Fun Fast Fourier Transforms and FORTRAN

Stephen Huenneke

April 20, 2007
Fast Fourier Transforms and Fun with FORTRAN

- What is a Fourier Transform?
Fast Fourier Transforms and Fun with FORTRAN

- What is a Fourier Transform?
- Discretizing the Fourier Transform
Fast Fourier Transforms and Fun with FORTRAN

- What is a Fourier Transform?
- Discretizing the Fourier Transform
- Direct Method
  - Easily Implemented
  - Poor Performance
Fast Fourier Transforms and Fun with FORTRAN

- What is a Fourier Transform?
- Discretizing the Fourier Transform
- Direct Method
  - Easily Implemented
  - Poor Performance
- Cooley-Tukey
  - History
  - Divide and Conquer
  - Significant Gains
Fast Fourier Transforms and Fun with FORTRAN

- What is a Fourier Transform?
- Discretizing the Fourier Transform
- Direct Method
  - Easily Implemented
  - Poor Performance
- Cooley-Tukey
  - History
  - Divide and Conquer
  - Significant Gains
- Gorey FORTRAN Details
Fast Fourier Transforms and Fun with FORTRAN

- What is a Fourier Transform?
- Discretizing the Fourier Transform
- Direct Method
  - Easily Implemented
  - Poor Performance
- Cooley-Tukey
  - History
  - Divide and Conquer
  - Significant Gains
- Gorey FORTRAN Details
- Conclusion
Fun Fast Fourier Transforms and FORTRAN

What is the Fourier Transform?
Discretizing the Fourier Transform
Direct Method
Cooley-Tukey Algorithm
Conclusion

Clean Signal

Un corrupted Signal combination

time (milliseconds)

Stephen Huenneke
Clean Signal

Single-Sided Amplitude Spectrum of y(t)
What is the Fourier Transform?
Discretizing the Fourier Transform
Direct Method
Cooley-Tukey Algorithm
Conclusion

Noisy Signal

Signal with Some Random Noise

Stephen Huenneke
Noisy Signal

Single-Sided Amplitude Spectrum of y(t)

Frequency (Hz)

[Graph showing the amplitude spectrum of a noisy signal]
Discretizing the Fourier Transform

\[ F(\omega) = \int_{-\infty}^{\infty} f(x) e^{-i\omega x} dx \]
Discretizing the Fourier Transform

- \( F(\omega) = \int_{-\infty}^{\infty} f(x) e^{-i\omega x} \, dx \)

- By sampling at points at regular intervals of \( \tau \) we can construct a new discrete sum:
  \[
  D(\omega) = \sum_{r=0}^{n-1} f(r\tau) e^{-i\omega r\tau}
  \]
Discretizing the Fourier Transform

- \( F(\omega) = \int_{-\infty}^{\infty} f(x) e^{-i\omega x} \, dx \)

- By sampling at points at regular intervals of \( \tau \) we can construct a new discrete sum:
  \[
  D(\omega) = \sum_{r=0}^{n-1} f(r\tau) e^{-i\omega r\tau}
  \]

- Observe that we introduced periodicity by discretizing the transform that doesn’t exist in the integral form. This produces recycled data outside of the range of \( n \) values.
Applications Become Evident

• What can we use this new DFT for?
Applications Become Evident

• What can we use this new DFT for?
• Quality Assurance, Cable TV, Communications
Applications Become Evident

- What can we use this new DFT for?
- Quality Assurance, Cable TV, Communications
- Analysis of high noise signals.
What is the Fourier Transform?

Discretizing the Fourier Transform

Direct Method

Cooley-Tukey Algorithm

Conclusion

Direct Algorithms

\[ D(\omega) = \sum_{r=0}^{n-1} f(r\tau)e^{-i\omega r\tau} \]

- Very simple algorithm
Direct Algorithms

\[ D(\omega) = \sum_{r=0}^{n-1} f(r\tau) e^{-i\omega r\tau} \]

- Very simple algorithm
- Quickly, simply implemented in several lines of code.
Direct Algorithms

\[ D(\omega) = \sum_{r=0}^{n-1} f(r\tau)e^{-i\omega r\tau} \]

- Very simple algorithm
- Quickly, simply implemented in several lines of code.
- Performs acceptably for small samples.
Direct Algorithms

- Brute force approach consumes resources
Direct Algorithms

- Brute force approach consumes resources
- Precalculated, saved values can speed up somewhat.
Direct Algorithms

- Brute force approach consumes resources
- Precalculated, saved values can speed up somewhat.
- Direct implementations of the transform are always bounded at $n^2$ running time.
Cooley-Tukey Algorithm

- James Cooley worked with Richard Garwin and John Tukey to devise a way to reduce the number of computations in a transform.
Cooley-Tukey Algorithm

• James Cooley worked with Richard Garwin and John Tukey to devise a way to reduce the number of computations in a transform.

• While not the first to discover the properties of the FFT, Cooley’s work is some of the widest applied in industry.
Cooley-Tukey Algorithm

- James Cooley worked with Richard Garwin and John Tukey to devise a way to reduce the number of computations in a transform.
- While not the first to discover the properties of the FFT, Cooley’s work is some of the widest applied in industry.
- By picking a highly composite sample size, we can reduce the computational complexity to $N \log N$ time.
Cooley-Tukey Algorithm

Radix-2

- We split the DFT into even and odd indexed elements. Then perform smaller DFT’s on those sums.
Cooley-Tukey Algorithm

Radix-2

- We split the DFT into even and odd indexed elements. Then perform smaller DFT’s on those sums.
- Now we can exploit the periodicity introduced in the original discretization process.
Cooley-Tukey Algorithm
Radix-2

- We split the DFT into even and odd indexed elements. Then perform smaller DFT’s on those sums.
- Now we can exploit the periodicity introduced in the original discretization process.
- Applying iteratively, we can continue to reduce the size of the computations.
Cooley-Tukey Algorithm
Radix-2

- Compared to hand-calcutions and other computer programs available in the mid-60’s this was between 100,000 and 800,000 times faster that previous algorithms.
Cooley-Tukey Algorithm
Radix-2

- Compared to hand-calculations and other computer programs available in the mid-60’s this was between 100,000 and 800,000 times faster than previous algorithms.
- Can be implemented fairly simply in a chip.
Cooley-Tukey Algorithm
Radix-2

- Compared to hand-calculations and other computer programs available in the mid-60’s this was between 100,000 and 800,000 times faster than previous algorithms.
- Can be implemented fairly simply in a chip.
- Real-time applications become practical.
Conclusion

- References
  - J. Cooley, ‘How the FFT Gained Acceptance’
  - M. Cartwright, ‘Fourier Methods’

My website: http://www.cs.umb.edu/* *shuenne/
Conclusion

• References
  • J. Cooley, ‘How the FFT Gained Acceptance’
  • M. Cartwright, ’Fourier Methods’
  • L. Ludeman, ‘Fundamentals of Digital Signal Processing’

• Thank Yous
  • Professor Steven Jackson
  • UMass Boston Math and Computer Science Faculty
  • Siena College

My website: http://www.cs.umb.edu/* *shuenne/