

Fast Fourier Transform

Andrew S. Barr

University of Washington

December 8, 2010

Outline

The Algorithm

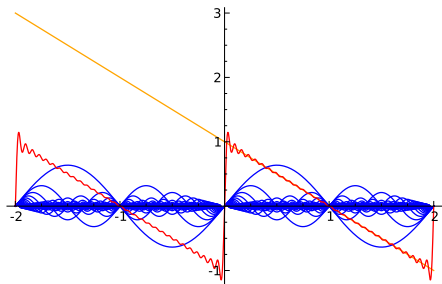
The Applications

The Implementations

Considerations

Varieties

Two Similar Decompositions



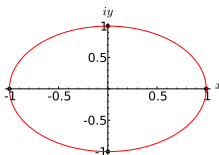
$$\vec{f} = (9, 7, 5, 7)$$

$$\vec{f} = \hat{f}_0 \vec{w}_0 + \hat{f}_1 \vec{w}_1 + \hat{f}_2 \vec{w}_2 + \hat{f}_3 \vec{w}_3$$

$$(9, 7, 5, 7) =$$

$$7 \cdot (1, 1, 1, 1) + 1 \cdot (1, i, -1, -i) + 0 \cdot (1, -1, 1, -1) + 1 \cdot (1, -i, -1, i)$$

Generation of the Basis



The Basis the signal is projected onto: $(\vec{w}_k)_\ell = (e^{i \cdot 2\pi/N})^{k\ell} = \omega_N^{k\ell}$

The Inner Product for which the basis is orthogonal.

$$\langle \vec{z}, \vec{w} \rangle_N = \frac{1}{N} \sum_{m=0}^{N-1} z_m \cdot \overline{w_m}$$

Generation of the w_{0-3} Basis Vectors

$$\vec{w}_0 = \left([\omega_4^0]^0, [\omega_4^0]^1, [\omega_4^0]^2, [\omega_4^0]^3 \right) = (1, 1, 1, 1)$$

$$\vec{w}_1 = \left([\omega_4^1]^0, [\omega_4^1]^1, [\omega_4^1]^2, [\omega_4^1]^3 \right) = (1, i, -1, -i)$$

$$\vec{w}_2 = \left([\omega_4^2]^0, [\omega_4^2]^1, [\omega_4^2]^2, [\omega_4^2]^3 \right) = (1, -1, 1, -1)$$

$$\vec{w}_3 = \left([\omega_4^3]^0, [\omega_4^3]^1, [\omega_4^3]^2, [\omega_4^3]^3 \right) = (1, -i, -1, i)$$

(chart copied from 'Wavelets Made Easy' by Yves Nievergelt)

A Small Example

Example of a Decomposition for Case $N = 4$:

$$\vec{f} = (9, 7, 5, 7)$$

$$\hat{f}_0 = \langle \vec{f}, \vec{w}_0 \rangle_4 = \langle (9, 7, 5, 7), (1, 1, 1, 1) \rangle_4 = \\ \frac{1}{4} \cdot (9 \cdot \overline{1} + 7 \cdot \overline{1} + 5 \cdot \overline{1} + 7 \cdot \overline{1}) = 7$$

$$\hat{f}_1 = \langle \vec{f}, \vec{w}_1 \rangle_4 = \langle (9, 7, 5, 7), (1, i, -1, -i) \rangle_4 = \\ \frac{1}{4} \cdot (9 \cdot \overline{1} + 7 \cdot \overline{i} + 5 \cdot \overline{-1} + 7 \cdot \overline{-i}) = 1$$

$$\hat{f}_2 = \langle \vec{f}, \vec{w}_2 \rangle_4 = \langle (9, 7, 5, 7), (1, -1, 1, -1) \rangle_4 = \\ \frac{1}{4} \cdot (9 \cdot \overline{1} + 7 \cdot \overline{-1} + 5 \cdot \overline{1} + 7 \cdot \overline{-1}) = 0$$

$$\hat{f}_3 = \langle \vec{f}, \vec{w}_3 \rangle_4 = \langle (9, 7, 5, 7), (1, -i, -1, i) \rangle_4 = \\ \frac{1}{4} \cdot (9 \cdot \overline{1} + 7 \cdot \overline{-i} + 5 \cdot \overline{-1} + 7 \cdot \overline{i}) = 1$$

Transform

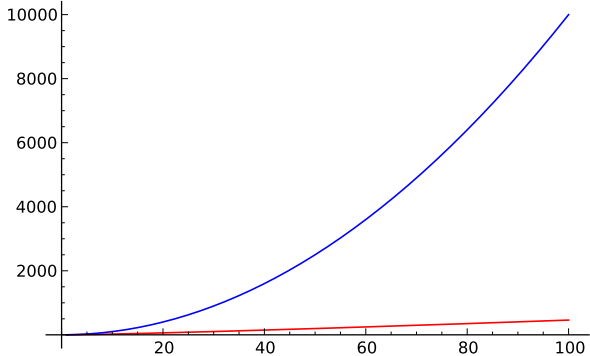
$$\vec{f} = \hat{f}_0 \vec{w}_0 + \hat{f}_1 \vec{w}_1 + \hat{f}_2 \vec{w}_2 + \hat{f}_3 \vec{w}_3$$

Check:

$$(9, 7, 5, 7) =$$

$$7 \cdot (1, 1, 1, 1) + 1 \cdot (1, i, -1, -i) + 0 \cdot (1, -1, 1, -1) + 1 \cdot (1, -i, -1, i)$$

Example courtesy of Yves Nievergelt



For $N = 1024$, the speed-up is nearly 100 fold, and in application, they are often much bigger.

The FFT Formula

The part we have all been waiting for.

Property of $\exp()$ $f(x) \cdot f(y) = f(x + y)$

We split up the arrays based on the parity of their index.

Split a list of length N enumerated by x_m into lists of length $(\frac{N}{2} - 1)$ plus a multiplication by $\frac{2\pi}{N}$

$$m \in \{0, \dots, (\frac{N}{2}) - 1\}$$

$$\text{even } x_m = x_{2m} = 2m \cdot \frac{2\pi}{N} = m \cdot \frac{2\pi}{(N/2)}$$

$$\text{odd } x_m = x_{2m+1} = [2m + 1] \cdot \frac{2\pi}{N} = m \cdot \frac{2\pi}{N} + \frac{2\pi}{N}$$

Re-index and Rescale:

$$(\text{even } \vec{f})_m = \vec{f}_{2m} = f(2m \cdot \frac{2\pi}{N})$$

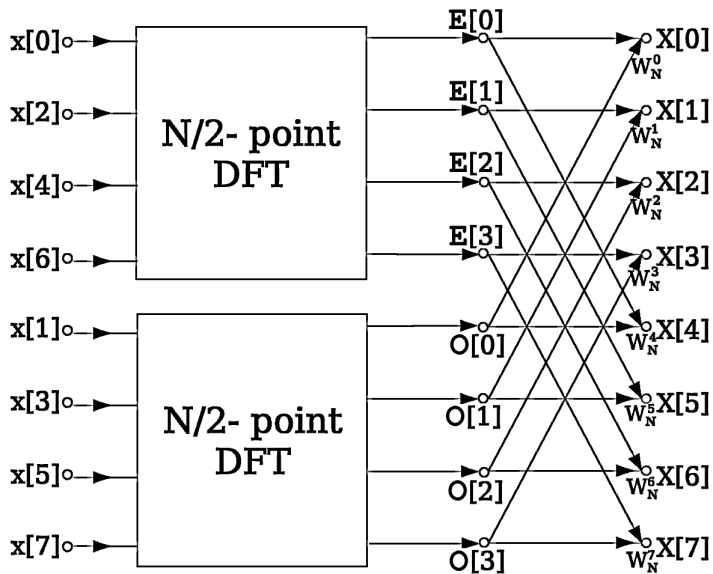
$$(\text{odd } \vec{f})_m = \vec{f}_{2m+1} = f([2m + 1] \cdot \frac{2\pi}{N})$$

The Transform Rule (How 1 transform can be taken by taking 2 smaller ones)

$$\hat{f}_k = \frac{1}{2} \cdot \left(\text{even } \hat{f}_k + [e^{-i \cdot \frac{2\pi}{N}}]^k [\text{odd } \hat{f}_k] \right)$$

$$\hat{f}_{k+(\frac{N}{2})} = \frac{1}{2} \cdot \left(\text{even } \hat{f}_k - [e^{-i \cdot \frac{2\pi}{N}}]^k [\text{odd } \hat{f}_k] \right)$$

Butterfly Diagrams



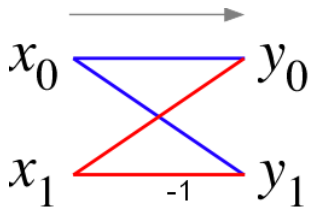
The Sorting

This is a recursive algorithm. Indexing and bookkeeping are the most difficult parts and where one is most likely to get stuck.

For FFTs, the sorting can be simplified by obtaining a binary representation of the index, reversing the digits, and then pairing two by two.

It has to do with continued splitting of the array.

Seen in the butterfly. The general structure (that gets stretched by $N/2$) is below.



Applications

- ▶ All modern telecommunications systems (see Digital Modulation).
- ▶ Image Processing and Signal Processing. see
- ▶ Things to do with lasers (google "FFT lasers")
- ▶ Anything to do with waves (light, sound, seismic, fluid, etc.)
- ▶ General data analysis.
- ▶ Fast multiplication of large numbers. see

sites.google.com/a/iupr.com/ipiu-course/sage-notebooks

www.aimath.org/news/congruentnumbers/howtomultiply.html

The Fastest Fourier Transform in the West

- ▶ Developed by Matteo Frigo and Steven Johnson at MIT to put an end, once and for all, to the fast FFT contest.
- ▶ Uses a bunch of 'codelets' to create 'plans' for an FFT of a particular size.
- ▶ It makes use of a 'simplifier' to make expressions nice. (i.e makes it so k and $-k$ don't both load into a register, see p. 6 in Frigo's paper 'A Fast Fourier Compiler' for fascinating read.)
- ▶ In addition to generating code specific to the radix, FFTW generates code specific to a machine.
- ▶ Many different options available (parallel multidimensional, real data,...)

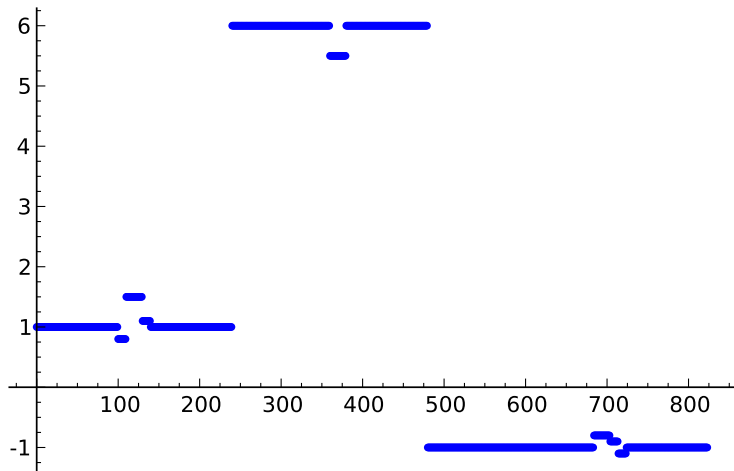
FFTW

- ▶ Fastest code available
- ▶ Very well documented and portable do any platform with a C compiler.
- ▶ Widely used (so google often can get you the answer to your question)
- ▶ Overhead for plan generation.
- ▶ It would be nice if it had more methods (the idea was to just implement FFT and let the DSP or CSE guys handle the software to play with.)
- ▶ Recommended over Christmas break: Read over some of the papers at FFTW website. It is some really cool stuff.

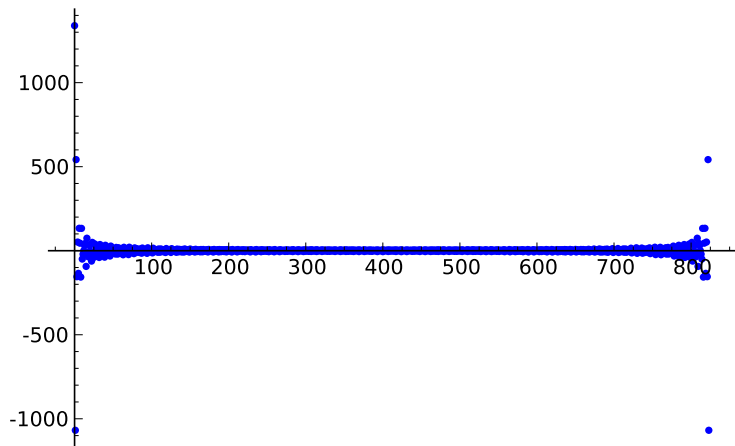
Numpy/SciPy

- ▶ Very well documented. Easy to find many examples online.
- ▶ Many options available. Supports N-dimensional transforms.
- ▶ Optimized for real arrays. Also for mixed radix.
- ▶ Includes useful helper methods. (return freqs, fftshift)
- ▶ It is pretty much a more helpful version of FFTPACK. (the files are .pyf which are Fortran-Python)

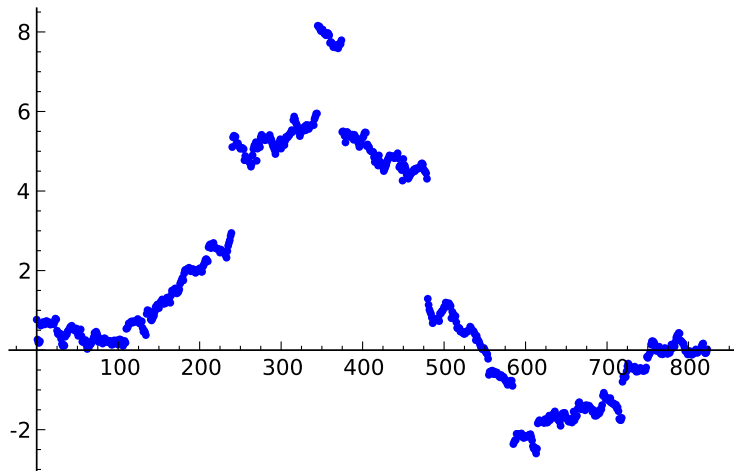
see <http://480.sagenb.org/home/pub/52/> for code



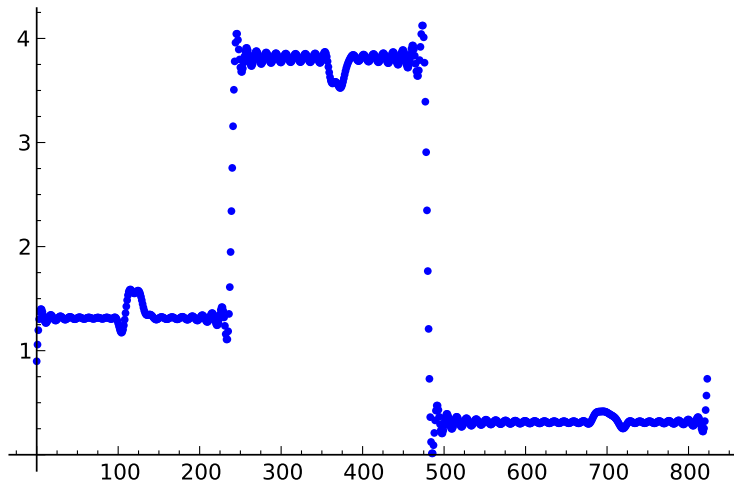
see <http://480.sagenb.org/home/pub/52/> for code



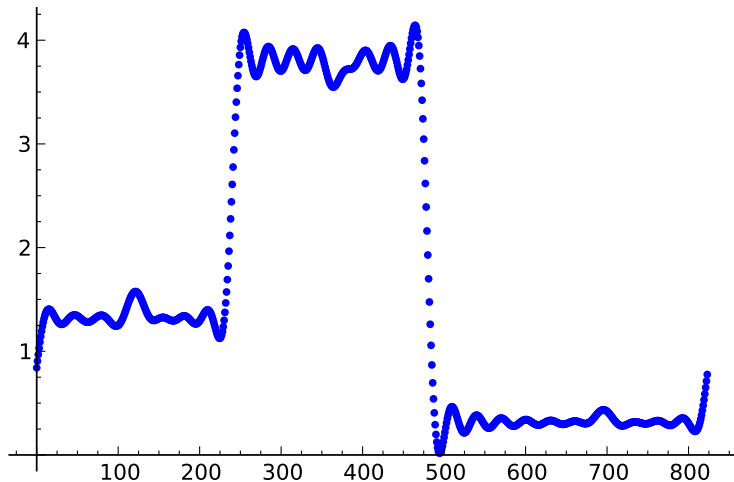
see <http://480.sagenb.org/home/pub/52/> for code



see <http://480.sagenb.org/home/pub/52/> for code



see <http://480.sagenb.org/home/pub/52/> for code



Where/What is Sage's FFT?

Applications Places System 9:23 AM

Search — Sage Reference Manual v4.6 - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://www.sagemath.org/doc/reference/search.html?q=FFT&check_keywords=yes sage documentation

YouTube - Carajo - J... Most Visited YouTube - Carajo - J... Getting Started Latest Headlines /home/sage/sage_in...

Mail ... LAN Airli... FFT Pres... FFT Pres... Tentativ... Numpy E... Data typ... [sage-su... Search ... Sear... x

Sage Reference v4.6 » modules | index

Search

From here you can search these documents. Enter your search words into the box below and click "search". Note that the search function will automatically search for all of the words. Pages containing fewer words won't appear in the result list.

FFT search

Search Results

Search finished, found 4 page(s) matching the search query.

- [sage.finance.time_series.TimeSeries.fft](#) (method, in Time Series)
- [sage.finance.time_series.TimeSeries.iff](#) (method, in Time Series)
- Generic Convolution.**
Generic Convolution. ===== .. This file has been autogenerated. .. automodule:: sage.rings.polynomial.convolution :members: :undoc-members: :show-inheritance:...
- Time Series**
Time Series ===== .. This file has been autogenerated. .. automodule:: sage.finance.time_series :members: :undoc-members: :show-inheritance:...

Sage Reference v4.6 » modules | index

© Copyright 2005–2010, The Sage Development Team. Created using Sphinx 0.6.3.

Where/What is Sage's FFT?

Applications Places System 9:24 AM

Search — Sage Reference Manual v4.6 - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://www.sagemath.org/doc/reference/search.html?q=Fast+Fourier

Search

From here you can search these documents. Enter your search words into the box below and click "search". Note that the search function will automatically search for all of the words. Pages containing fewer words won't appear in the result list.

Fast Fourier search

Search Results

Search finished, found 3 page(s) matching the search query.

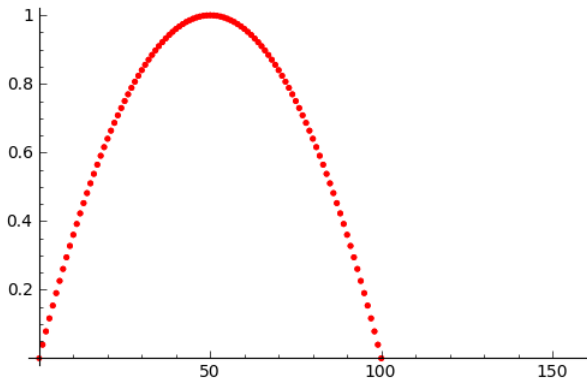
- Elliptic curves over the rational numbers**
Elliptic curves over the rational numbers ===== .. This file has been autogenerated. .. automodule:: sage.schemes.elliptic_curves.ell_rational_field :members: :undoc-members: :show-inheritance...
- Finite word**
Finite word ===== .. This file has been autogenerated. .. automodule:: sage.combinat.words.finite_word :members: :undoc-members: :show-inheritance...
- Time Series**
Time Series ===== .. This file has been autogenerated. .. automodule:: sage.finance.time_series :members: :undoc-members: :show-inheritance...

Sage Reference v4.6 » modules | index

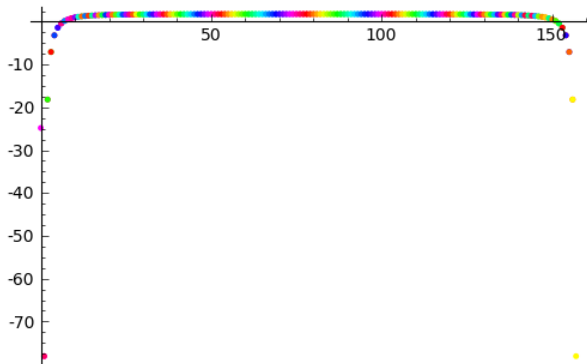
GSL

- ▶ Default used in Sage. (in `sage.sage.gsl.fft.*`)
- ▶ Developed by Physicists at Los Alamos.
- ▶ Not very easy to work with (see my code example. Makes FFT class where array is stored in tuple for complex number.)
- ▶ No extra methods to work with the transform
- ▶ The programming is interesting, with the 'wavetable' and you can see the factorization.
- ▶ They also do have small base cases (2,3,4,5,6,7) resolved for Mixed-Radix.
- ▶ I tested it in (480.sagemb.org/home/pub/45/) and it was much faster than numpy. (282 ns vs 914 μ s)
- ▶ In their documentation they recommend using FFTW for anything serious.

GSL Example - The Signal -see 480.sagemb.org/home/pub/45



GSL Example - The Transform -see 480.sagemb.org/home/pub/45



This happened really fast (in the hundreds of ns for 200 items,
5 μ s for 5000)

FFTPACK

- ▶ Fortran subroutines.
- ▶ Has a long history of use, in both research and industry (been around since 5 years before I was born)
- ▶ Pretty fast too (see comprehensive benchmarks on fftw.org)

Home

This is the home page of the SPIRAL project. The goal of SPIRAL is to push the limits of automation in software and hardware development and optimization for digital signal processing (DSP) algorithms and other numerical kernels beyond what is possible with current tools.

Our basic research question is

Can we teach computers to write fast libraries?

Our flagship is the [SPIRAL program generation system](#), which, entirely autonomously, generates platform-tuned implementations of signal processing transform such as the discrete Fourier transform, discrete cosine transform, and many others. Look at a few [benchmarks](#). But we also provide other online generators (see the right column).

SPIRAL addresses one of the current key problems in numerical software and hardware development: how to achieve close to optimal performance with reasonable coding effort? ([More detailed problem statement](#).)

SPIRAL comprises an [interdisciplinary team](#) of researchers in the areas of signal processing, algorithms, scientific computing, compilers, computer architecture, and mathematics.

In the domain of linear transform, and for standard multicore platforms (Core 2 Duo like), we have achieved complete automation: the computer generation of [general input-size](#), [vectorized](#), [parallel](#) libraries.

Ongoing Research

- Generate code for Intel IPP 6.0, which [now has an entire domain](#) for Spiral-generated code
- [DFT code generation for the Cell BE](#)
- Library generation for matrix-matrix multiplication
- [DFT IP generation for FPGAs](#)
- [Software generation for Viterbi decoding](#)
- Software generation for Synthetic Aperture Radar (SAR) imaging

Online Generators

We provide a number of online generators, which are easy and fun to use or play with.

Online generators currently available

Software (C code)

[Linear transforms](#)
[\(DFT, DCT, many others\)](#)
[Multiplierless multiplication](#)
[Viterbi decoder](#)

Hardware (Verilog)

[DFT](#)
[multiplier blocks](#)
[multiplierless filters](#)

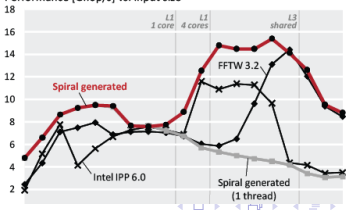
DFT = discrete Fourier transform, DCT = discrete cosine transform.

Browse other [software](#) and [hardware](#).

Featured Result

Complex DFT (Intel Core i7, 2.66 GHz, 4 cores, double precision)

Performance [Gflop/s] vs. input size



SPIRAL Team



NSF Discovery

July 2008: Computer generation of libraries using Spiral was selected as [NSF discovery](#).

Thesis Award

May 2008: Yevgen Voronenko receives the ECE/CMU [best thesis](#) award. With the thesis the computer generation of high performance transform libraries becomes possible.

Intel Collaboration

April 2008: [Intel announces](#) that its flagship performance library IPP (6.0 beta) will have a domain for library functions generated by Spiral.

Beyond Transforms

Our strength are transforms. But we study

Important Considerations

- ▶ Careful interpretation of the data (FFTW guys have a whole section of a paper on it).
- ▶ Reordering of the frequency array. (HShift in Matlab)
- ▶ Differing Conventions.
- ▶ Not using 'frequencies' above the Nyquist limit.(So really only half of the output is valid for a real signal)

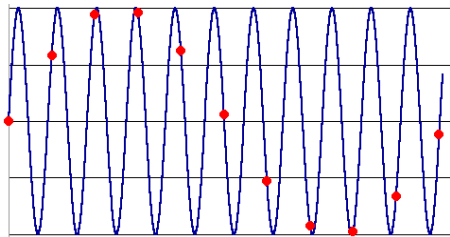


Figure: Aliasing: Avoid at all costs!

Ways to Optimize

- ▶ Factoring into Efficient Base Cases
- ▶ Parallel
- ▶ Real Data
- ▶ Recursion Considerations (tail end, functional paradigm)

Fast Fourier Varieties

- ▶ Non-uniform FFT: Samples taken at specified, albeit irregular intervals. see wikipedia
- ▶ Real-Time FFT: Super interesting algorithm for a sliding window. see www.convict.lu/Jeunes/ultimate_tuff/RFT.pdf
- ▶ In-Place: Interesting from an algorithmic prospective. Useful for those trying to fit as much as possible in a small chip.
- ▶ Mixed-Radix: Used for cases when the array is not size 2^k
- ▶ Fast Fractional Fourier Transform: It is a rotation operation between time and frequency domains. Wikipedia has a good explanation and example.
- ▶ Quantum Fourier Transform (see Wikipedia...)