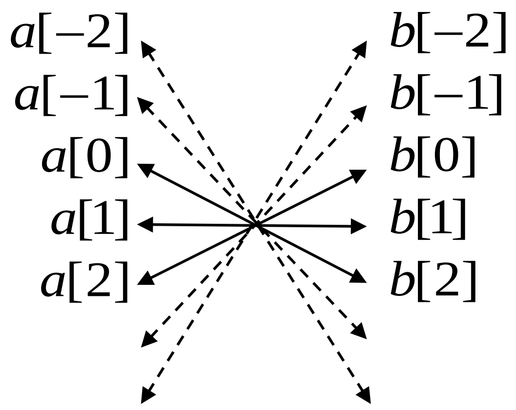
  The 2nd order coefficient of the product is:



Applications of the FFT

- Moving Dot Products / Cross Correlation
- Convolution
- Polynomial Multiplication

  The coefficients of the product can be computed by convolving the arrays corresponding to the coefficients of the original polynomials.



Applications of the FFT

- Moving Dot Products / Cross Correlation
- Convolution
- Polynomial Multiplication
- Big Integer Multiplication

  Given an integer, we can treat it as a polynomial.

  Example:

$$47601345 = 5\cdot10^0 + 4\cdot10^1 + 3\cdot10^2 + 1\cdot10^3 + \cdots$$

Applications of the FFT

- Moving Dot Products / Cross Correlation
- Convolution
- Polynomial Multiplication
- Big Integer Multiplication

  Given an integer, we can treat it as a polynomial.
  To multiply two integers, we need to figure out what the new value in the 1s place,

  Example:

$$\begin{aligned} 47601345 &= 5\cdot 10^0 + 4\cdot 10^1 + 3\cdot 10^2 + 1\cdot 10^3 + \cdots \\ &\;\;\updownarrow \\ 46018729 &= 9\cdot 10^0 + 2\cdot 10^1 + 7\cdot 10^2 + 8\cdot 10^3 + \cdots \end{aligned}$$

Applications of the FFT

- Moving Dot Products / Cross Correlation
- Convolution
- Polynomial Multiplication
- Big Integer Multiplication

  Given an integer, we can treat it as a polynomial.
  To multiply two integers, we need to figure out what the new value in the 1s place, the 10s place,

  Example:

$$47601345 = 5\cdot 10^0 + 4\cdot 10^1 + 3\cdot 10^2 + 1\cdot 10^3 + \cdots$$

$$46018729 = 9\cdot 10^0 + 2\cdot 10^1 + 7\cdot 10^2 + 8\cdot 10^3 + \cdots$$

Applications of the FFT

- Moving Dot Products / Cross Correlation
- Convolution
- Polynomial Multiplication
- Big Integer Multiplication

  Given an integer, we can treat it as a polynomial.
  To multiply two integers, we need to figure out what the new value in the 1s place, the 10s place, the 100s place, etc. will be.

  Example:

$$47601345 = 5\cdot 10^0 + 4\cdot 10^1 + 3\cdot 10^2 + 1\cdot 10^3 + \cdots$$

$$46018729 = 9\cdot 10^0 + 2\cdot 10^1 + 7\cdot 10^2 + 8\cdot 10^3 + \cdots$$

Applications of the FFT

- Moving Dot Products / Cross Correlation
- Convolution
- Polynomial Multiplication
- Big Integer Multiplication

  Given an integer, we can treat it as a polynomial.
  To multiply two integers, we need to figure out what the new value in the 1s place, the 10s place, the 100s place, etc. will be.

  So big integer multiplication can be implemented as a convolution.

Outline

The FFT Algorithm

Applications in 1D

Multi-Dimensional FFTs

More Applications

Real FFTs

Multi-Dimensional FFTs

How do we compute the Fourier transform of a multi-dimensional signal?

Examples:

- Images
- Voxel Grids
- Etc.

2D FFTs

For regularly sampled, *n*x*n* grids, the irreducible representations are spanned by the orthogonal basis $\{v_{lm}[\,][\,]\}$ where:

$$v_{lm}[j][k] = \sqrt{\frac{1}{n^2}} e^{il2\pi j/n} \cdot e^{im2\pi k/n}$$

2D FFTs

For regularly sampled, *nxn* grids, the irreducible representations are spanned by the orthogonal basis $\{v_{lm}[\,][\,]\}$ where:

$$v_{lm}[j][k] = \sqrt{\frac{1}{n^2}} e^{il2\pi j/n} \cdot e^{im2\pi k/n}$$

To see this, consider the action of an index shift by (α,β) on any one of these basis functions:

$$\rho_{(\alpha,\beta)}\left(v_{lm}[j][k]\right) = v_{lm}[j-\alpha][k-\beta]$$

2D FFTs

For regularly sampled, *nxn* grids, the irreducible representations are spanned by the orthogonal basis $\{v_{lm}[\;][\;]\}$ where:

$$v_{lm}[j][k] = \sqrt{\frac{1}{n^2}} e^{il2\pi j/n} \cdot e^{im2\pi k/n}$$

To see this, consider the action of an index shift by (α,β) on any one of these basis functions:

$$\begin{aligned}\rho_{(\alpha,\beta)}\left(v_{lm}[j][k]\right) &= v_{lm}[j-\alpha][k-\beta] \\ &= \sqrt{\frac{1}{n^2}} e^{il2\pi(j-\alpha)/n} \cdot e^{ik2\pi(k-\beta)/n}\end{aligned}$$

2D FFTs

For regularly sampled, *nxn* grids, the irreducible representations are spanned by the orthogonal basis $\{v_{lm}[\ ][\ ]\}$ where:

$$v_{lm}[j][k] = \sqrt{\frac{1}{n^2}} e^{il2\pi j/n} \cdot e^{im2\pi k/n}$$

To see this, consider the action of an index shift by (α,β) on any one of these basis functions:

$$\begin{aligned}\rho_{(\alpha,\beta)}\left(v_{lm}[j][k]\right) &= v_{lm}[j-\alpha][k-\beta] \\ &= \sqrt{\frac{1}{n^2}} e^{il2\pi(j-\alpha)/n} \cdot e^{ik2\pi(k-\beta)/n} \\ &= \left(e^{-il2\pi\alpha/n} \cdot e^{-i2\pi\beta/n}\right) \cdot \sqrt{\frac{1}{n^2}} e^{il2\pi j/n} \cdot e^{im2\pi k/n}\end{aligned}$$

2D FFTs

For regularly sampled, *nxn* grids, the irreducible representations are spanned by the orthogonal basis $\{v_{lm}[\,][\,]\}$ where:

$$v_{lm}[j][k]=\sqrt{\frac{1}{n^2}}e^{il2\pi j/n}\cdot e^{im2\pi k/n}$$

To see this, consider the action of an index shift by (α,β) on any one of these basis functions:

$$\begin{aligned}\rho_{(\alpha,\beta)}\left(v_{lm}[j][k]\right)&=v_{lm}[j-\alpha][k-\beta]\\&=\sqrt{\frac{1}{n^2}}e^{il2\pi(j-\alpha)/n}\cdot e^{ik2\pi(k-\beta)/n}\\&=\left(e^{-il2\pi\alpha/n}\cdot e^{-i2\pi\beta/n}\right)\sqrt{\frac{1}{n^2}}e^{il2\pi j/n}\cdot e^{im2\pi k/n}\\&=\left(e^{-il2\pi\alpha/n}\cdot e^{-i2\pi\beta/n}\right)v_{lm}[j][k]\end{aligned}$$

2D FFTs

How can we compute the 2D Fourier coefficients efficiently?

2D FFTs

How can we compute the 2D Fourier coefficients efficiently?

To do this, we will leverage the fact that the 2D basis vectors can be expressed as the product of 1D basis vectors.

2D FFTs

How can we compute the 2D Fourier coefficients efficiently?

To do this, we will leverage the fact that the 2D basis vectors can be expressed as the product of 1D basis vectors.

In particular, setting $v_l[\,]$ to be the n-dimensional array:

$$v_l[j] = \sqrt{\frac{1}{n}} e^{il2\pi j/n}$$

2D FFTs

How can we compute the 2D Fourier coefficients efficiently?

To do this, we will leverage the fact that the 2D basis vectors can be expressed as the product of 1D basis vectors.

In particular, setting $v_l[\ ]$ to be the n-dimensional array:

$$v_l[j] = \sqrt{\frac{1}{n}} e^{il2\pi j/n}$$

we get:

$$v_{lm}[j][k] = v_l[j] \cdot v_m[k]$$

2D FFTs

To compute the (l,m)-th Fourier coefficient of an (nxn)-dimensional grid $f[\,][\,]$, we compute the dot-product of $f[\,][\,]$ with $v_{lm}[\,][\,]$:

$$\hat{f}[l][m] = \left\langle f[\,][\,], v_{lm}[\,][\,] \right\rangle$$

2D FFTs

To compute the (*l*,*m*)-th Fourier coefficient of an (*n*x*n*)-dimensional grid *f*[ ][ ], we compute the dot-product of *f*[ ][ ] with v_{lm}[ ][ ]:

$$\hat{f}[l][m] = \left\langle f[\,][\,], v_{lm}[\,][\,] \right\rangle$$
$$= \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} f[j][k] \cdot \overline{v_{lm}[j][k]}$$

2D FFTs

To compute the (*l*,*m*)-th Fourier coefficient of an (*n*x*n*)-dimensional grid *f*[ ][ ], we compute the dot-product of *f*[ ][ ] with v_{lm}[ ][ ]:

$$
\begin{aligned}
\hat{f}[l][m] &= \left\langle f[\,][\,], v_{lm}[\,][\,] \right\rangle \\
&= \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} f[j][k] \cdot \overline{v_{lm}[j][k]} \\
&= \sqrt{\frac{1}{n^2}} \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} f[j][k] \cdot e^{-im2\pi k/n} \cdot e^{-il2\pi j/n}
\end{aligned}
$$

2D FFTs

To compute the (l,m)-th Fourier coefficient of an (nxn)-dimensional grid f[ ][ ], we compute the dot-product of f[ ][ ] with v_{lm}[ ][ ]:

$$\begin{aligned}\hat{f}[l][m] &= \left\langle f[\,][\,], v_{lm}[\,][\,] \right\rangle \\ &= \sum_{j=0}^{n-1}\sum_{k=0}^{n-1} f[j][k]\cdot\overline{v_{lm}[j][k]} \\ &= \sqrt{\frac{1}{n^2}}\sum_{j=0}^{n-1}\sum_{k=0}^{n-1} f[j][k]\cdot e^{-im2\pi k/n}\cdot e^{-il2\pi j/n} \\ &= \sqrt{\frac{1}{n^2}}\sum_{j=0}^{n-1}\left(\sum_{k=0}^{n-1} f[j][k]\cdot e^{-im2\pi k/n}\right)\cdot e^{-il2\pi j/n}\end{aligned}$$

2D FFTs

$$\hat{f}[l][m] = \sqrt{\frac{1}{n}} \sum_{j=0}^{n-1} \boxed{\sqrt{\frac{1}{n}} \left(\sum_{k=0}^{n-1} f[j][k] \cdot e^{-im2\pi k/n} \right)} \cdot e^{-il2\pi j/n}$$

The interior summation looks distinctly like a 1D Fourier transform!

2D FFTs

$$\hat{f}[l][m] = \sqrt{\frac{1}{n}} \sum_{j=0}^{n-1} \boxed{\sqrt{\frac{1}{n}} \left(\sum_{k=0}^{n-1} f[j][k] \cdot e^{-im2\pi k/n} \right)} \cdot e^{-il2\pi j/n}$$

The interior summation looks distinctly like a 1D Fourier transform!

In particular, if we set $f_j[\ ]$ to be the 1D array obtained by pulling the j-th row of $f[\ ][\ ]$:

$$f_j[k] = f[j][k]$$

we get:

$$\hat{f}[l][m] = \sqrt{\frac{1}{n}} \sum_{j=0}^{n-1} \sqrt{\frac{1}{n}} \left(\sum_{k=0}^{n-1} f_j[k] \cdot e^{-im2\pi k/n} \right) \cdot e^{-il2\pi j/n}$$

2D FFTs

$$\hat{f}[l][m] = \sqrt{\frac{1}{n}} \sum_{j=0}^{n-1} \boxed{\sqrt{\frac{1}{n}} \left(\sum_{k=0}^{n-1} f[j][k] \cdot e^{-im2\pi k/n} \right)} \cdot e^{-il2\pi j/n}$$

The interior summation looks distinctly like a 1D Fourier transform!

In particular, if we set $f_j[\,]$ to be the 1D array obtained by pulling the j-th row of $f[\,][\,]$:

$$f_j[k] = f[j][k]$$

we get:

$$\hat{f}[l][m] = \sqrt{\frac{1}{n}} \sum_{j=0}^{n-1} \sqrt{\frac{1}{n}} \left(\sum_{k=0}^{n-1} f_j[k] \cdot e^{-im2\pi k/n} \right) \cdot e^{-il2\pi j/n}$$

$$= \sqrt{\frac{1}{n}} \sum_{j=0}^{n-1} \hat{f}_j[m] \cdot e^{-il2\pi j/n}$$

2D FFTs

$$\hat{f}[l][m] = \sqrt{\frac{1}{n}} \sum_{j=0}^{n-1} \hat{f}_j[m] \cdot e^{-il2\pi j/n}$$

But then the exterior summation also looks like a 1D Fourier transform!

2D FFTs

$$\hat{f}[l][m] = \sqrt{\frac{1}{n}} \sum_{j=0}^{n-1} \hat{f}_j[m] \cdot e^{-il2\pi j/n}$$

But then the exterior summation also looks like a 1D Fourier transform!

In particular, we see that the (*l*,*m*)-th Fourier coefficient can be computed by:

1. Computing the 1D Fourier transform of each row independently
2. Computing the 1D Fourier transform of the 1D array consisting of the *m*-th Fourier coefficients of the different rows
3. Pulling the *l*-th Fourier coefficient of the second transform.

2D FFTs

$$\hat{f}[l][m] = \sqrt{\frac{1}{n}} \sum_{j=0}^{n-1} \hat{f}_j[m] \cdot e^{-il2\pi j/n}$$

Thus, to compute all of the Fourier coefficients, we need to:

1. Compute the Fourier coefficients of each row
2. Compute the Fourier coefficients of each column

And the total complexity of this operation is:

$$O(n^2 \log n)$$

Outline

The FFT Algorithm

Applications in 1D

Multi-Dimensional FFTs

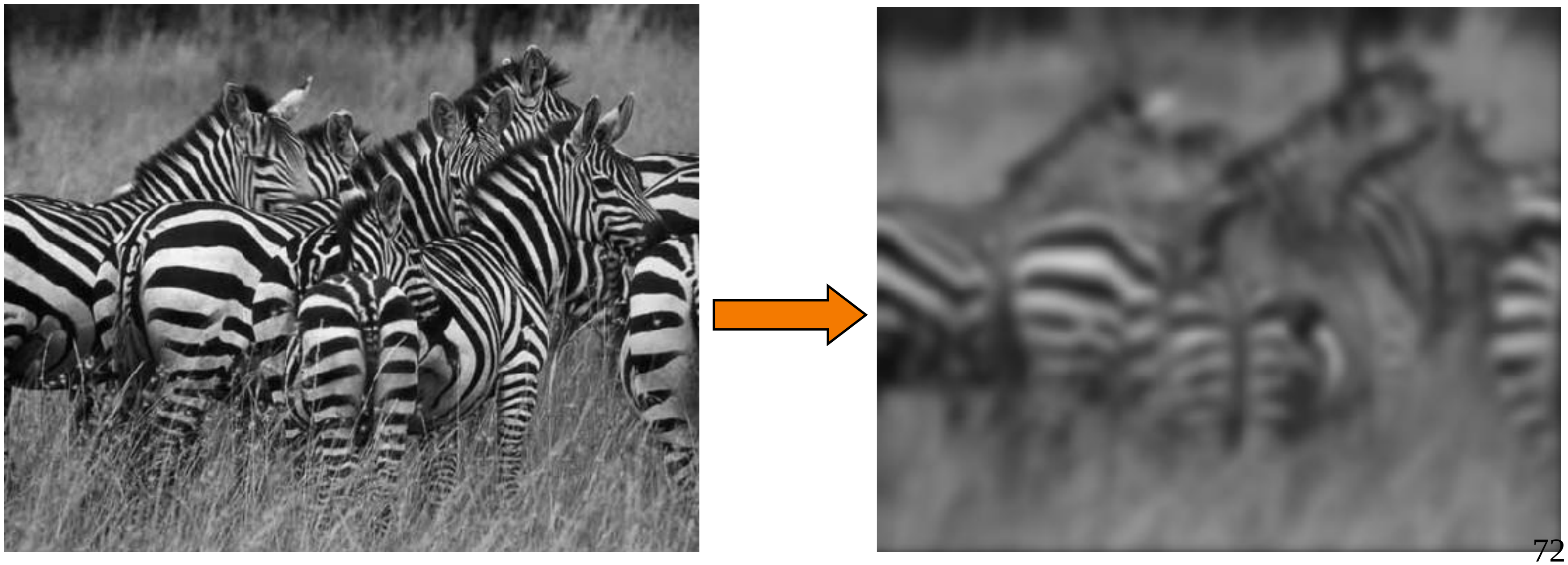
More Applications

- Gaussian Smoothing
- Up-Sampling
- Differentiation
- Boundary Detection
- Gaussian Sharpening

Real FFTs

Gaussian Smoothing

Given an *nxn* grid of values, we would like to smooth the grid.



Gaussian Smoothing

Given an *nxn* grid of values, we would like to smooth the grid.

To do this we need:

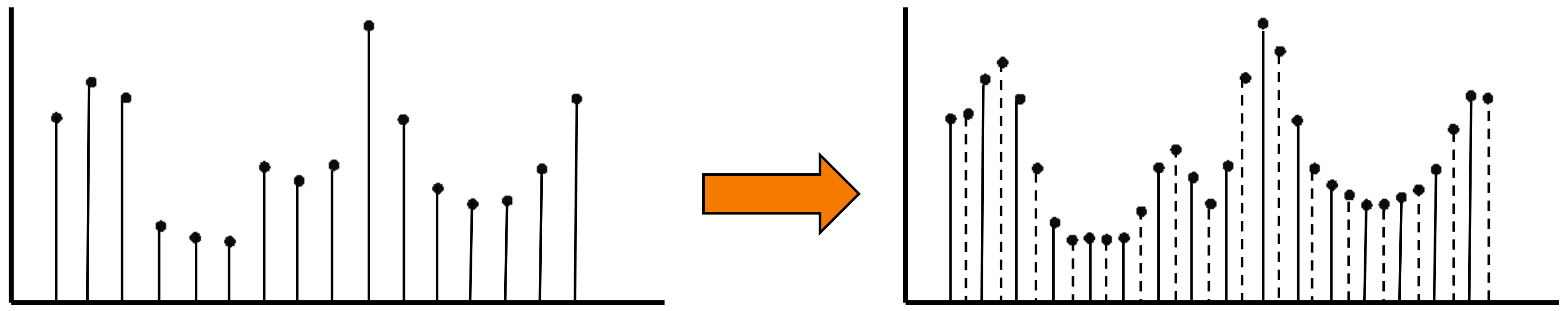
- *f*[ ][ ]: The initial grid
- *g*[ ][ ]: The smoothing filter, usually a Gaussian:

$$g[j][k] = \frac{e^{-\lambda(j^2+k^2)}}{\sum_{j,k} e^{-\lambda(j^2+k^2)}}$$

  with –n/2<j,k≤n/2.
- (*f*[ ][ ]**g*[ ][ ])[ ][ ]: The Gaussian-smoothed grid

Up-Sampling

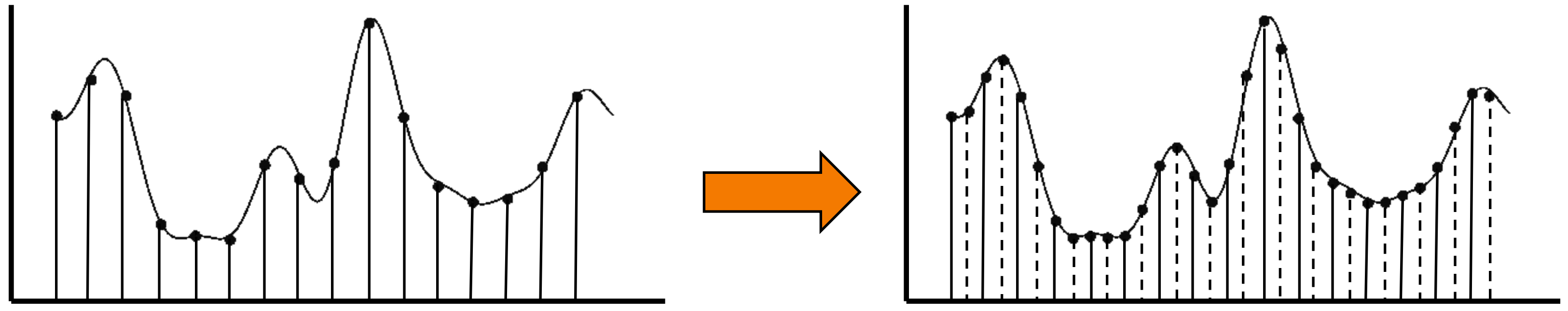
Given an *n*-dimensional array, we would like to extrapolate the array to a 2*n*-dimensional array.



Up-Sampling

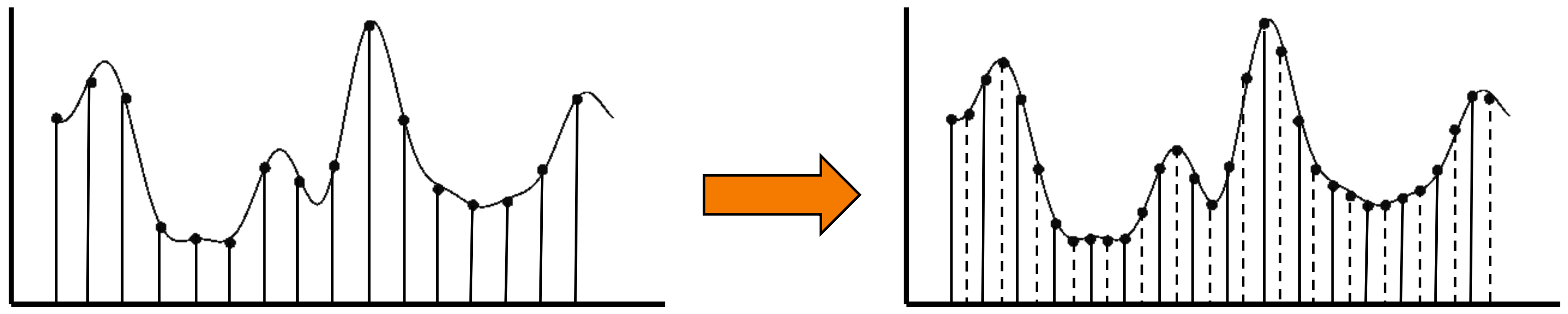
Given an *n*-dimensional array, we would like to extrapolate the array to a 2*n*-dimensional array.

One way to do this is to fit a continuous function to the original array and then sample at 2*n* regular samples.



Up-Sampling

How do we generate a continuous function from a set of *n* samples?



Up-Sampling

How do we generate a continuous function from a set of *n* samples?

Recall that the Fourier decomposition expresses the filter *f*[ ] as:

$$f[j] = \sqrt{\frac{1}{n}} \sum_{k=0}^{n-1} \hat{f}[k] \cdot e^{ik(2\pi j/n)}$$

Up-Sampling

How do we generate a continuous function from a set of *n* samples?

We can fit a continuous function to the data by replacing the discrete index $0 \leq j < n$ with a continuous index $0 \leq s < n$:

$$f[j] = \sqrt{\frac{1}{n}} \sum_{k=0}^{n-1} \hat{f}[k] \cdot e^{ik(2\pi j/n)} \quad \Longleftrightarrow \quad f(s) = \sqrt{\frac{1}{n}} \sum_{k=0}^{n-1} \hat{f}[k] \cdot e^{ik(2\pi s/n)}$$

Up-Sampling

How do we generate a continuous function from a set of *n* samples?

We can fit a continuous function to the data by replacing the discrete index $0 \le j < n$ with a continuous index $0 \le s < n$:

$$f[j] = \sqrt{\frac{1}{n}} \sum_{k=0}^{n-1} \hat{f}[k] \cdot e^{ik(2\pi j/n)} \quad \Longleftrightarrow \quad f(s) = \sqrt{\frac{1}{n}} \sum_{k=0}^{n-1} \hat{f}[k] \cdot e^{ik(2\pi s/n)}$$

Since at integer values *j*, we have:

$$f[j] = f(j)$$

we know that the continuous function interpolates the *n* discrete samples.

Up-Sampling

Word of Warning:

Recall that for integer values of *j*, the complex exponential satisfies the condition:

$$e^{ik(2\pi j/n)} = e^{i(k+n)(2\pi j/n)}$$

Up-Sampling

Word of Warning:

Recall that for integer values of *j*, the complex exponential satisfies the condition:

$$e^{ik(2\pi j/n)} = e^{i(k+n)(2\pi j/n)}$$

Thus, if we were to fit a function:

$$f[j] = \sqrt{\frac{1}{n}} \sum_{k=0}^{n-1} \hat{f}[k] \cdot e^{ik(2\pi j/n)} \quad \Longleftrightarrow \quad f(s) = \sqrt{\frac{1}{n}} \sum_{k=0}^{n-1} \hat{f}[k] \cdot e^{i(k+n)(2\pi s/n)}$$

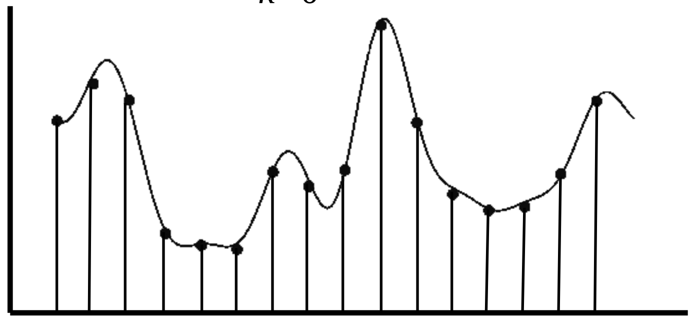
we would also get a continuous function that interpolates the *n* discrete samples.

Up-Sampling

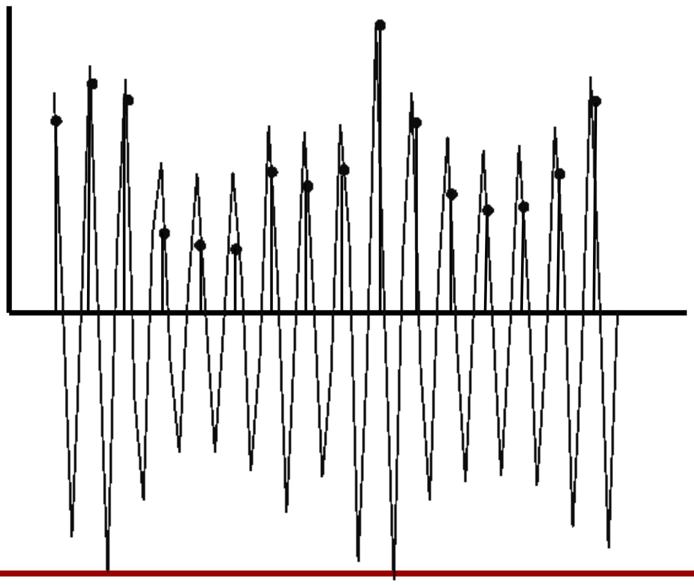
Word of Warning:

The difference is in how the array is interpolated:

$$f(s)=\sqrt{\frac{1}{n}}\sum_{k=0}^{n-1}\hat{f}[k]\cdot e^{ik(2\pi s/n)}$$



$$f(s)=\sqrt{\frac{1}{n}}\sum_{k=0}^{n-1}\hat{f}[k]\cdot e^{i(k+n)(2\pi s/n)}$$

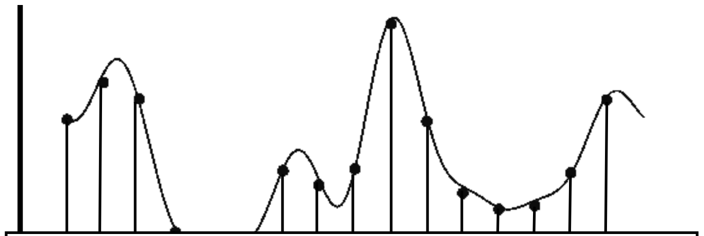


Up-Sampling

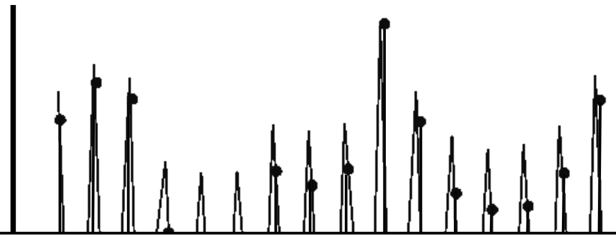
Word of Warning:

Extrapolating the discrete samples is an under-constrained problem, and so there are many different solutions.

In practice, we would like the smoothest possible fit, so we would like to minimize the contribution of high frequency terms



$$f(s)=\sqrt{\frac{1}{n}}\sum_{k=0}^{n-1}\hat{f}[k]\cdot e^{ik(2\pi s/n)}$$



$$f(s)=\sqrt{\frac{1}{n}}\sum_{k=0}^{n-1}\hat{f}[k]\cdot e^{i(k+n)(2\pi s/n)}$$

Up-Sampling

Word of Warning:

Thus, the "best" fit is obtained using the function:

$$\boxed{f(s) = \sqrt{\frac{1}{n}} \sum_{k=-n/2+1}^{n/2} \hat{f}[k] \cdot e^{ik(2\pi s/n)}}$$

Up-Sampling

Word of Warning:

Thus, the "best" fit is obtained using the function:

$$\boxed{f(s) = \sqrt{\frac{1}{n}} \sum_{k=-n/2+1}^{n/2} \hat{f}[k] \cdot e^{ik(2\pi s/n)}}$$

A simple algorithm for implementing this is:

1. Compute the *n* Fourier coefficients of *f*[ ]
2. Generate an array of 2*n* Fourier coefficients:
$$\hat{g}[k] = \begin{cases} \hat{f}[k] & -n/2 < k \leq n/2 \\ 0 & \text{otherwise} \end{cases}$$
3. Compute the inverse Fourier transform to get back the 2*n*-dimensional array *g*[ ]

Differentiation

Given an *n*-dimensional array *f*[ ], how do we compute the derivative of *f*[ ]?

Differentiation

Given an *n*-dimensional array *f*[ ], how do we compute the derivative of *f*[ ]?

Finite Differences:

Define the derivative at some index *j* as the average of the discrete left and right derivatives:

$$f'[j] = \frac{\left(f[j+1] - f[j]\right) + \left(f[j] - f[j-1]\right)}{2}$$

Differentiation

Given an *n*-dimensional array *f*[ ], how do we compute the derivative of *f*[ ]?

Finite Differences:

Define the derivative at some index *j* as the average of the discrete left and right derivatives:

$$
\begin{aligned}
f'[j] &= \frac{\left(f[j+1]-f[j]\right)+\left(f[j]-f[j-1]\right)}{2} \\
&= \frac{f[j+1]-f[j-1]}{2}
\end{aligned}
$$

Differentiation

Given an *n*-dimensional array *f*[ ], how do we compute the derivative of *f*[ ]?

Finite Differences:

Define the derivative at some index *j* as the average of the discrete left and right derivatives:

$$f'[j] = \frac{f[j+1] - f[j-1]}{2}$$

Continuous Differentiation:

Fit a continuous function to the samples and take the derivative of the continuous function.

Differentiation

Continuous Differentiation:

Fit a continuous function to the samples and take the derivative of the continuous function.

If we a continuous function to *f*[ ] by:

$$f(s)=\sqrt{\frac{1}{n}}\sum_{k=-n/2+1}^{n/2}\hat{f}[k]\cdot e^{ik(2\pi s/n)}$$

Differentiation

Continuous Differentiation:

Fit a continuous function to the samples and take the derivative of the continuous function.

If we a continuous function to *f*[ ] by:

$$f(s)=\sqrt{\frac{1}{n}}\sum_{k=-n/2+1}^{n/2}\hat{f}[k]\cdot e^{ik(2\pi s/n)}$$

Then using the fact that:

$$\frac{\partial}{\partial s}e^{\lambda s}=\lambda e^{\lambda s}$$

Differentiation

Continuous Differentiation:

Fit a continuous function to the samples and take the derivative of the continuous function.

If we a continuous function to *f*[ ] by:

$$f(s)=\sqrt{\frac{1}{n}}\sum_{k=-n/2+1}^{n/2}\hat{f}[k]\cdot e^{ik(2\pi s/n)}$$

Then using the fact that:

$$\frac{\partial}{\partial s}e^{\lambda s}=\lambda e^{\lambda s}$$

We get:

$$f'(s)=\sqrt{\frac{1}{n}}\sum_{k=-n/2+1}^{n/2}\hat{f}[k](ik2\pi/n)\cdot e^{ik(2\pi s/n)}$$

Differentiation

Continuous Differentiation:

$$f'(s) = \sqrt{\frac{1}{n}} \sum_{k=-n/2+1}^{n/2} \hat{f}[k](ik2\pi/n) \cdot e^{ik(2\pi s/n)}$$

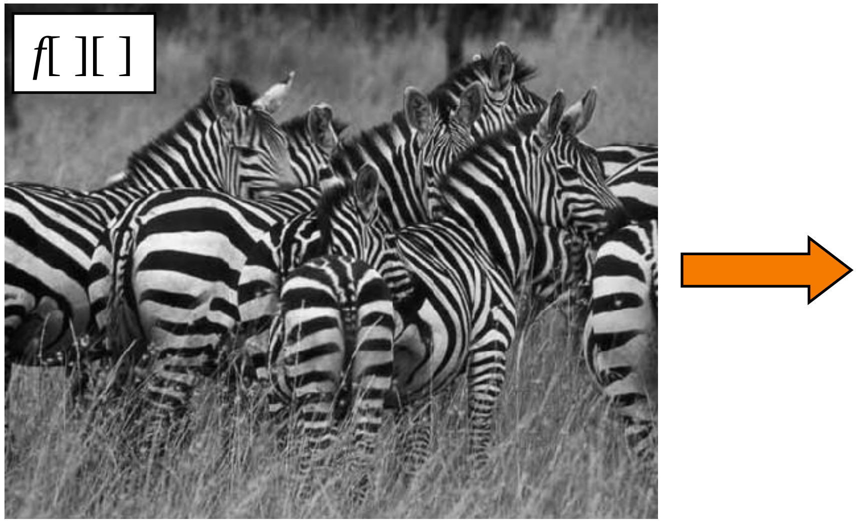
Thus, we can obtain the values of the continuous derivative by multiplying the *k*-th Fourier coefficient of *f*[ ] by (*ik*2π/*n*):

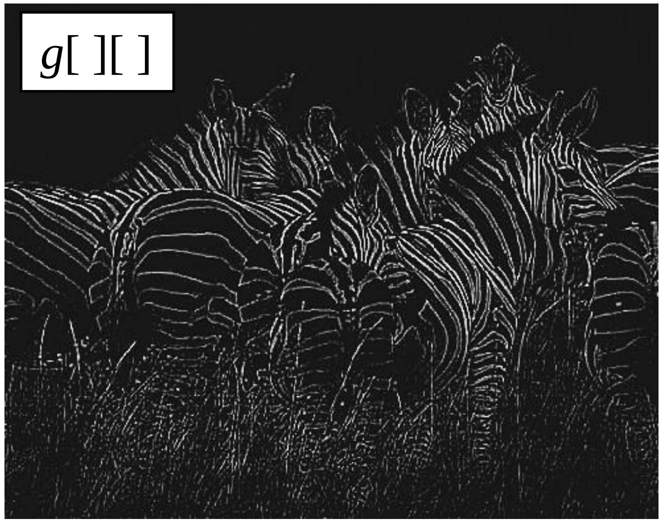
$$\hat{f}'[k] = (ik2\pi/n)\hat{f}[k]$$

Boundary Detection

To compute the boundary of a grid *f*[ ][ ], we would like to measure how much the grid *f*[ ][ ] is changing at every index.

Specifically, we would like to define a grid *g*[ ][ ] such that *g*[*j*][*k*] measure the “rate of change” of *f*[ ][ ] at the index (*j*,*k*).





Boundary Detection

Gradient Method:

One way to measure the rate of change is by computing the gradient of *f*[ ][ ] at every point and setting:

$$g[j][k] = \left\| \nabla f[j][k] \right\|$$

Boundary Detection

Gradient Method:

One way to measure the rate of change is by computing the gradient of *f*[ ][ ] at every point and setting:

$$g[j][k] = \left\| \nabla f[j][k] \right\|$$

To compute the gradient, we need to compute the partial derivatives of *f*[ ][ ] at every point.

Boundary Detection

<u>Gradient Method</u>:

This can be done with the FFT:

1. Compute the Fourier coefficients of *f*[ ][ ]
2. Generate the two grids corresponding to the Fourier coefficients of the partial derivatives:
$$\hat{f}_x[j][k] = \hat{f}[j][k] \cdot (ij2\pi/n)$$
$$\hat{f}_y[j][k] = \hat{f}[j][k] \cdot (ik2\pi/n)$$
3. Compute the inverse Fourier transforms to get the grids of partial derivatives f_x[ ][ ] and f_y[ ][ ].
4. Set *g*[ ][ ] to be the grid of gradient lengths:
$$g[j][k] = \sqrt{f_x[j][k]^2 + f_y[j][k]^2}$$

Boundary Detection

Laplacian Method:

An alternate way to measure the rate of change is to compute the difference between the original grid and a smoothed version of the grid.

Boundary Detection

Laplacian Method:

An alternate way to measure the rate of change is to compute the difference between the original grid and a smoothed version of the grid.

A measure of this difference can be obtained by computing the Laplacian (the sum of unmixed) partial derivatives:

$$\Delta f = f_{xx} + f_{yy}$$

Boundary Detection

Laplacian Method:

This can be done with the FFT:

1. Compute the Fourier coefficients of *f*[ ][ ]
2. Generate the grid corresponding to the Fourier coefficients of the Laplacian:

$$\left(\hat{f}_{xx} + \hat{f}_{yy}\right)[j][k] = \hat{f}[j][k] \cdot \left((ij2\pi/n)^2 + (ik2\pi/n)^2\right)$$
$$= -\hat{f}[j][k] \cdot \left((j^2 + k^2)4\pi^2/n^2\right)$$

3. Set *g*[ ][ ] to be the inverse Fourier transform of the Laplacian Fourier coefficients.

Gaussian Sharpening

How do we undo the effects of Gaussian-smoothing a grid?

Gaussian Sharpening

How do we undo the effects of Gaussian-smoothing a grid?

We compute the Gaussian smoothing of *f*[ ][ ] by:

1. Computing the Fourier transforms of *f*[ ][ ] and the Gaussian grid *g*[ ][ ]
2. Multiplying the Fourier coefficients of *f*[ ][ ] by the the Fourier coefficients of *g*[ ][ ]
3. Computing the inverse Fourier transform

Gaussian Sharpening

How do we undo the effects of Gaussian-smoothing a grid?

To sharpen an image, we have to undo the convolution. This can be done by:

1. Computing the Fourier transforms of *f*[ ][ ] and *g*[ ][ ]
2. Multiplying the Fourier coefficients of *f*[ ][ ] by the reciprocals of the Fourier coefficients of *g*[ ][ ]
3. Computing the inverse Fourier transform

Gaussian Sharpening

How do we undo the effects of Gaussian-smoothing a grid?

To sharpen an image, we have to undo the convolution. This can be done by:

1. Computing the Fourier transforms of *f*[ ][ ] and *g*[ ][ ]
2. Multiplying the Fourier coefficients of *f*[ ][ ] by the reciprocals of the Fourier coefficients of *g*[ ][ ]
3. Computing the inverse Fourier transform

As long as the Fourier coefficients of *g*[ ][ ] are non-zero, this process is well-defined.

Outline

Real FFTs

So far, we have considered the Fourier transform of complex valued functions. What happens when the values of the function are all real?

$$\boxed{f(\theta) - \overline{f(\theta)} = 0}$$

Real FFTs

If we write out the function *f* in terms of its Fourier decomposition, we get:

$$f(\theta) = \sum_{k=-\infty}^{\infty} \hat{f}(k) \cdot \sqrt{\frac{1}{2\pi}} e^{ik\theta}$$

Real FFTs

If we write out the function *f* in terms of its Fourier decomposition, we get:

$$f(\theta) = \sum_{k=-\infty}^{\infty} \hat{f}(k) \cdot \sqrt{\frac{1}{2\pi}} e^{ik\theta}$$

Using the fact that $f(\theta)-\overline{f(\theta)}=0$ we get:

$$0 = \sum_{k=-\infty}^{\infty} \hat{f}(k) \cdot \sqrt{\frac{1}{2\pi}} e^{ik\theta} - \sum_{k=-\infty}^{\infty} \overline{\hat{f}(k)} \cdot \sqrt{\frac{1}{2\pi}} e^{-ik\theta}$$

Real FFTs

If we write out the function *f* in terms of its Fourier decomposition, we get:

$$f(\theta) = \sum_{k=-\infty}^{\infty} \hat{f}(k) \cdot \sqrt{\frac{1}{2\pi}} e^{ik\theta}$$

Using the fact that $f(\theta)-\overline{f(\theta)}=0$ we get:

$$0 = \sum_{k=-\infty}^{\infty} \hat{f}(k) \cdot \sqrt{\frac{1}{2\pi}} e^{ik\theta} - \sum_{k=-\infty}^{\infty} \overline{\hat{f}(k)} \cdot \sqrt{\frac{1}{2\pi}} e^{-ik\theta}$$

But this can be re-written as:

$$0 = \sum_{k=-\infty}^{\infty} \hat{f}(k) \cdot \sqrt{\frac{1}{2\pi}} e^{ik\theta} - \sum_{k=-\infty}^{\infty} \overline{\hat{f}(-k)} \cdot \sqrt{\frac{1}{2\pi}} e^{ik\theta}$$

Real FFTs

$$\boxed{0=\sum_{k=-\infty}^{\infty}\hat{f}(k)\cdot\sqrt{\frac{1}{2\pi}}e^{ik\theta}-\sum_{k=-\infty}^{\infty}\overline{\hat{f}(-k)}\cdot\sqrt{\frac{1}{2\pi}}e^{ik\theta}}$$

Simplifying this equation we get:

$$0=\sum_{k=-\infty}^{\infty}\left(\hat{f}(k)-\overline{\hat{f}(-k)}\right)\cdot\sqrt{\frac{1}{2\pi}}e^{ik\theta}$$

Real FFTs

$$\boxed{0=\sum_{k=-\infty}^{\infty}\hat{f}(k)\cdot\sqrt{\frac{1}{2\pi}}e^{ik\theta}-\sum_{k=-\infty}^{\infty}\overline{\hat{f}(-k)}\cdot\sqrt{\frac{1}{2\pi}}e^{ik\theta}}$$

Simplifying this equation we get:

$$0=\sum_{k=-\infty}^{\infty}\left(\hat{f}(k)-\overline{\hat{f}(-k)}\right)\cdot\sqrt{\frac{1}{2\pi}}e^{ik\theta}$$

But since the function on the left is equal to zero, this must imply that all the Fourier coefficients are equal to zero:

$$0=\hat{f}(k)-\overline{\hat{f}(-k)}$$

$$\Updownarrow$$

$$\hat{f}(k)=\overline{\hat{f}(-k)}$$

Real FFTs

$$\boxed{\hat{f}(k) = \overline{\hat{f}(-k)}}$$

Thus when the function *f* is real, the Fourier coefficients have the property that the *k*-th Fourier coefficient is the complex conjugate of the (-*k*)-th Fourier coefficient.

Real FFTs

Although this discussion holds true for real functions, a similar argument shows that for real-valued, *n*-dimensional arrays *f*[ ], the Fourier coefficients have the property that:

$$\boxed{\hat{f}[k] = \overline{\hat{f}[n-k]}}$$

Real FFTs

For real-valued arrays:

- In 1D we have:

$$\hat{f}[j] = \overline{\hat{f}[n-j]}$$

- In 2D we have:

$$\hat{f}[j][k] = \overline{\hat{f}[n-j][n-k]}$$

- In 3D we have:

$$\hat{f}[j][k][l] = \overline{\hat{f}[n-j][n-k][n-l]}$$

So when the array is real-valued, we only have to compute and store half of the coefficients.