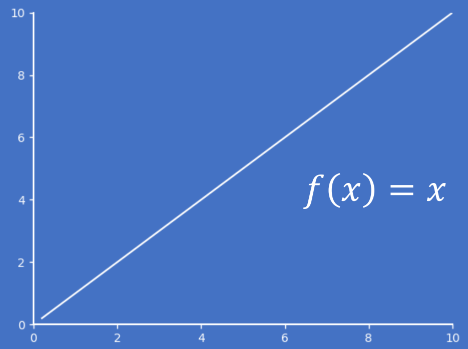
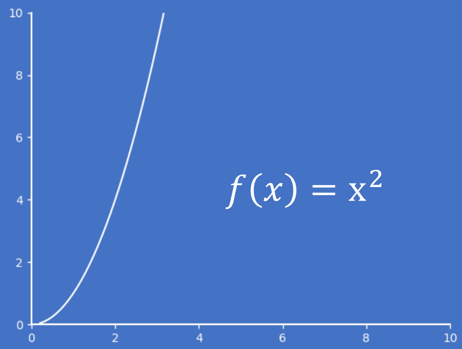
Figure 1



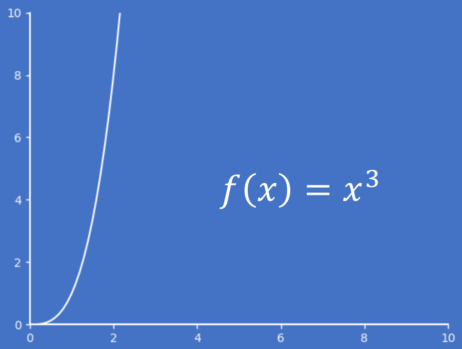
$\cdot$

Figure 2



$=$

Figure 3



POLYNOMIAL MULTIPLICATION

USING THE FAST FOURIER TRANSFORM

Thomas Helbrecht

June 12, 2020

Motivation

- Polynomials model real world objects on paper
- Faster multiplication leads to improvement in wide variety of fields, such as
    - Image processing
    - Compression
    - Cryptography

Dramatic performance improvement with Fast Fourier Transform(FFT)

- Traditional approach of multiplying two polynomials: $O(n^2)$
- Polynomial multiplication using FFT: $O(n \cdot log(n))$

Figure 4

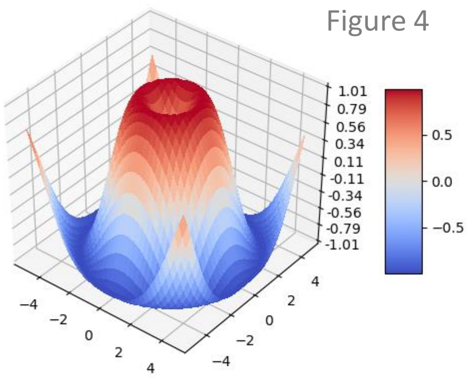
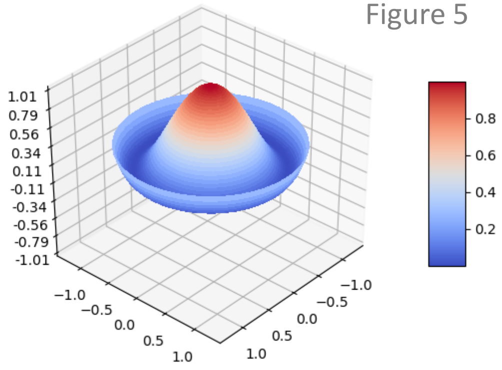


Figure 5



Preliminaries

- Polynomials usually appear in form $\mathrm{f(x)} = \sum_{i=0}^{n-1} a_i \cdot x^i$ and have $\mathrm{degree(f)} = n - 1$

Representation	Definition
Coefficient matrices	$(\mathrm{a_0, a_1, \dots, a_{n-1}})$
Roots	$\{x \mid f(x) = 0\}$
Values at random points	$\{f(x_k)\} \mid x_k \in \{x_k \mid x_k \ chosen\ arbitrarily\ \}\}$
Values at n^{th} roots of unity	$\{f(\omega_n^j)$ | with $\omega_n = e^{\frac{2\pi i}{n}}$ $for\ all\ j \in \{0,1,\dots,n-1\}\}$

Complex n^th roots of unity:
$\omega_n^0, \omega_n^1, \dots, \omega_n^{n-1}$

Halving Lemma: If $n > 0$ is even, then the squares of the n complex n^{th} roots of unity are the $\frac{n}{2}$ complex $\frac{n^{\text{th}}}{2}$ roots of unity.

Euler's formula: $e^{iu} = \cos(u) + i \cdot \sin(u)$

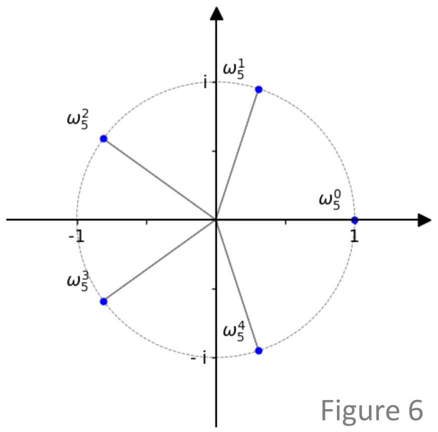


Figure 6

Example:
Let $f(x) = x + 1$.
1. $(1,1)$
2. $\{-1\}$
3. $\{1,2,3\}$ at $x = 0, 1, 2$
4. $\{2, 0\}$ at $x = \omega_2^0, \omega_2^1$

Fast Fourier Transform(FFT)

Discrete Fourier Transform(DFT)

- **Input:** $A(x) = \sum_{j=0}^{n-1} a_j x^j$ in coefficient form as a vector $(a_0, a_1, \dots, a_{\{n-1\}})$
- **Result:** Vector $(y_0, y_1, \dots, y_{n-1})$ with $y_k = A(\omega_n^k) = \sum_{j=0}^{n-1} a_j \omega_n^{k\cdot j}$
- Calculation takes $O(n^2)$ time(n times evaluating in $O(n)$ time)

Horner's rule:
$A(x) = a_0 + x(a_1 + \dots + x(a_{n-1} + xa_n)\dots)$

Fast Fourier Transform(FFT)

- Divide and conquer approach to computing the Discrete Fourier transform
- Let $n = 2^r$ for some n. We get:

$$A(\omega_n^k) = \sum_{j=0}^{n-1} a_j \cdot \omega_n^{kj} = \sum_{j=0}^{2^r-1} a_j \cdot e^{\frac{2\pi ikj}{2^r}} = \sum_{m=0}^{2^{r-1}-1} a_{2m} \cdot e^{\frac{2\pi ikm}{2^{r-1}}} + \sum_{m=0}^{2^{r-1}-1} a_{2m+1} \cdot e^{\frac{2\pi ik(2m+1)}{2^r}}$$

$$= \sum_{m=0}^{2^{r-1}-1} a_{2m} \cdot e^{\frac{2\pi ikm}{2^{r-1}}} + e^{\frac{2\pi ik}{2^r}} \cdot \sum_{m=0}^{2^{r-1}-1} a_{2m+1} \cdot e^{\frac{2\pi ikm}{2^{r-1}}}$$

$$A(\omega_n^k) = A^{[0]}(\omega_n^{2k}) + \omega_n^k \cdot A^{[1]}(\omega_n^{2k})$$

With:

$$A^{[0]}(x) = a_0 + a_2 \cdot x + \dots + a_{n-2} x^{\frac{n}{2}-1}$$

$$A^{[1]}(x) = a_1 + a_3 \cdot x + \dots + a_{n-1} x^{\frac{n}{2}-1}$$

Fast Fourier Transform(FFT)

- Calculating the FFTs of the vectors with entries of even indexes and odd indexes makes it possible to calculate the FFT of the original vector

Pseudocode

```
function FFT (n:integer; x :complexarray):complexarray;
    if n = 1 then
        FFT [0] := x[0]
    else
        evenarray := {x[0], x[2],...,x[n - 2]};
        oddarray := {x[1], x[3],...,x[n - 1]};

        {u[0], u[1],...u[ n/2 - 1]} := FFT (n/2, evenarray);
        {v[0], v[1],...v[ n/2 - 1]} := FFT (n/2, oddarray);

        for j := 0 to n - 1 do
            τ := exp{2πij/n};
            FFT [j] := u[j mod n/2 ] + τv[j mod n/2 ]
end
```

Pseudocode adapted from Wilf, H. (1994). *Algorithms and Complexity.* Internet Edition. https://www.math.upenn.edu/~wilf/AlgoComp.pdf

Complexity:

$$\mathrm{T(n)} = 2 \cdot \mathrm{T}\left(\frac{\mathrm{n}}{2}\right) + \mathrm{O(n)}$$

Applying the master theorem yields:

$O(n \cdot \log(n))$

Polynomial multiplication using FFT

- **Goal:** Computing $fg(x) = f(x) \cdot g(x)\ \forall x$
- **Input:** Two polynomials p, q in coefficient representation with degree m, n respectively
- **Output:** Coefficient representation of product fg

degree(fg)
$= \text{degree(f)} + \text{degree(g)}$

Algorithm

1. Double the degree of input to smallest power of two that is greater than $m + n + 1$
2. Extend input vectors with zero's
3. Compute the ***FFT*** of both vectors
4. Pointwise multiply $f\left(\omega_n^k\right) \cdot g(\omega_n^k)$ for each $k \in \{0,1, \dots, n-1\}$ to obtain ***FFT*** vector of fg
5. Use ***IFFT*** to convert from ***FFT*** of the product back to the coefficient representation of fg

Complexity: $3 \cdot O\big(n \cdot log(n)\big) + O(n) = O(n \cdot \log(n))$

Inverse Fourier Transform(IFFT)

Obtained by applying simple modifications to the FFT algorithm

From $A\left(\omega_n^k\right) = \sum_{j=0}^{n-1} a_j \cdot e^{\frac{2\pi ijk}{n}}$ we get $a_j = \frac{1}{n} \cdot \sum_{j=0}^{n-1} A\left(\omega_n^k\right) \cdot e^{\frac{-2\pi ijk}{n}}$

THANK YOU FOR YOUR ATTENTION

References

- Cormen, T. H., Leiderson, C. E., Rivest, R. L. & Stein C.(2013). *Introduction to Algorithms. Third edition.* The MIT Press

- Wilf, H. (1994). *Algorithms and Complexity.* Internet Edition. https://www.math.upenn.edu/~wilf/AlgoComp.pdf

- Figures 1-6 can be found in my GitHub repository at https://github.com/thelbrecht/ac-fft