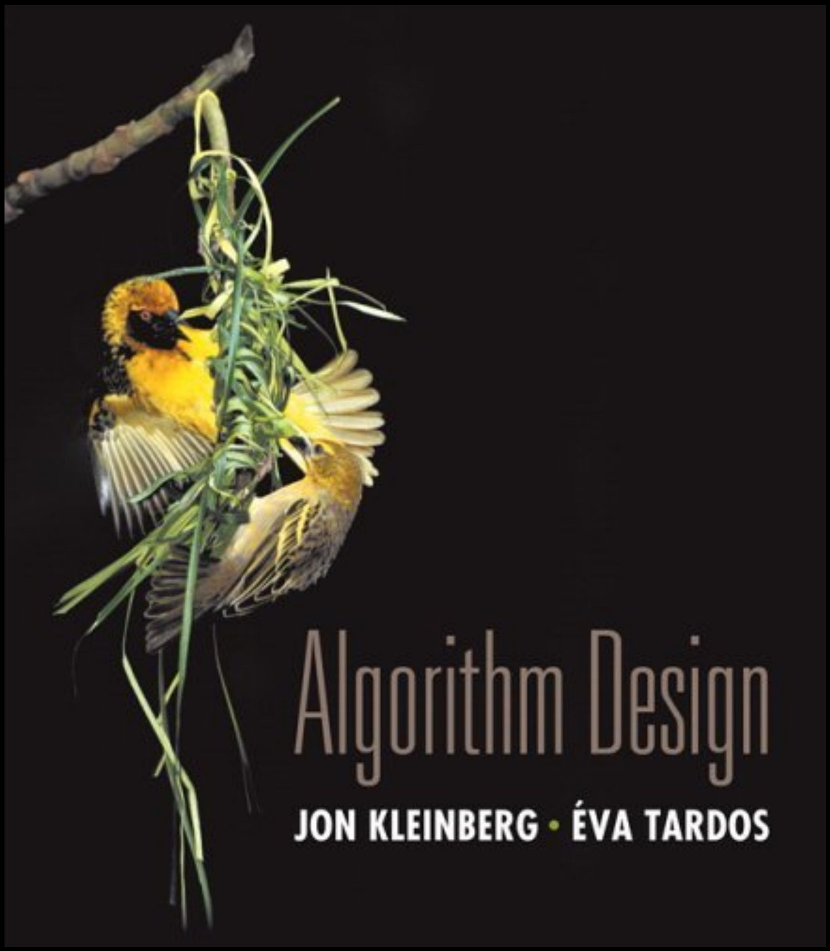


Chapter 5

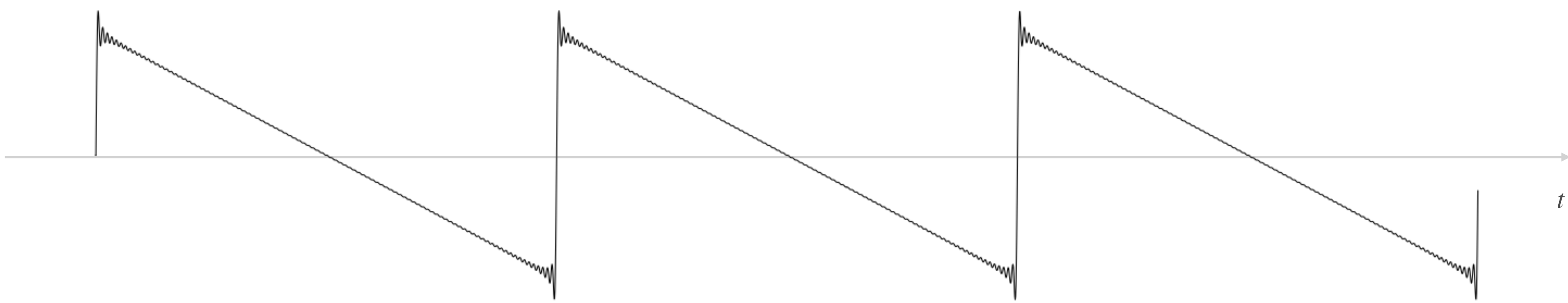
Divide and Conquer

Slides by Kevin Wayne.

Fourier Analysis

Fourier Analysis

Fourier theorem. [Fourier, Dirichlet, Riemann] Any (sufficiently smooth) periodic function can be expressed as the sum of a series of sinusoids.

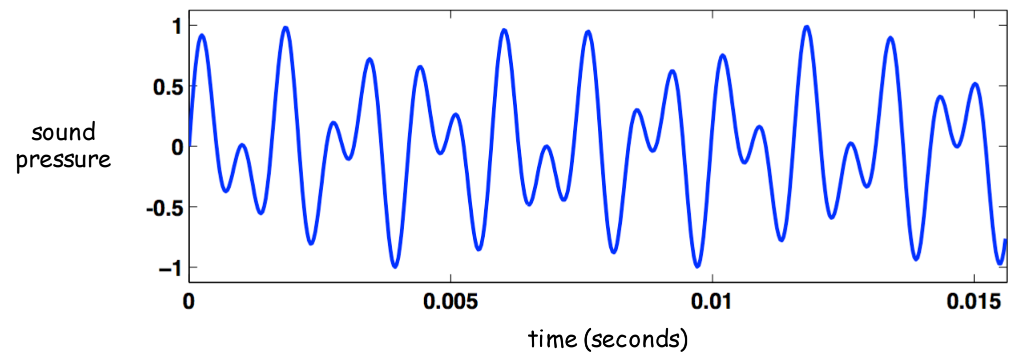


$$y(t) = \frac{2}{\pi} \sum_{k=1}^{N} \frac{\sin kt}{k} \qquad N = 100$$

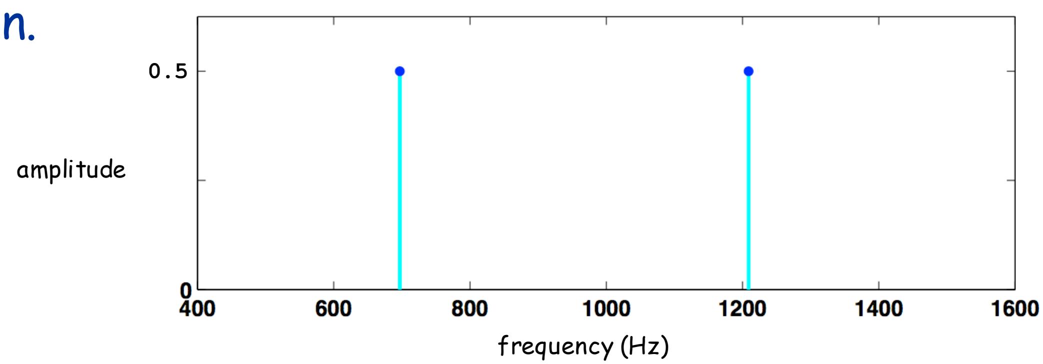
Time Domain vs. Frequency Domain

Signal. [touch tone button 1] $y(t) = \frac{1}{2}\sin(2\pi \cdot 697\ t) + \frac{1}{2}\sin(2\pi \cdot 1209\ t)$

Time domain.

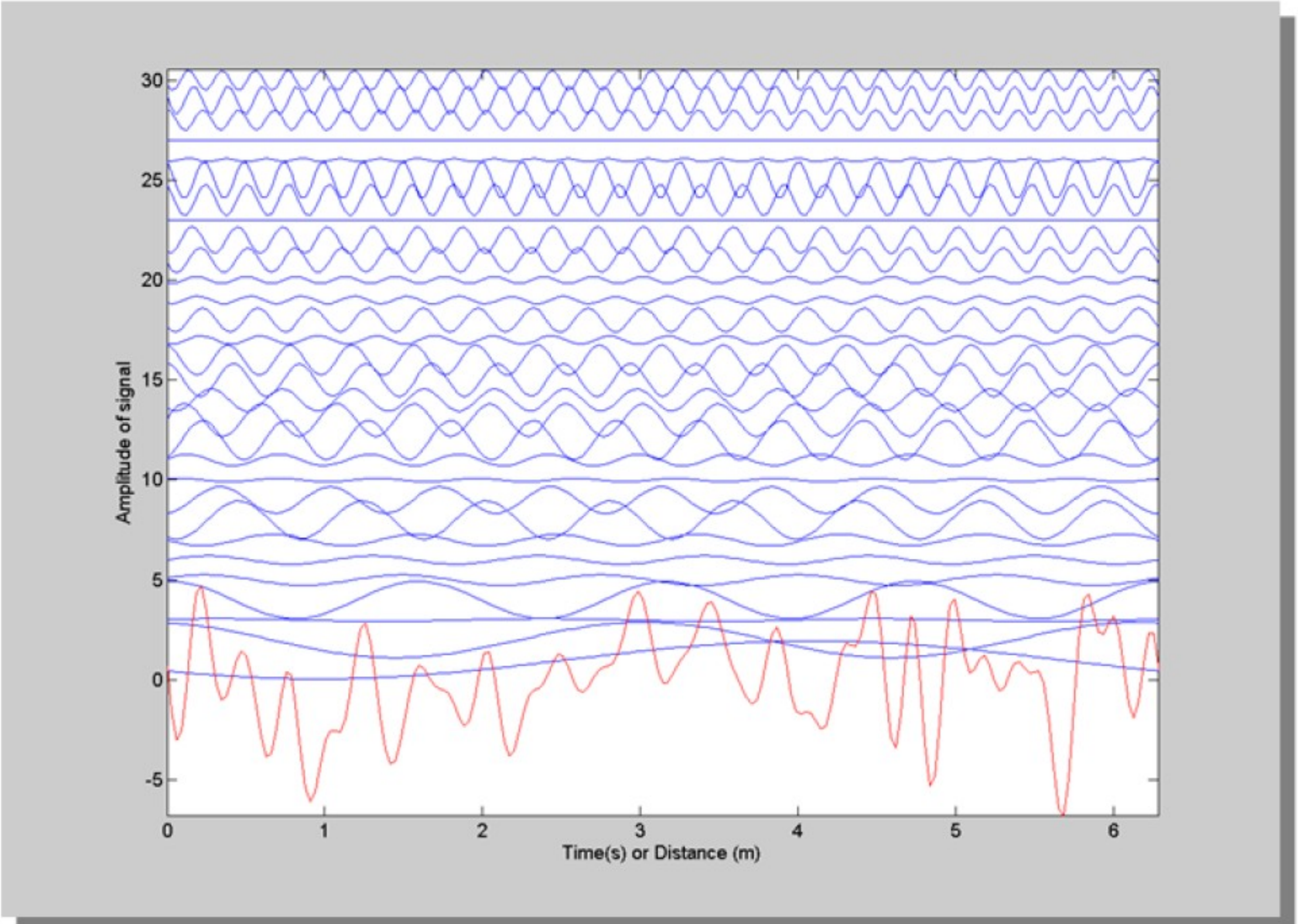


Frequency domain.

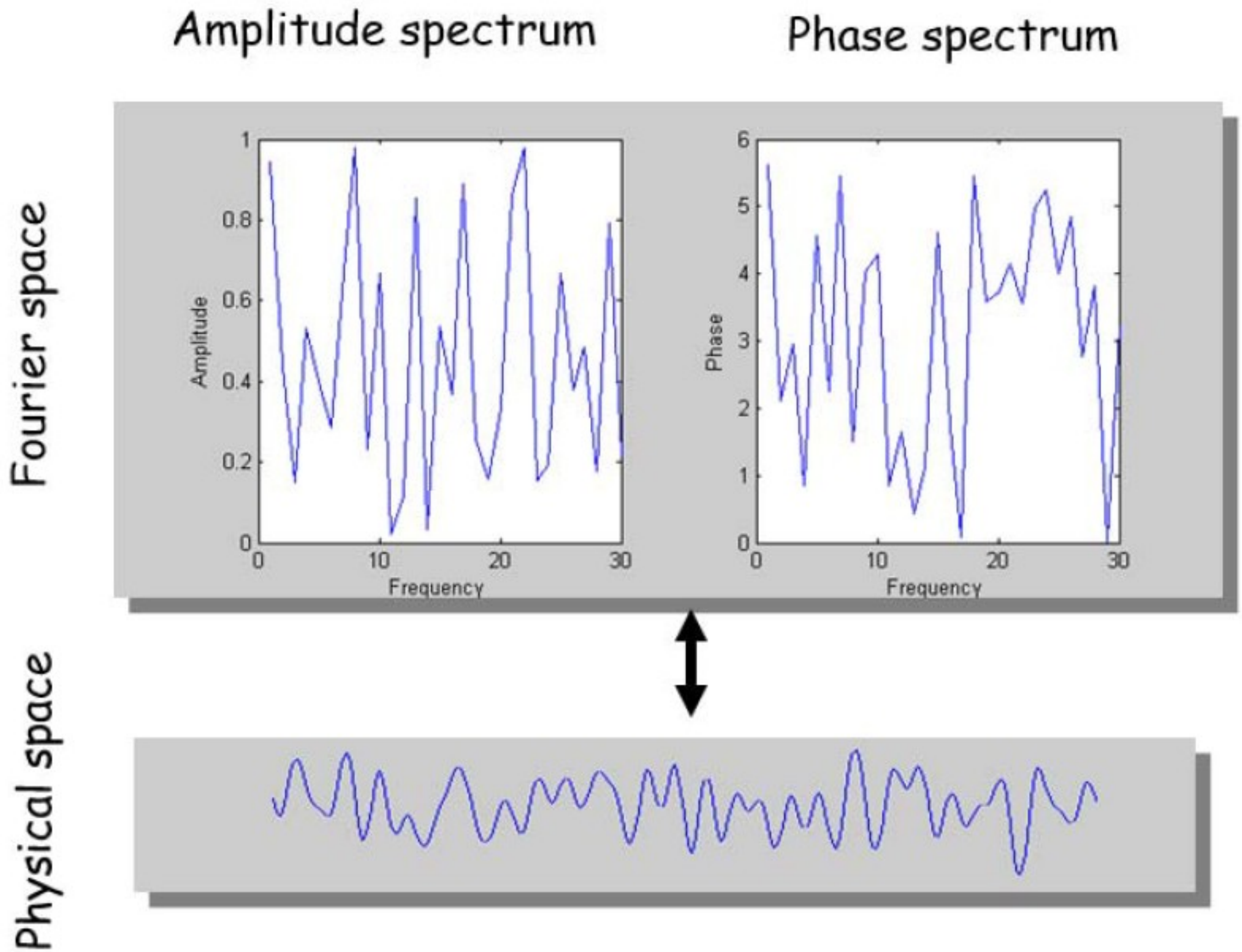


Reference: Cleve Moler, Numerical Computing with MATLAB

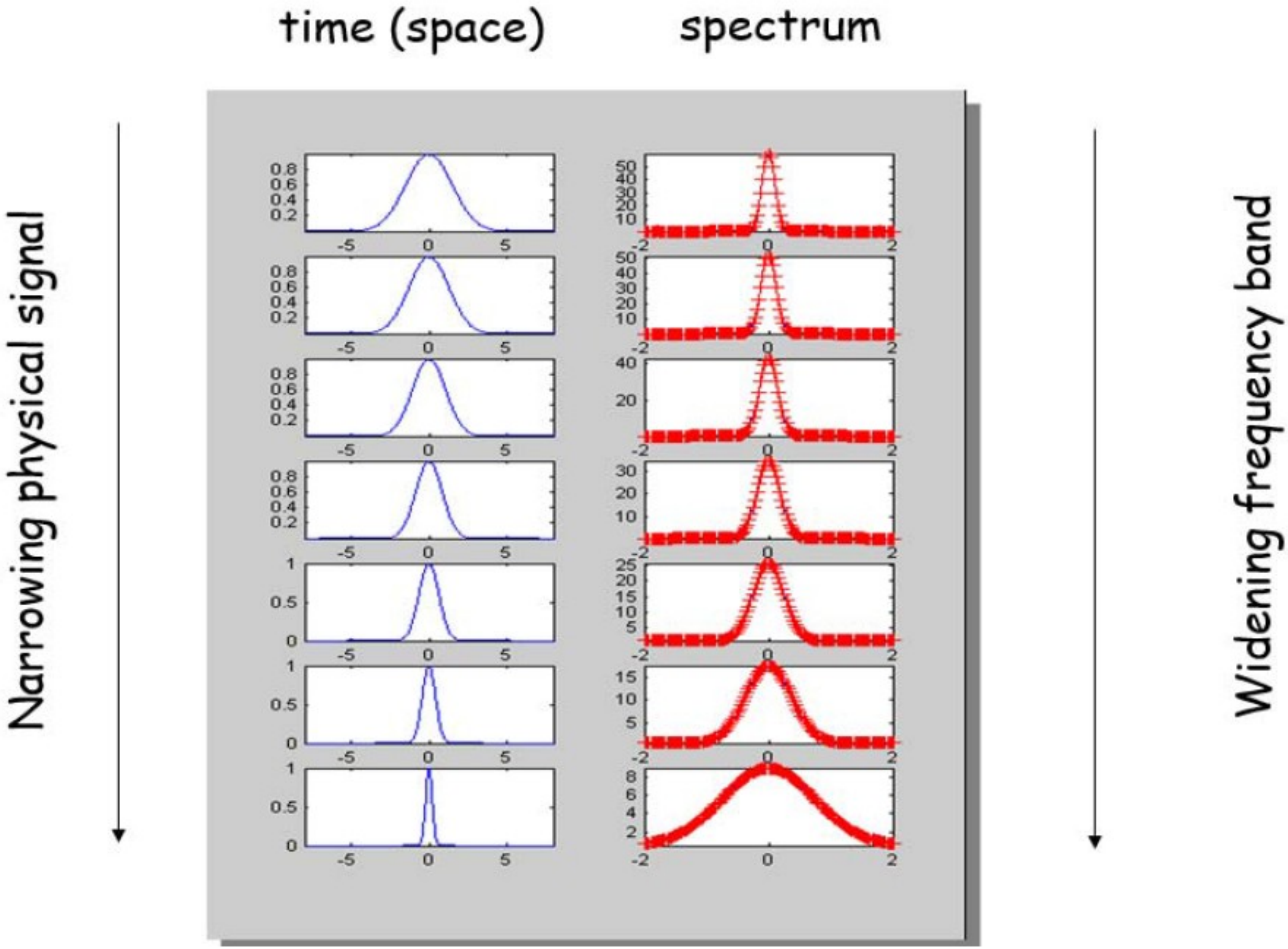
Spectral Synthesis : Computational Geophysics and Data Analysis



Spectral Synthesis : Computational Geophysics and Data Analysis

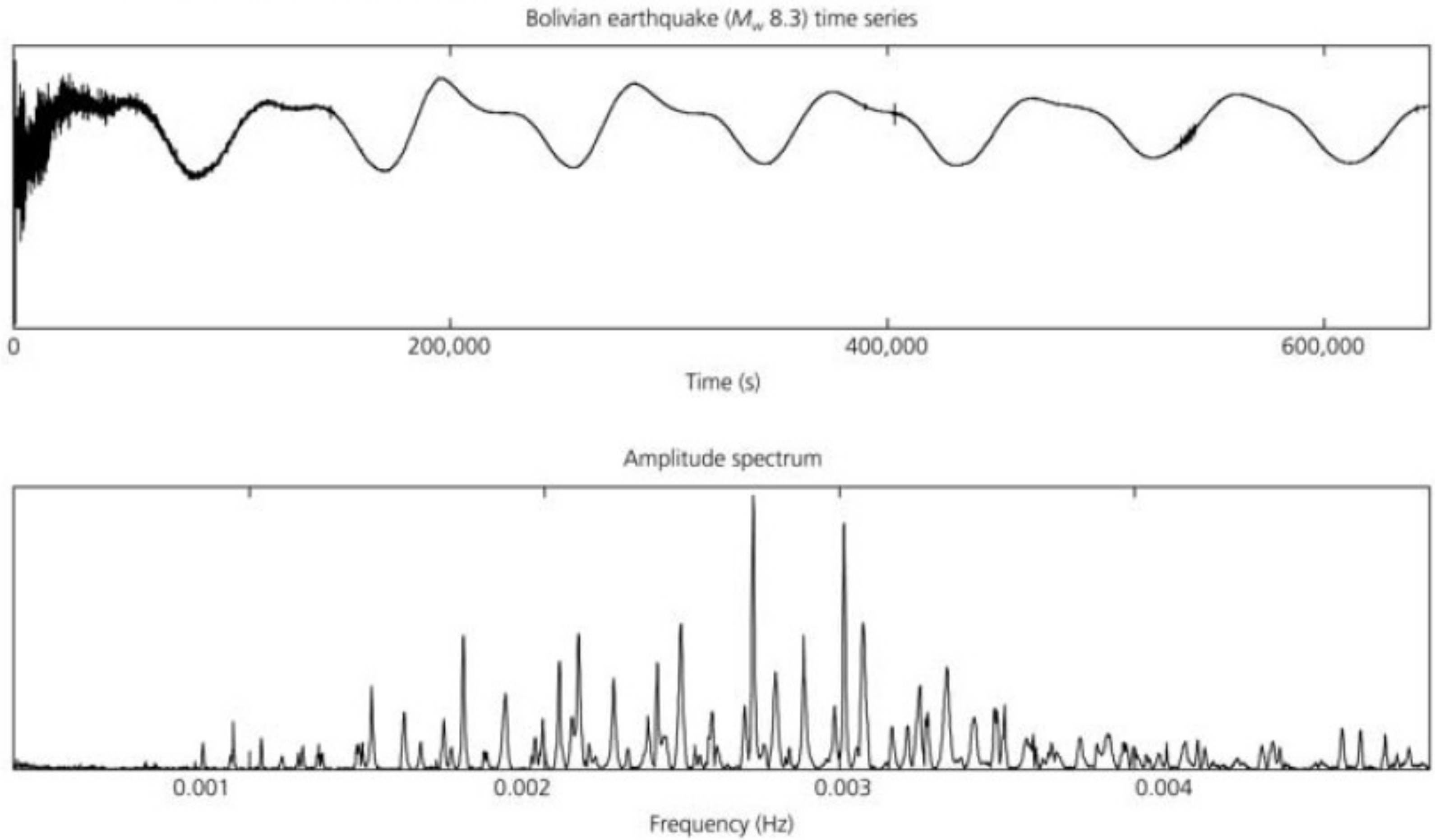


Relationship between the shapes of time-domain and frequency-domain graphs



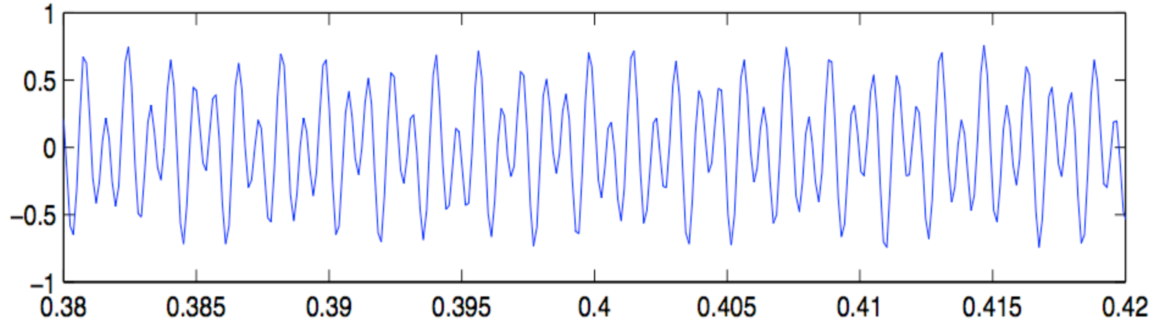
Amplitude Spectrum of an Earthquake

Figure 6.2-4: Amplitude spectra of a vertical-component seismogram from the great 1994 Bolivian earthquake.

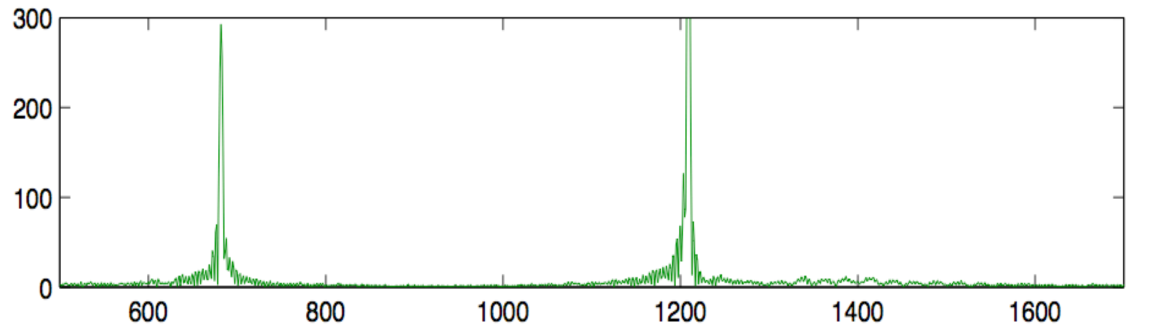


Time Domain vs. Frequency Domain: Digitizing analog signal

Signal. [recording, 8192 samples per second]



Magnitude of discrete Fourier transform.



Reference: Cleve Moler, Numerical Computing with MATLAB

Nyquist-Shannon Sampling Theorem

- If a function x(t) contains no frequencies higher than B cps (cycles per second / Hertz), it is completely determined by giving its ordinates at a series of points spaced < 1/(2B) seconds apart.
  - Equivalently, a band-limited analog signal can be "faithfully" reconstructed from its digitized version if it is sampled at Nyquist Rate, which is twice the largest frequency in the analog signal.

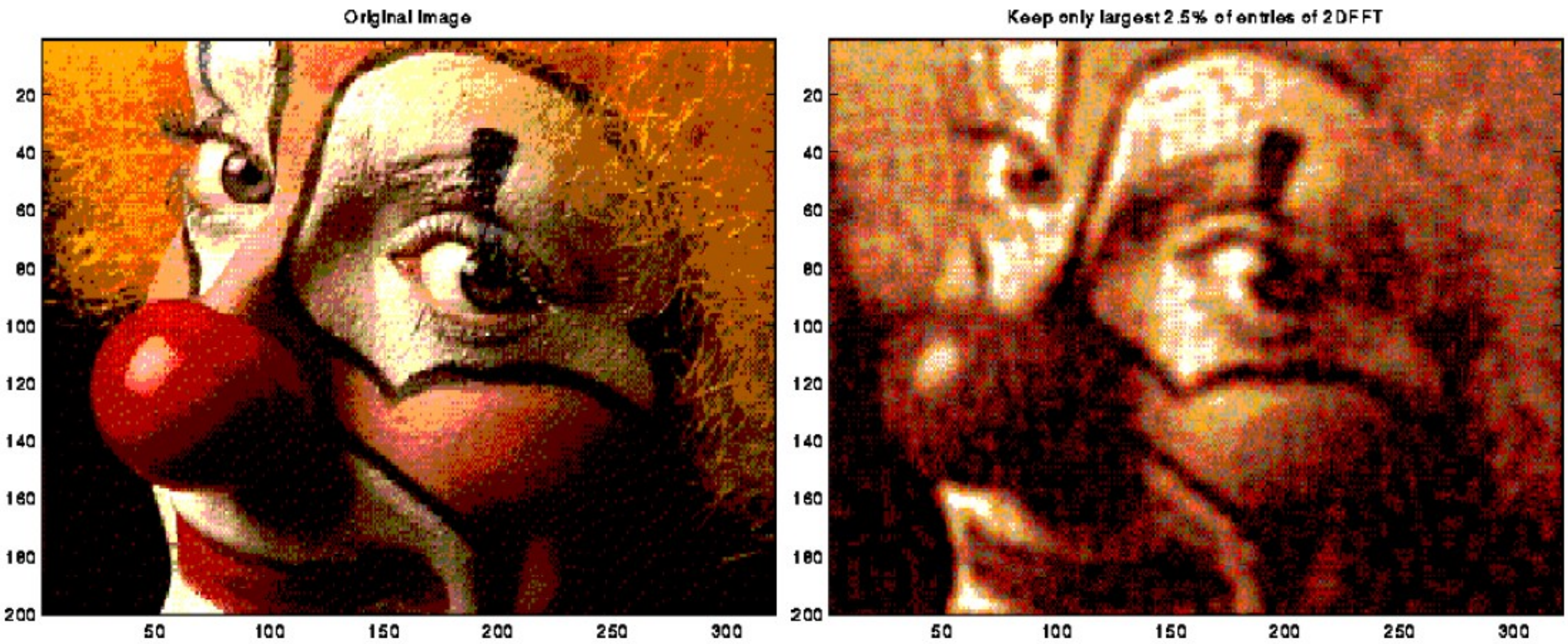
Aliasing absent because orange not allowed.

Using the 2D FFT for image compression

Image = 200x320 matrix of values

Compress by keeping largest 2.5% of FFT components

Similar idea used by jpeg



Fast Fourier Transform (also: Fourier Series, Fourier Transform)

- Signal Processing Viewpoint: Approach to understand time-domain signal in terms of its frequency domain description as a superposition of sinusoids (characterized by their frequency, amplitude and phase)
    - A *periodic* signal of frequency f is made up of sinusoids of frequencies f, 2f, 3f, … [harmonics]
- Computational viewpoint: An encoding technique for efficient computation, especially w.r.t. trade-offs involved in multiplying and evaluating polynomials

5.6 Convolution and FFT

Fast Fourier Transform

FFT. Fast way to convert between time-domain and frequency-domain.

Alternate viewpoint. Fast way to multiply and evaluate polynomials.

we take this approach

> If you speed up any nontrivial algorithm by a factor of a million or so the world will beat a path towards finding useful applications for it. -Numerical Recipes

Fast Fourier Transform: Applications

Applications.

- Optics, acoustics, quantum physics, telecommunications, radar, control systems, signal processing, speech recognition, data compression, image processing, seismology, mass spectrometry…
- Digital media. [DVD, JPEG, MP3, H.264]
- Medical diagnostics. [MRI, CT, PET scans, ultrasound]
- Numerical solutions to Poisson's equation.
- Shor's quantum factoring algorithm.
- …

> The FFT is one of the truly great computational developments of [the 20th] century. It has changed the face of science and engineering so much that it is not an exaggeration to say that life as we know it would be very different without the FFT. -Charles van Loan

Fast Fourier Transform: Brief History

Gauss (1805, 1866). Analyzed periodic motion of asteroid Ceres.

Runge-König (1924). Laid theoretical groundwork.

Danielson-Lanczos (1942). Efficient algorithm, x-ray crystallography.

Cooley-Tukey (1965). Monitoring nuclear tests in Soviet Union and tracking submarines. Rediscovered and popularized FFT.

Importance not fully realized until advent of digital computers.

Polynomials: Coefficient Representation

Polynomial. [coefficient representation]

$$A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$$

$$B(x) = b_0 + b_1 x + b_2 x^2 + \cdots + b_{n-1} x^{n-1}$$

Add. $O(n)$ arithmetic operations.

$$A(x) + B(x) = (a_0 + b_0) + (a_1 + b_1)x + \cdots + (a_{n-1} + b_{n-1})x^{n-1}$$

Evaluate. $O(n)$ using Horner's method.

$$A(x) = a_0 + (x(a_1 + x(a_2 + \cdots + x(a_{n-2} + x(a_{n-1}))\cdots))$$

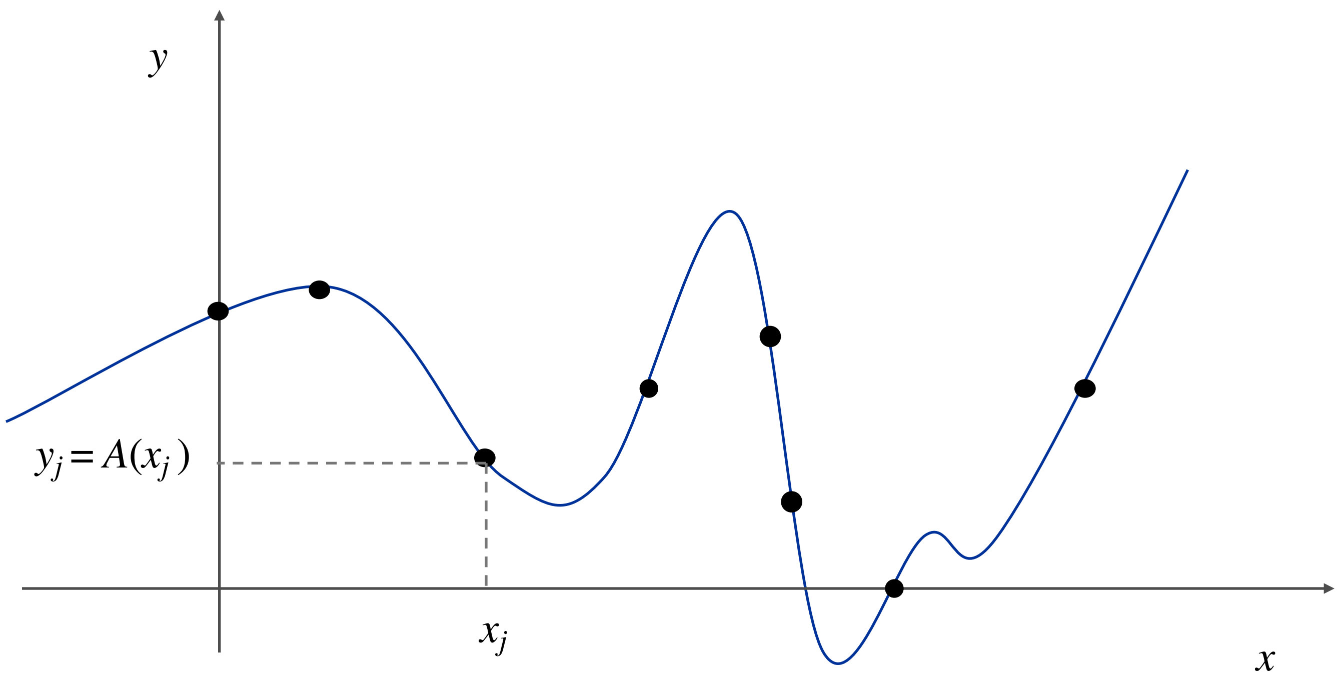
Multiply (convolve). $O(n^2)$ using brute force.

$$A(x) \times B(x) = \sum_{i=0}^{2n-2} c_i x^i, \text{ where } c_i = \sum_{j=0}^{i} a_j b_{i-j}$$

Polynomials: Point-Value Representation

Fundamental theorem of algebra. [Gauss, PhD thesis] A degree n polynomial with complex coefficients has exactly n complex roots.

Corollary. A degree n-1 polynomial $A(x)$ is uniquely specified by its evaluation at n distinct values of x.



Polynomial Basics

Factoring polynomial: $x^2+2x-15=(x-3)(x+5)$

Zeros/roots of a polynomial equation/function:

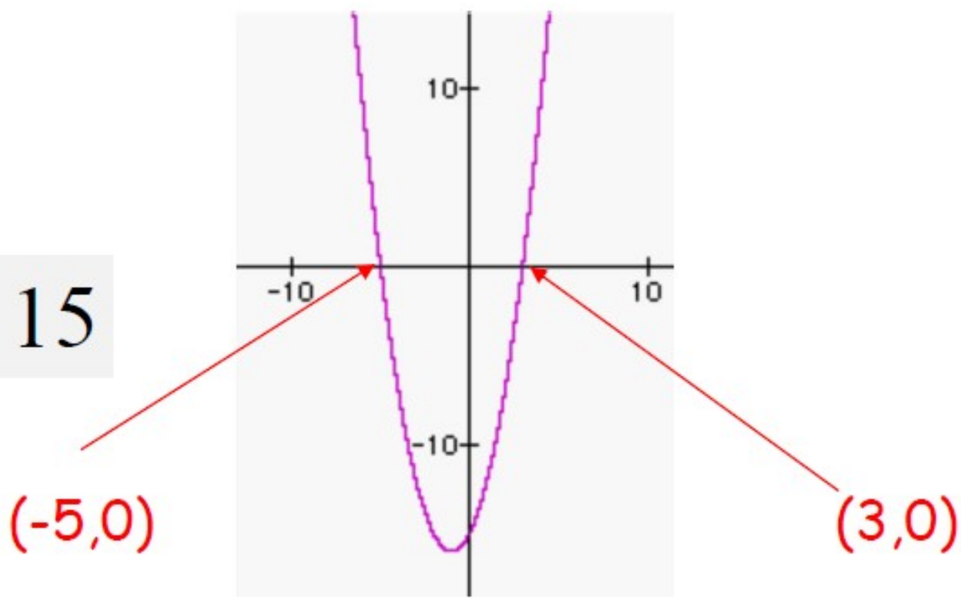
$$(x+5)(x-3)=0$$

$$(x+5)=0 \quad \text{and} \quad (x-3)=0$$

$$x=-5 \qquad\qquad x=3$$

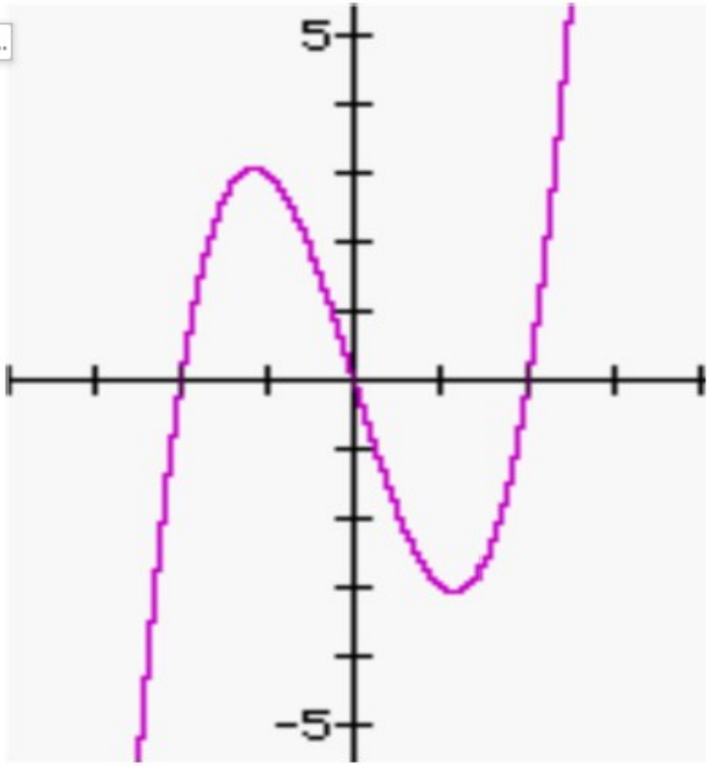
Graphing:

$$y=x^2+2x-15$$



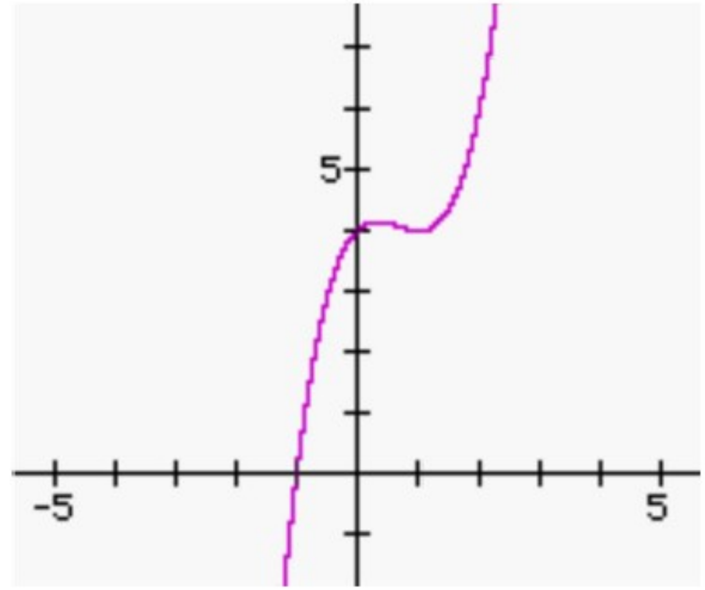
$$y = x^3 - 4x$$

alue Repr...



Roots = -2, 0, 2

$$y = x^3 - 2x^2 + x + 4$$



Real Root = -1
(plus two complex roots)

$$f(x) = x^3 - 5x^2 - 7x + 51$$

Roots: **-3, 4 – i, 4 + i.**

$$= (x + 3)(x - (4 - i))(x - (4 + i))$$

Polynomials: Point-Value Representation

Polynomial. [point-value representation]

$$A(x):\ (x_0, y_0), \ldots, (x_{n-1}, y_{n-1})$$
$$B(x):\ (x_0, z_0), \ldots, (x_{n-1}, z_{n-1})$$

Add. $O(n)$ arithmetic operations.

$$A(x)+B(x):\ \ (x_0, y_0+z_0), \ldots, (x_{n-1}, y_{n-1}+z_{n-1})$$

Multiply (convolve). $O(n)$, but need $2n$-1 points.

$$A(x) \times B(x):\ \ (x_0, y_0 \times z_0), \ldots, (x_{2n-1}, y_{2n-1} \times z_{2n-1})$$

Evaluate. $O(n^2)$ using Lagrange's formula.

$$A(x) = \sum_{k=0}^{n-1} y_k \frac{\prod_{j \neq k} (x - x_j)}{\prod_{j \neq k} (x_k - x_j)}$$

Lagrange Interpolating Polynomial

Data Set	Polynomial
$\{(x_1, y_1)\}$	$p(x) = y_1$
$\{(x_1, y_1), (x_2, y_2)\}$	$p(x) = y_1 \frac{(x - x_2)}{(x_1 - x_2)} + y_2 \frac{(x - x_1)}{(x_2 - x_1)}$
$\{(x_1, y_1), (x_2, y_2), (x_3, y_3)\}$	$p(x) = y_1 \frac{(x - x_2)(x - x_3)}{(x_1 - x_2)(x_1 - x_3)} + y_2 \frac{(x - x_1)(x - x_3)}{(x_2 - x_1)(x_2 - x_3)} + y_3 \frac{(x - x_1)(x - x_2)}{(x_3 - x_1)(x_3 - x_2)}$
$\{(x_1, y_1), (x_2, y_2), \cdots, (x_m, y_m)\}$	$p(x) = y_1 \frac{(x - x_2)(x - x_3)\cdots(x - x_m)}{(x_1 - x_2)(x_1 - x_3)\cdots(x_1 - x_m)} + y_2 \frac{(x - x_1)(x - x_3)\cdots(x - x_m)}{(x_2 - x_1)(x_2 - x_3)\cdots(x_2 - x_m)} + \cdots + y_m \frac{(x - x_1)(x - x_2)\cdots(x - x_{m-1})}{(x_m - x_1)(x_m - x_2)\cdots(x_m - x_{m-1})}$ $p(x) = \sum_{i=1}^{m} y_i \prod_{\substack{j=1 \\ j \neq i}}^{m} \frac{(x - x_j)}{(x_i - x_j)}$

Converting Between Two Polynomial Representations

Tradeoff. Fast evaluation or fast multiplication. We want both!

representation	multiply	evaluate
coefficient	$O(n^2)$	$O(n)$
point-value	$O(n)$	$O(n^2)$

Goal. Efficient conversion between two representations $\Rightarrow$ all ops fast.

$a_0, a_1, \ldots, a_{n-1}$

coefficient representation

$(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})$

point-value representation

Converting Between Two Representations: Brute Force

- Coefficient to point-value. Given a polynomial $a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$, evaluate it at n distinct points $x_0, \ldots, x_{n-1}$.

- $O(n^3)$ for Gaussian elimination

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

$O(n^2)$ for matrix-vector multiply

$O(n^3)$ for Gaussian elimination

Vandermonde matrix is invertible iff x_i distinct

- Point-value to coefficient. Given n distinct points $x_0, \ldots, x_{n-1}$ and values $y_0, \ldots, y_{n-1}$, find unique polynomial $a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$, that has given values at given points.

Divide-and-Conquer

Decimation in frequency. Break up polynomial into low and high powers.

- $A(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5 + a_6x^6 + a_7x^7.$
- $A_{low}(x) = a_0 + a_1x + a_2x^2 + a_3x^3.$
- $A_{high}(x) = a_4 + a_5x + a_6x^2 + a_7x^3.$
- $A(x) = A_{low}(x) + x^4 A_{high}(x).$

Decimation in time. Break polynomial up into even and odd powers.

- $A(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5 + a_6x^6 + a_7x^7.$
- $A_{even}(x) = a_0 + a_2x + a_4x^2 + a_6x^3.$
- $A_{odd}(x) = a_1 + a_3x + a_5x^2 + a_7x^3.$
- $A(x) = A_{even}(x^2) + x\, A_{odd}(x^2).$

Coefficient to Point-Value Representation: Intuition

Coefficient $\Rightarrow$ point-value. Given a polynomial $a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$, evaluate it at n distinct points $x_0, \ldots, x_{n-1}$.

we get to choose which ones!

Divide. Break polynomial up into even and odd powers.

- $A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7$.
- $A_{even}(x) = a_0 + a_2 x + a_4 x^2 + a_6 x^3$.
- $A_{odd}(x) = a_1 + a_3 x + a_5 x^2 + a_7 x^3$.
- $A(x) = A_{even}(x^2) + x\, A_{odd}(x^2)$.
- $A(-x) = A_{even}(x^2) - x\, A_{odd}(x^2)$.

Intuition. Choose two points to be ± 1.

- $A(1) = A_{even}(1) + 1\, A_{odd}(1)$.
- $A(-1) = A_{even}(1) - 1\, A_{odd}(1)$.

Can evaluate polynomial of degree $\leq n$ at 2 points by evaluating two polynomials of degree $\leq \frac{1}{2} n$ at 1 point.

Coefficient to Point-Value Representation: Intuition

Coefficient $\Rightarrow$ point-value. Given a polynomial $a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$, evaluate it at n distinct points $x_0, \ldots, x_{n-1}$.

we get to choose which ones!

Divide. Break polynomial up into even and odd powers.

- $A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7$.
- $A_{even}(x) = a_0 + a_2 x + a_4 x^2 + a_6 x^3$.
- $A_{odd}(x) = a_1 + a_3 x + a_5 x^2 + a_7 x^3$.
- $A(x) = A_{even}(x^2) + x\, A_{odd}(x^2)$.
- $A(-x) = A_{even}(x^2) - x\, A_{odd}(x^2)$.

Intuition. Choose four complex points to be $\pm 1, \pm i$.

- $A(1) = A_{even}(1) + 1\, A_{odd}(1)$.
- $A(-1) = A_{even}(1) - 1\, A_{odd}(1)$.
- $A(i) = A_{even}(-1) + i\, A_{odd}(-1)$.
- $A(-i) = A_{even}(-1) - i\, A_{odd}(-1)$.

Can evaluate polynomial of degree $\le n$ at 4 points by evaluating two polynomials of degree $\le \frac{1}{2}n$ at 2 points.

Discrete Fourier Transform

Coefficient ⇒ point-value. Given a polynomial $a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$, evaluate it at n distinct points $x_0, \ldots, x_{n-1}$.

Key idea. Choose $x_k = \omega^k$ where ω is principal n^{th} root of unity.

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^1 & \omega^2 & \omega^3 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(n-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \omega^{3(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

DFT

Fourier matrix F_n

Roots of Unity

Def. An n^{th} root of unity is a complex number x such that $x^n = 1$.

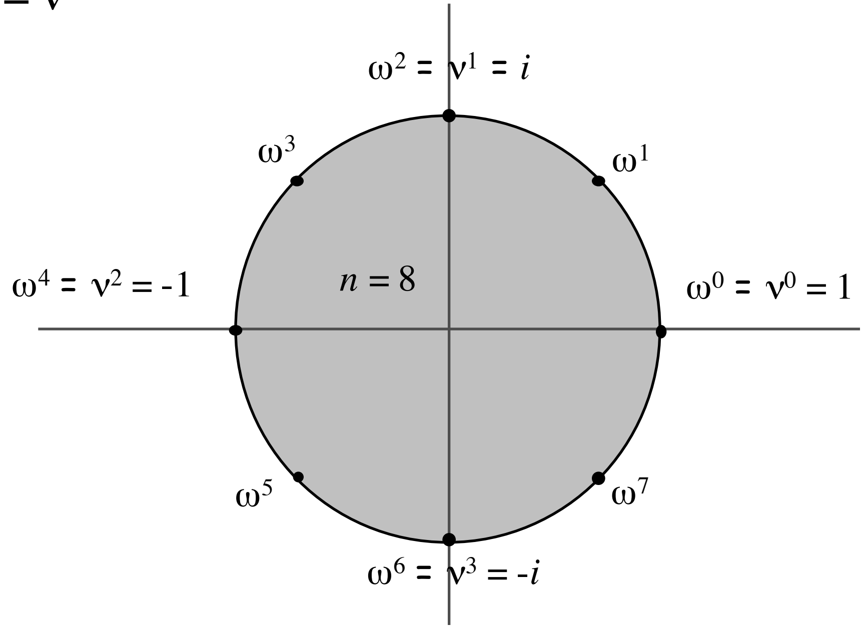
Euler's identity. $e^{ix} = \cos x + i \sin x$.

Fact. The n^{th} roots of unity are: $\omega^0, \omega^1, \ldots, \omega^{n-1}$ where $\omega = e^{2\pi i / n}$.

Pf. $(\omega^k)^n = (e^{2\pi i k / n})^n = (e^{\pi i})^{2k} = (-1)^{2k} = 1$.

Fact. The $\frac{1}{2}n^{th}$ roots of unity are: $\nu^0, \nu^1, \ldots, \nu^{n/2-1}$ where $\nu = \omega^2 = e^{4\pi i / n}$.

Fact. $\omega^2 = \nu$ and $(\omega^2)^k = \nu^k$



Fast Fourier Transform

Goal. Evaluate a degree n-1 polynomial $A(x) = a_0 + \ldots + a_{n-1}\, x^{n-1}$ at its n^{th} roots of unity: $\omega^0, \omega^1, \ldots, \omega^{n-1}$.

Divide. Break up polynomial into even and odd powers.

- $A_{even}(x) \;=\; a_0 + a_2 x + a_4 x^2 + \ldots + a_{n-2}\, x^{n/2 - 1}$.
- $A_{odd}(x) \;=\; a_1 + a_3 x + a_5 x^2 + \ldots + a_{n-1}\, x^{n/2 - 1}$.
- $A(x) \;= A_{even}(x^2) + x\, A_{odd}(x^2)$.

Conquer. Evaluate $A_{even}(x)$ and $A_{odd}(x)$ at the $\frac{1}{2}n^{th}$ roots of unity: $\nu^0, \nu^1, \ldots, \nu^{n/2-1}$.

$$
\begin{aligned}
\nu^k &= (\omega^2)^k \\
&= (\omega^k)^2 \\
&= (\omega^k)^2\,(\omega^n) \\
&= (\omega^{k+\frac{1}{2}n})^2
\end{aligned}
$$

Combine.

$\nu^k = (\omega^k)^2$

- $A(\omega^k) \;= A_{even}(\nu^k) + \omega^k A_{odd}(\nu^k), \quad 0 \le k < n/2$
- $A(\omega^{k+\frac{1}{2}n}) = A_{even}(\nu^k) - \omega^k A_{odd}(\nu^k), \quad 0 \le k < n/2$

$\nu^k = (\omega^{k+\frac{1}{2}n})^2$ $\qquad$ $\omega^{k+\frac{1}{2}n} = -\omega^k$

$$\omega^{\frac{1}{2}n} = (e^{2\pi i/n})^{\frac{1}{2}n} = \cos\pi + i\sin\pi = -1 + i\,0$$

FFT Algorithm

```
fft(n, a₀,a₁,…,aₙ₋₁) {
   if (n == 1) return a₀

   (e₀,e₁,…,eₙ/₂₋₁) ← FFT(n/2, a₀,a₂,a₄,…,aₙ₋₂)
   (d₀,d₁,…,dₙ/₂₋₁) ← FFT(n/2, a₁,a₃,a₅,…,aₙ₋₁)

   for k = 0 to n/2 - 1 {
      ωᵏ ← e^(2πik/n)
      yₖ       ← eₖ + ωᵏ dₖ
      yₖ₊ₙ/₂ ← eₖ - ωᵏ dₖ
   }

   return (y₀,y₁,…,yₙ₋₁)
}
```

FFT Summary

Theorem. FFT algorithm evaluates a degree n-1 polynomial at each of the n^{th} roots of unity in $O(n \log n)$ steps.

assumes n is a power of 2

Running time.

$$T(n) = 2T(n/2) + \Theta(n) \;\Rightarrow\; T(n) = \Theta(n \log n)$$

$$a_0, a_1, \ldots, a_{n\text{-}1} \;\underset{???}{\overset{O(n \log n)}{\rightleftarrows}}\; (\omega^0, y_0), \ldots, (\omega^{n-1}, y_{n-1})$$

coefficient representation

point-value representation

Point-Value to Coefficient Representation: Inverse DFT

Point-value to coefficient. Given n distinct points $x_0, \ldots, x_{n-1}$ and values $y_0, \ldots, y_{n-1}$, find unique polynomial $a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$, that has given values at given points.

$$
\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \end{bmatrix}
=
\begin{bmatrix}
1 & 1 & 1 & 1 & \cdots & 1 \\
1 & \omega^1 & \omega^2 & \omega^3 & \cdots & \omega^{n-1} \\
1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(n-1)} \\
1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(n-1)} \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
1 & \omega^{n-1} & \omega^{2(n-1)} & \omega^{3(n-1)} & \cdots & \omega^{(n-1)(n-1)}
\end{bmatrix}^{-1}
\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \end{bmatrix}
$$

Inverse DFT

Fourier matrix inverse $(F_n)^{-1}$

Inverse DFT

Claim. Inverse of Fourier matrix F_n is given by following formula.

$$
G_n = \frac{1}{n}\begin{bmatrix}
1 & 1 & 1 & 1 & \cdots & 1 \\
1 & \omega^{-1} & \omega^{-2} & \omega^{-3} & \cdots & \omega^{-(n-1)} \\
1 & \omega^{-2} & \omega^{-4} & \omega^{-6} & \cdots & \omega^{-2(n-1)} \\
1 & \omega^{-3} & \omega^{-6} & \omega^{-9} & \cdots & \omega^{-3(n-1)} \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
1 & \omega^{-(n-1)} & \omega^{-2(n-1)} & \omega^{-3(n-1)} & \cdots & \omega^{-(n-1)(n-1)}
\end{bmatrix}
$$

Consequence. To compute inverse FFT, apply same algorithm but use $\omega^{-1} = e^{-2\pi i / n}$ as principal n^{th} root of unity (and divide by n).

Inverse FFT: Proof of Correctness

Claim. F_n and G_n are inverses.

Pf.

$$\left(F_n\, G_n\right)_{k\,k'} \;=\; \frac{1}{n}\sum_{j=0}^{n-1} \omega^{kj}\,\omega^{-j\,k'} \;=\; \frac{1}{n}\sum_{j=0}^{n-1} \omega^{(k-k')\,j} \;=\; \begin{cases} 1 & \text{if } k = k' \\ 0 & \text{otherwise} \end{cases}$$

summation lemma

Summation lemma. Let ω be a principal n^{th} root of unity. Then

$$\sum_{j=0}^{n-1} \omega^{kj} \;=\; \begin{cases} n & \text{if } k \equiv 0 \bmod n \\ 0 & \text{otherwise} \end{cases}$$

Pf.

- If k is a multiple of n then $\omega^k = 1 \;\Rightarrow\;$ series sums to n.
- Each n^{th} root of unity ω^k is a root of $x^n - 1 = (x - 1)\,(1 + x + x^2 + \ldots + x^{n-1})$.
- if $\omega^k \neq 1$ we have: $1 + \omega^k + \omega^{k(2)} + \ldots + \omega^{k(n-1)} = 0 \;\Rightarrow\;$ series sums to 0. ▪

Inverse FFT: Algorithm

```
ifft(n, a_0,a_1,…,a_{n-1}) {
   if (n == 1) return a_0/n

   (e_0,e_1,…,e_{n/2-1}) ← FFT(n/2, a_0,a_2,a_4,…,a_{n-2})
   (d_0,d_1,…,d_{n/2-1}) ← FFT(n/2, a_1,a_3,a_5,…,a_{n-1})

   for k = 0 to n/2 - 1 {
      ω^k ← e^{-2πik/n}
      y_{k+n/2} ← (e_k + ω^k d_k) / n
      y_{k+n/2} ← (e_k - ω^k d_k) / n
   }

   return (y_0,y_1,…,y_{n-1})
}
```

Inverse FFT Summary

Theorem. Inverse FFT algorithm interpolates a degree n-1 polynomial given values at each of the n^{th} roots of unity in $O(n \log n)$ steps.

assumes n is a power of 2

$$a_0, a_1, \ldots, a_{n-1} \underset{O(n \log n)}{\overset{O(n \log n)}{\rightleftarrows}} (\omega^0, y_0), \ldots, (\omega^{n-1}, y_{n-1})$$

coefficient representation

point-value representation

Polynomial Multiplication

Theorem. Can multiply two degree n-1 polynomials in $O(n \log n)$ steps.

coefficient representation

$a_0, a_1, \ldots, a_{n-1}$
$b_0, b_1, \ldots, b_{n-1}$

2 FFTs $O(n \log n)$

$A(\omega^0), \ldots, A(\omega^{2n-1})$
$B(\omega^0), \ldots, B(\omega^{2n-1})$

point-value multiplication $O(n)$

$C(\omega^0), \ldots, C(\omega^{2n-1})$

inverse FFT $O(n \log n)$

coefficient representation

$c_0, c_1, \ldots, c_{2n-2}$

FFT in Practice

Fastest Fourier transform in the West. [Frigo and Johnson]

- Optimized C library.
- Features: DFT, DCT, real, complex, any size, any dimension.
- Won 1999 Wilkinson Prize for Numerical Software.
- Portable, competitive with vendor-tuned code.

Implementation details.

- Instead of executing predetermined algorithm, it evaluates your hardware and uses a special-purpose compiler to generate an optimized algorithm catered to "shape" of the problem.
- Core algorithm is nonrecursive version of Cooley-Tukey radix 2 FFT.
- $O(n \log n)$, even for prime sizes.

Reference: http://www.fftw.org

Integer Multiplication

Integer multiplication. Given two n bit integers $a = a_{n-1} \ldots a_1 a_0$ and $b = b_{n-1} \ldots b_1 b_0$, compute their product $a \cdot b$.

Convolution algorithm.

- Form two polynomials.
- Note: $a = A(2)$, $b = B(2)$.
- Compute $C(x) = A(x) \cdot B(x)$.
- Evaluate $C(2) = a \cdot b$.
- Running time: $O(n \log n)$ complex arithmetic operations.

$$A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$$

$$B(x) = b_0 + b_1 x + b_2 x^2 + \cdots + b_{n-1} x^{n-1}$$

Theory. [Schönhage-Strassen 1971] $O(n \log n \log \log n)$ bit operations.
Theory. [Fürer 2007] $O(n \log n \, 2^{O(\log^* n)})$ bit operations.

Practice. [GNU Multiple Precision Arithmetic Library] GMP proclaims to be "the fastest bignum library on the planet." It uses brute force, Karatsuba, and FFT, depending on the size of n.

Integer Multiplication, Redux

Integer multiplication. Given two n bit integers $a = a_{n-1} \ldots a_1 a_0$ and $b = b_{n-1} \ldots b_1 b_0$, compute their product $a \cdot b$.

"the fastest bignum library on the planet"

Practice. [GNU Multiple Precision Arithmetic Library]
It uses brute force, Karatsuba, and FFT, depending on the size of n.

Integer Arithmetic

Fundamental open question. What is complexity of arithmetic?

Operation	Upper Bound	Lower Bound
addition	$O(n)$	$\Omega(n)$
multiplication	$O(n \log n\ 2^{O(\log^* n)})$	$\Omega(n)$
division	$O(n \log n\ 2^{O(\log^* n)})$	$\Omega(n)$

Extra Slides

Fourier Matrix Decomposition

$$F_n = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^1 & \omega^2 & \omega^3 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(n-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \omega^{3(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{bmatrix}$$

$$I_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad D_4 = \begin{bmatrix} \omega^0 & 0 & 0 & 0 \\ 0 & \omega^1 & 0 & 0 \\ 0 & 0 & \omega^2 & 0 \\ 0 & 0 & 0 & \omega^3 \end{bmatrix} \qquad a = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

$$y = F_n a = \begin{bmatrix} I_{n/2} & D_{n/2} \\ I_{n/2} & -D_{n/2} \end{bmatrix} \begin{bmatrix} F_{n/2}\, a_{even} \\ F_{n/2}\, a_{odd} \end{bmatrix}$$

Factoring

Factoring. Given an n-bit integer, find its prime factorization.

$$2773 = 47 \times 59$$

$$2^{67}-1 = 147573952589676412927 = 193707721 \times 761838257287$$

a disproof of Mersenne's conjecture that $2^{67} - 1$ is prime

```
74037563479561712828046796097429573142593188889231289084936232638972765034028266276891996419625117843995894330502127585370118968098286733173273108930900552505116877063299072396380786710086096962537934650563796359
```

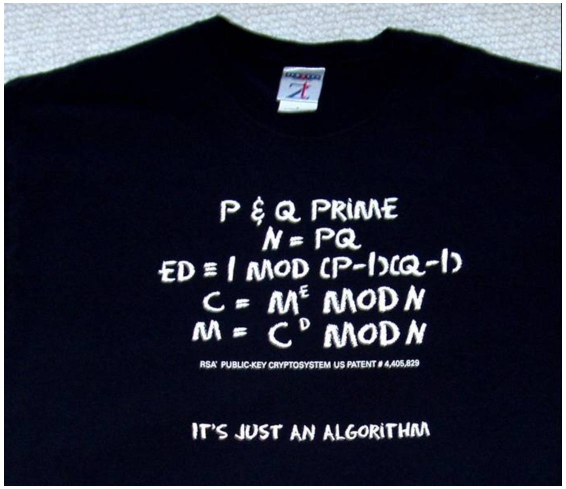
RSA-704
($30,000 prize if you can factor)

Factoring and RSA

Primality. Given an n-bit integer, is it prime?
Factoring. Given an n-bit integer, find its prime factorization.

Significance. Efficient primality testing $\Rightarrow$ can implement RSA.
Significance. Efficient factoring $\Rightarrow$ can break RSA.

Theorem. [AKS 2002] Poly-time algorithm for primality testing.



Shor's Algorithm

Shor's algorithm. Can factor an n-bit integer in $O(n^3)$ time on a quantum computer.

algorithm uses quantum QFT!

Ramification. At least one of the following is wrong:

- RSA is secure.
- Textbook quantum mechanics.
- Extending Church-Turing thesis.

Shor's Factoring Algorithm

Period finding.

2^i	1	2	4	8	16	32	64	128	…
$2^i \bmod 15$	1	2	4	8	1	2	4	8	…
$2^i \bmod 21$	1	2	4	8	16	11	1	2	…

period = 4

period = 6

Theorem. [Euler] Let p and q be prime, and let $N = p\,q$. Then, the following sequence repeats with a period divisible by $(p\text{-}1)\,(q\text{-}1)$:

$$x \bmod N,\;\; x^2 \bmod N,\;\; x^3 \bmod N,\;\; x^4 \bmod N,\; \ldots$$

Consequence. If we can learn something about the period of the sequence, we can learn something about the divisors of $(p\text{-}1)\,(q\text{-}1)$.

by using random values of x, we get the divisors of $(p\text{-}1)\,(q\text{-}1)$, and from this, can get the divisors of $N = p\,q$

Euler's Identity

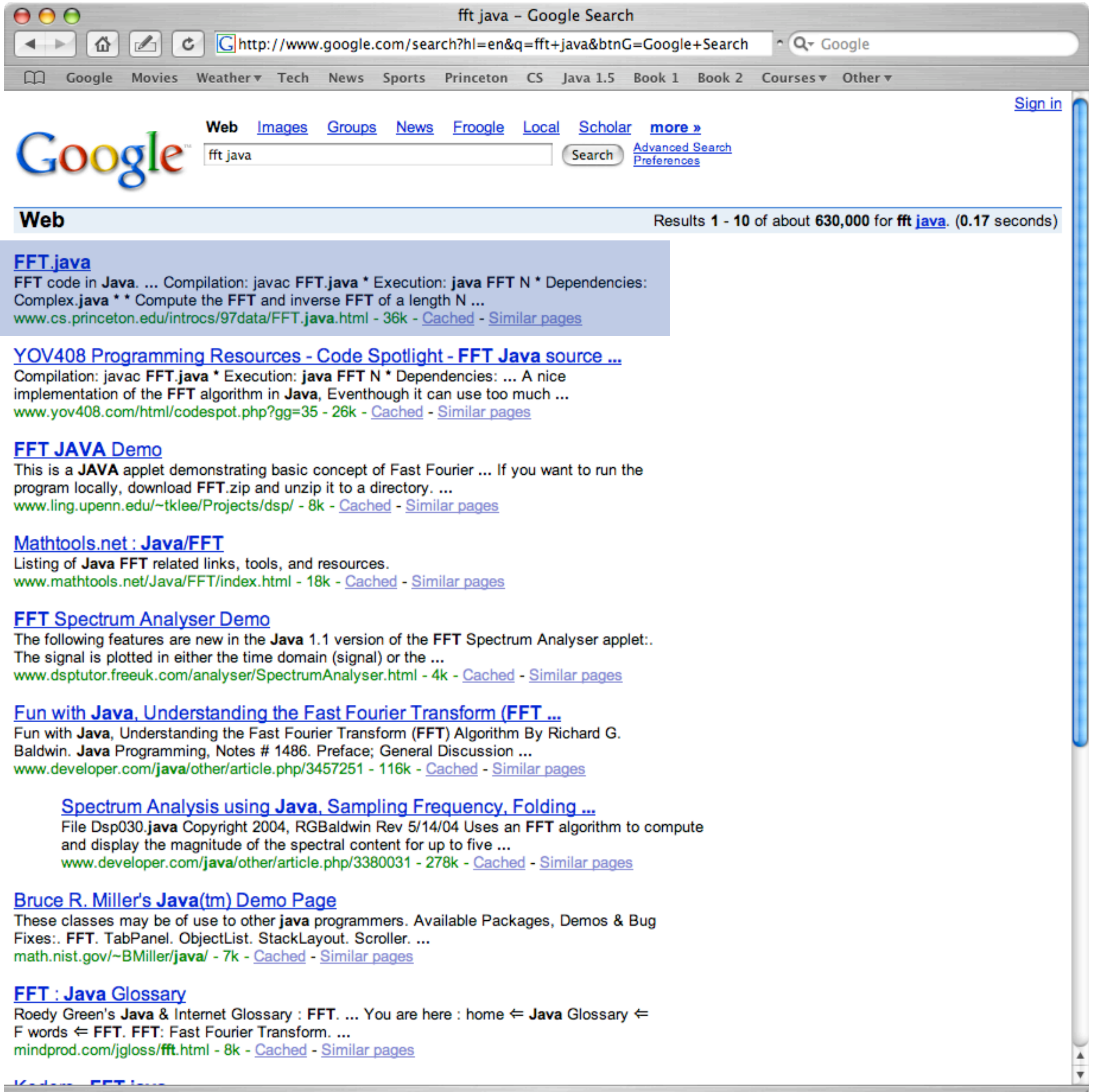
Sinusoids. Sum of sine an cosines.

$$e^{ix} = \cos x + i \sin x$$

Euler's identity

Sinusoids. Sum of complex exponentials.

FFT in Practice ?



Recursion Tree