# Fourier Transform

- Fourier transform converts a physical-space (or time series) representation of a function into frequency-space

  - Equivalent representation of the function, but gives a new window into its behavior

  - Inverse operation exists

- For discrete data, the discrete analog of the Fourier transform gives:

  - Amplitude and phase at discrete frequencies (wavenumbers)

  - Allows for an investigation into the periodicity of the discrete data

  - Allows for filtering in frequency-space

  - Can simplify the solution of PDEs: derivatives change into multiplications in frequency space.

# Discrete Fourier Transform

- The discrete Fourier transform (DFT) operates on discrete data
  - Usually we have evenly-spaced, real data
    - E.g. a time-series from an experiment
    - Simulation data for a velocity field
- DFT transforms the N spatial/temporal points into N frequency points
  - Transform: $$F_k = \sum_{n=0}^{N-1} f_n e^{-2\pi i n k/N}$$

  - Inverse: $$f_n = \frac{1}{N} \sum_{k=0}^{N-1} F_k e^{2\pi i n k/N}$$

  - This is the form used in NumPy, Garcia, and others. Pang splits the normalization

# Notation

- Many different notations are used in various texts
  - Original function: $f(x)$ or $f(t)$
  - Transformed function: $F(k), \mathcal{F}(k), \hat{f}(k)$
- For the discrete version:
  - Original function: $f_n$
  - Transformed function: $F_k, \mathcal{F}_k, \hat{f}_k$

# Real Space vs. Frequency Space

- **What are the significance of the Re and Im parts?**

  - Recall that we are integrating with $e^{-2\pi i k x}$

  - Euler's formula: $e^{i x} = \cos(x) + i \sin(x)$

  - Real part represents the cosine terms, symmetric functions

  - Imaginary part represents the sine terms, antisymmetric functions

  - If $f_n$ is Real, then:

  $$\mathrm{Re}(F_k) = \sum_{n=0}^{N-1} f_n \cos\left(\frac{2\pi nk}{N}\right)$$

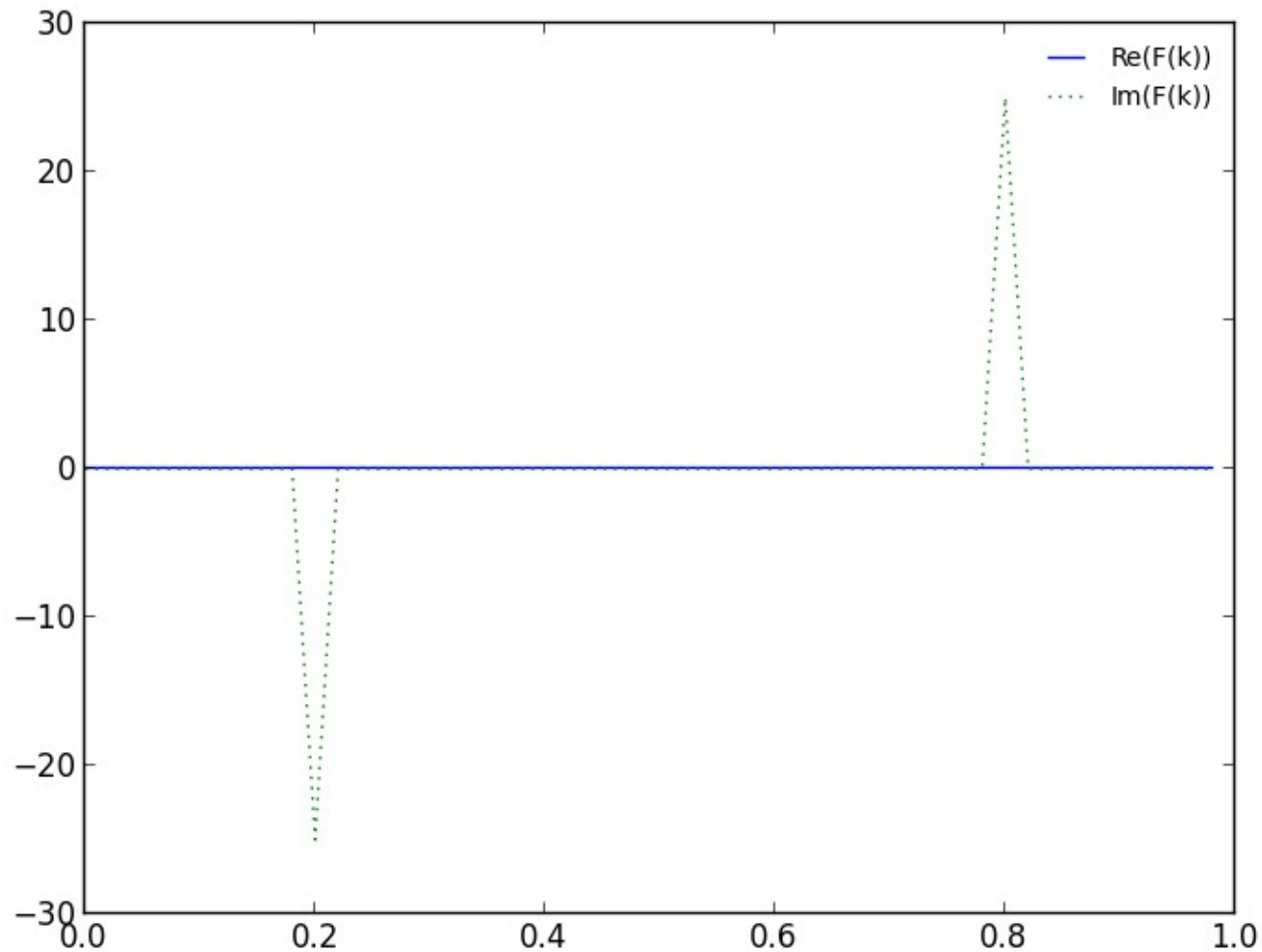  $$\mathrm{Im}(F_k) = \sum_{n=0}^{N-1} f_n \sin\left(\frac{2\pi nk}{N}\right)$$

# DFT Example

- Implementing the discrete Fourier transform is simple

  - Double sum: for each wavenumber, we sum over all the spatial points

```python
def dft(f_n):

    N = len(f_n)
    f_k = numpy.zeros( (N), dtype=numpy.complex128)

    k = 0
    while k < N:

        n = 0
        while n < N:
            f_k[k] += f_n[n]*numpy.exp(-2.0*math.pi*1j*n*k/N)
            n += 1

        k += 1

    return f_k
```

# DFT Example

- DFT of $\sin(2\,\pi\,f_0\,x)$ with $f_0 = 0.2$

# DFT Example

- Note that in the DFT, nowhere does the physical coordinate value, $x_n$ , enter—instead, we just look at the index $n$ itself

    - This assumes that the data is regularly gridded

- In this index space, the smallest wavelength is from one cell to the next, and the smallest frequency is $1/N$

    - Note that this means that if we add points to our data, then we open up higher and higher frequencies (in terms of index space)

- Clearly there is a physical scale for the frequency

$$\nu_k = \frac{k}{N} \cdot \frac{1}{\Delta x} = \frac{k}{N} \cdot \frac{N}{L}$$

    - Lowest frequency: $1/L$
    - Highest frequency ~ $1/\Delta x$

- $k = 0$ is special:

$$\mathrm{Re}(F_0) = \sum_{n=0}^{N-1} f_n \cos\left(\frac{2\pi n 0}{N}\right) = \sum_{n=0}^{N-1} f_n$$

$$\mathrm{Im}(F_k) = \sum_{n=0}^{N-1} f_n \sin\left(\frac{2\pi n 0}{N}\right) = 0$$

– This is sometimes termed the DC offset

# FFT = DFT

- The Fast Fourier Transform (FFT) is equivalent to the discrete Fourier transform

  - Faster because of special symmetries exploited in performing the sums

  - $O(N \log N)$ instead of $O(N^2)$

- Both texts offer a reasonable discussion on how the FFT works—we'll defer it to those sources.

# Normalization

- Normalization of the forward and inverse transforms follows from Parseval's theorem:

$$\int_{-\infty}^{\infty} |f(x)|^2 dx = \int_{-\infty}^{\infty} |F(k)|^2 dk$$

  - Discrete form(taking $\Delta x = 1$ and $\Delta k = 1/N$ )

$$\sum_{n=0}^{N-1} |f_n|^2 = \frac{1}{N} \sum_{k=0}^{N-1} |F_k|^2$$

- The definition of the inverse transformation compatible with this condition is

$$f_n = \frac{1}{N} \sum_{k=0}^{N-1} F_k e^{2\pi i n k/N}$$

# Normalization

- To illustrate this, consider the transform of 1

  - Analytically: $f(x) = 1 \iff F(k) = \delta(k)$

  - Discrete transform:

$$F_k = \sum_{n=0}^{N-1} 1 \cdot e^{-2\pi i k n/N} = \sum_{n=0}^{N-1} [\cos(2\pi k n/N) + i \sin(2\pi k n/N)]$$

    - For k = 0, cos(0) = 1, sin(0) = 0, so $F_0 = N$

    - For k > 0, we are essentially doing a discrete sum of the cosine and sine using equally-spaced points, and the sum is always over a multiple of a full wavelength, therefore, $F_k = 0$

  - Discrete inverse transform:

$$f_n = \frac{1}{N} \sum_{k=0}^{N-1} \underbrace{F_k} e^{2\pi i n k/N} = \frac{1}{N} N e^{2\pi i n 0/N} = 1$$

only non-zero
for k = 0

# Normalization

- When plotting, we want to put the physical scale back in, as well as make the correspondence to the continuous representation

  - Already saw how to put the wavenumbers into a physical frequency

  - Rewrite inverse expression:

    This is what we plot

    $$f_n = \frac{1}{N} \sum_{n=0}^{N-1} F_k e^{2\pi i n k/N} = \sum_{n=0}^{N-1} \left( \frac{F_k}{N} \right) e^{2\pi i n k/N}$$

    This looks like the continuous form

  - Another interpretation: $F_k \, / \, N \Leftrightarrow F_k \, dk$

# Real Space vs. Frequency Space

- Imagine transforming a real-valued function, $f$, with $N$ data points

  - The FFT of $f$ will have $N/2$ complex data points

  - Same amount of information in each case, but each complex point corresponds to 2 real numbers.

    - This is a consequence of the analytic Fourier transform satisfying $F(-k) = F^\star(k)$ if $f(x)$ is real

  - Most FFT routines will return $N$ complex points—half of them are duplicate, i.e. they don't add to the information content

    - Often, there are specific implementations optimized for the case where the function is real (e.g. rfft)

  - This affects normalization (note k=0 different)

- This is also referred to as aliasing.  The maximum frequency, $1/(2\,\Delta x)$ is called the Nyquist frequency

# Power Spectrum

- The power spectrum is simply defined as:

$$P(k) = |F(k)|^2 = F(k)F^\star(k)$$

  - Single number showing the importance/weight of a given wavenumber
- Also useful to look at the phase at each wavenumber

# Python/NumPy's FFT

- `numpy.fft`:
  http://docs.scipy.org/doc/numpy/reference/routines.fft.html

- `fft`/`ifft`: 1-d data

  - By design, the k=0, ... N/2 data is first, followed by the negative frequencies.  These later are not relevant for a real-valued f(x)

  - k's can be obtained from `fftfreq(n)`

  - `fftshift(x)` shifts the k=0 to the center of the spectrum

- `rfft`/`irfft`: for 1-d real-valued functions.  Basically the same as `fft`/`ifft`, but doesn't return the negative frequencies

- 2-d and n-d routines analogously defined

# Python's FFT

- It's always a good idea to run some simple tests to make sure the FFT is behaving the way you expect

  - $\sin(2\pi f_0 x)$ —should be purely imaginary at a single wavenumber

  - $\cos(2\pi f_0 x)$ —should be purely real at a single wavenumber

  - $\sin(2\pi f_0 x + \pi/4)$ —should have equal magnitude real and imaginary parts at a single wavenumber

Transform a single mode sine ($f_0 = 0.2$) and inverse transform back—do you get the original result (to roundoff)?

Notice that the since sine is odd, the FFT is only non-zero in the imaginary component.

And... we get back our original component.
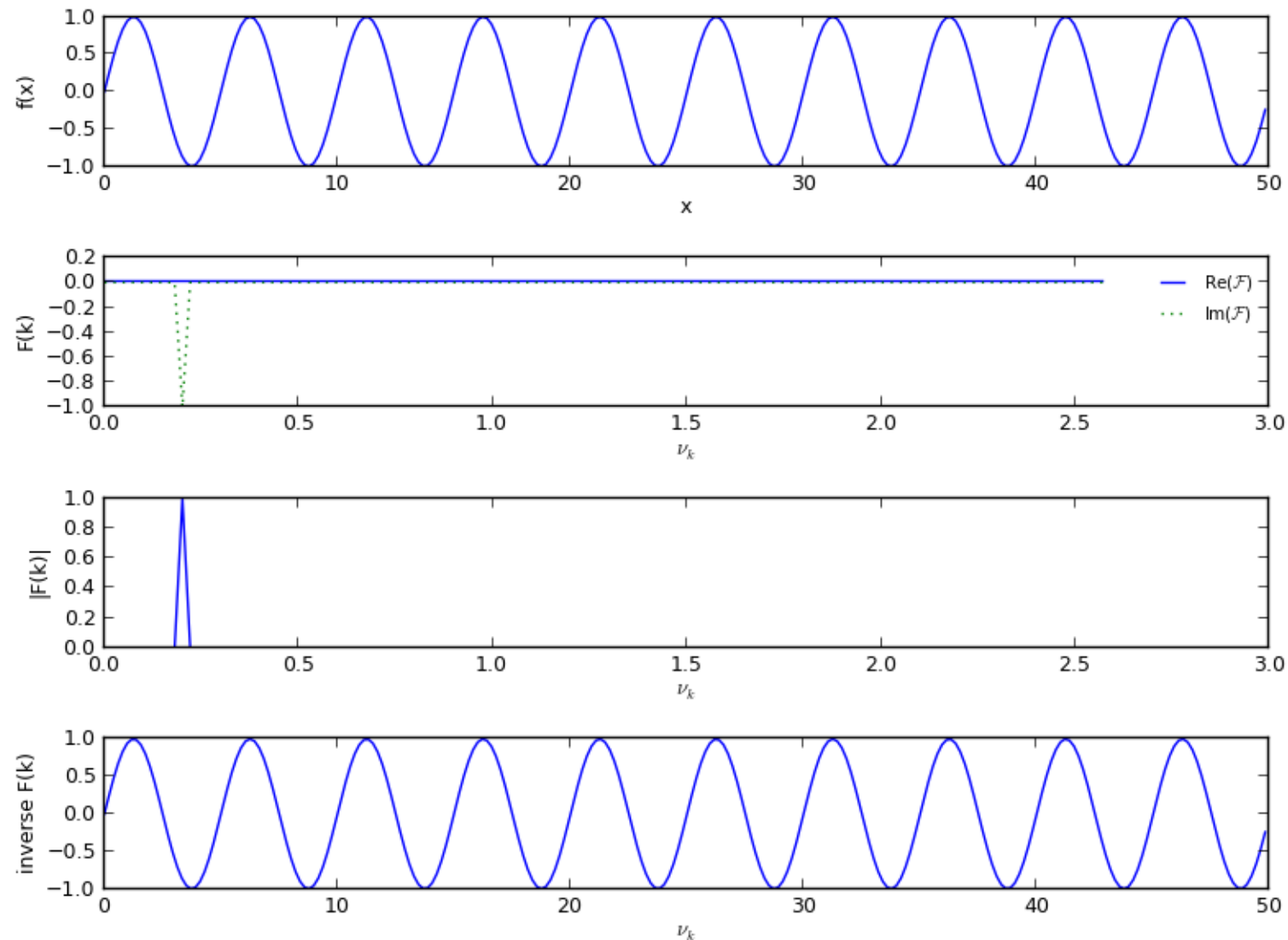
# Gotyas...

- Be careful about how you define the points

  - The FFT considers the data without any spatial coordinates—it just considers distance in terms of the number of points

  - Using the NumPy `linspace()` routine puts a point at both the start and end of the interval

    - e.g., `np.linspace(0, 1, 5)` = `[ 0. 0.25 0.5 0.75 1.]`
    - The FFT routine treats the first and last point as distinct
    - If you define $\sin(2\pi\,x)$ on this data, the first and last point will be equivalent, and the FFT picks up an extra (non-periodic) signal

  - You should do `np.linspace(0, 1, 5, endpoint=False)` instead

# Gotyas...

FFT with linspace putting a point at both endpoints.

Note the blip in the real portion of the FFT.
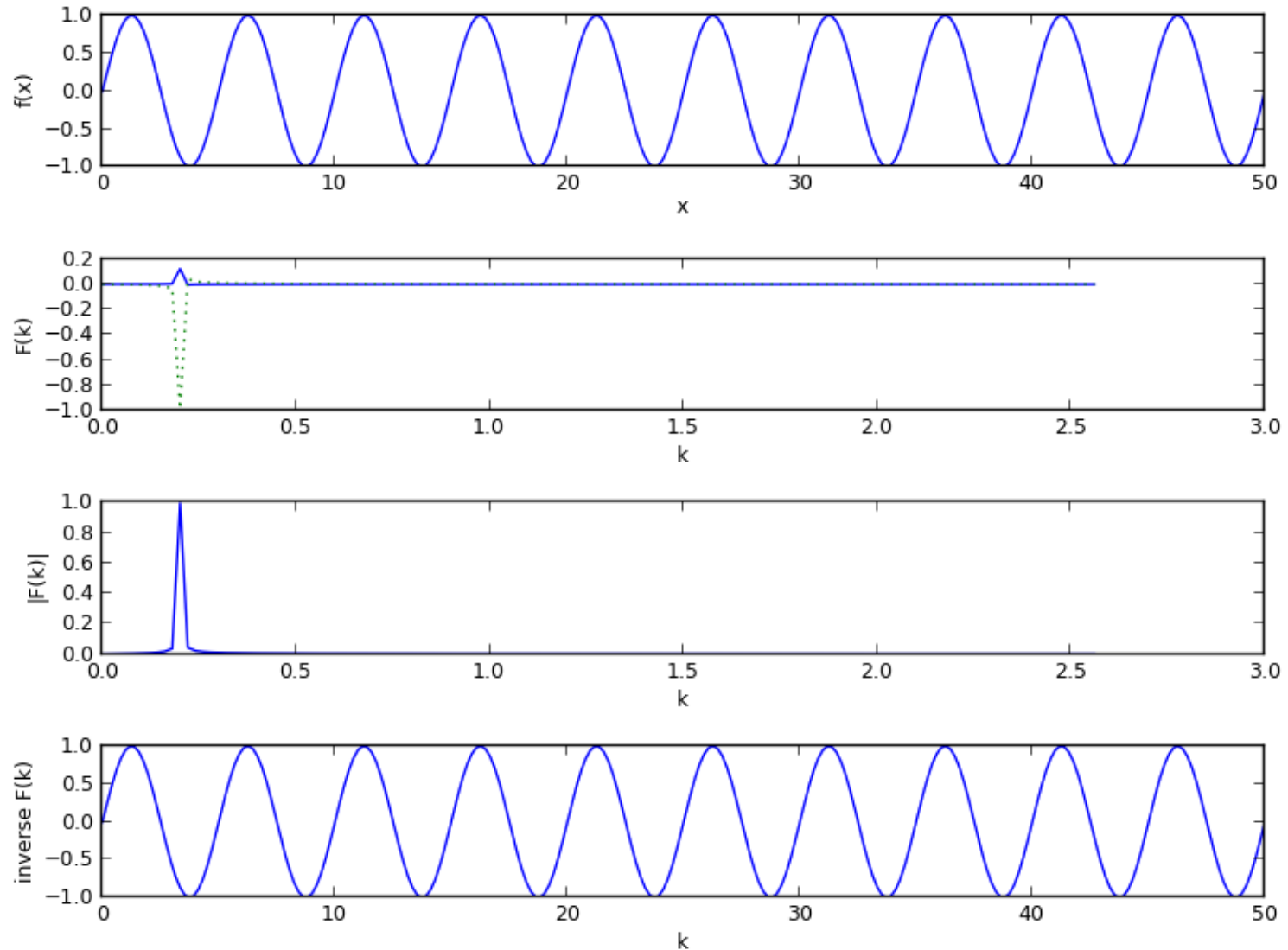
- And a cosine is "opposite" of the sine

Notice that the since cosine is even, the FFT is only non-zero in the real component.

Also note the normalizations
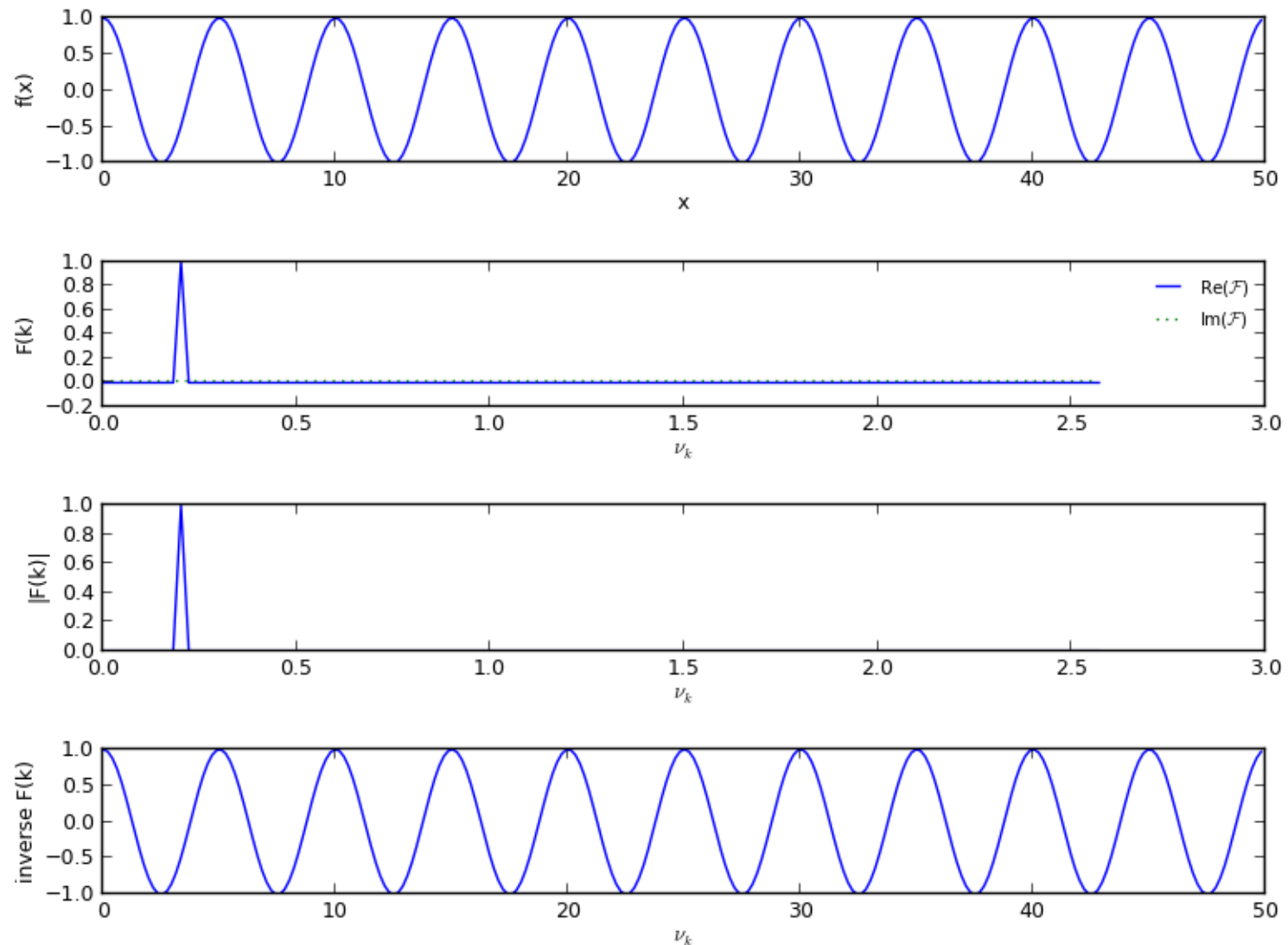
- Amplitude and phase.  Consider: $\sin(2\pi\, f_0\, x + \pi/4)$

- Phase:

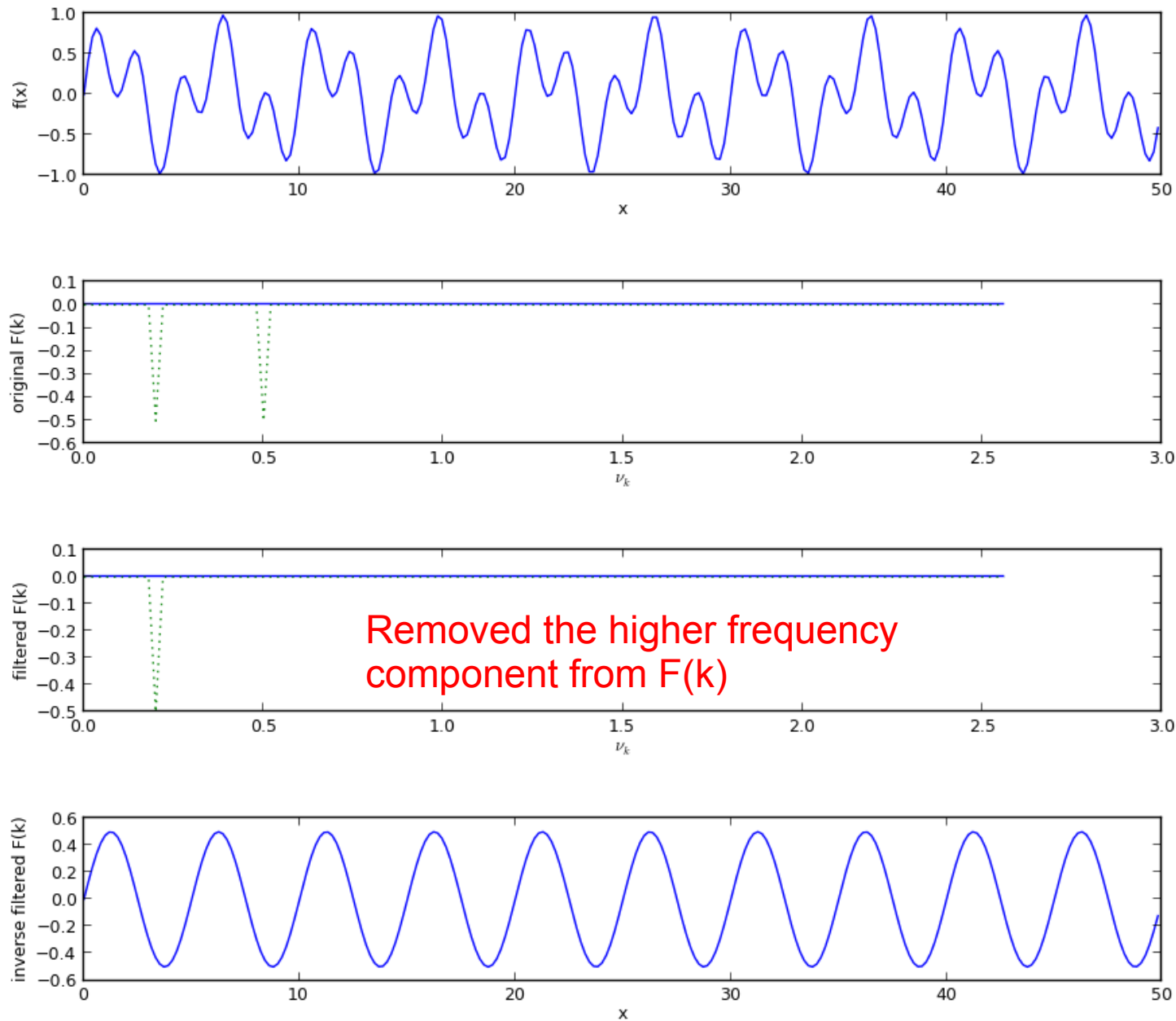$$\phi = \tan^{-1}\left(\frac{\mathrm{Im}(F_k)}{\mathrm{Re}(F_k)}\right)$$
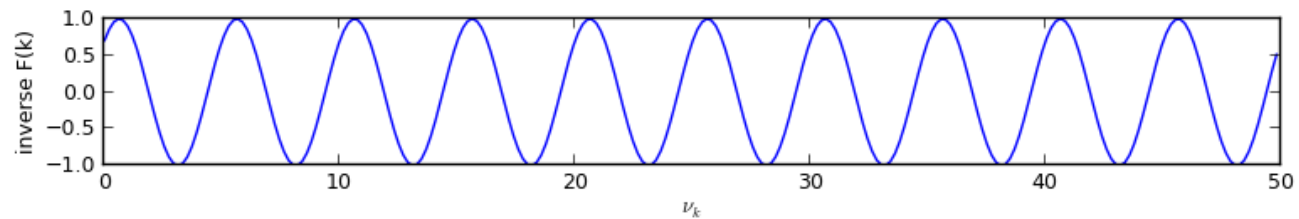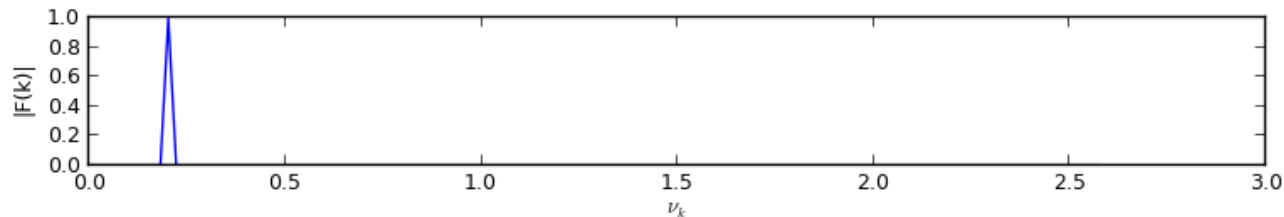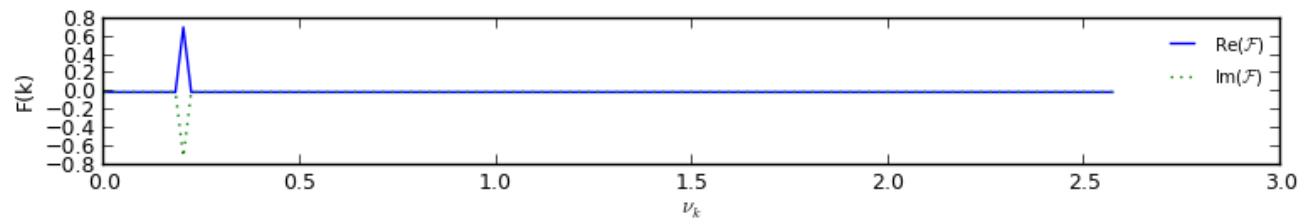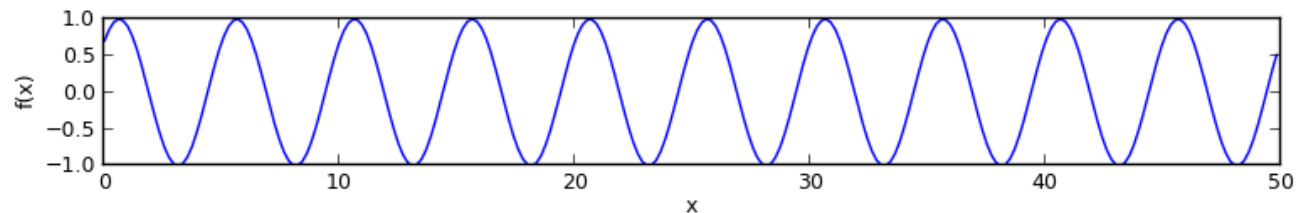
# Example: Filtering

- In frequency-space, we can filter out high or low frequency components.

- Consider:

$$f(x) = \frac{1}{2} \left[ \sin(2\pi f_0 x) + \sin(2\pi f_1 x) \right]$$

  – With $f_0 = 0.2$, $f_1 = 0.5$

# Example: Filtering



Removed the higher frequency component from F(k)

# Example: Time-Series Analysis

- Remember our pendulum problem from homework #2

  - Finite-amplitude system

  - Here's the solution with $\theta_0 = 25°$

50 periods integrated (using small-angle estimate of period)

12,568 points from 4th-order Runge-Kutta

- Power spectrum of the first period
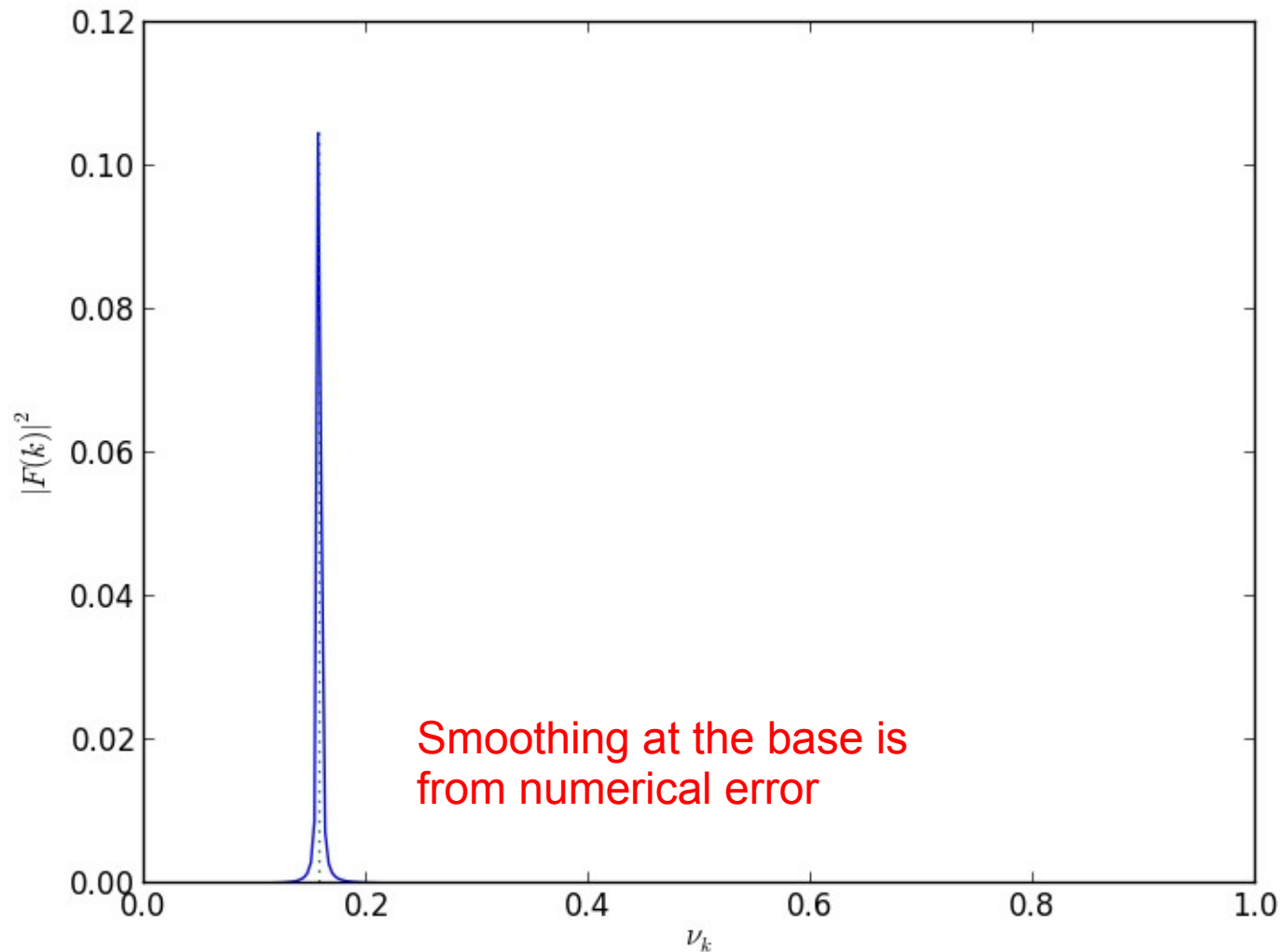  - Dotted line is analytic freq estimate: $f^{-1} = T \approx 2\pi \sqrt{\dfrac{L}{g}} \left( 1 + \dfrac{\theta_0^2}{16} \right)$
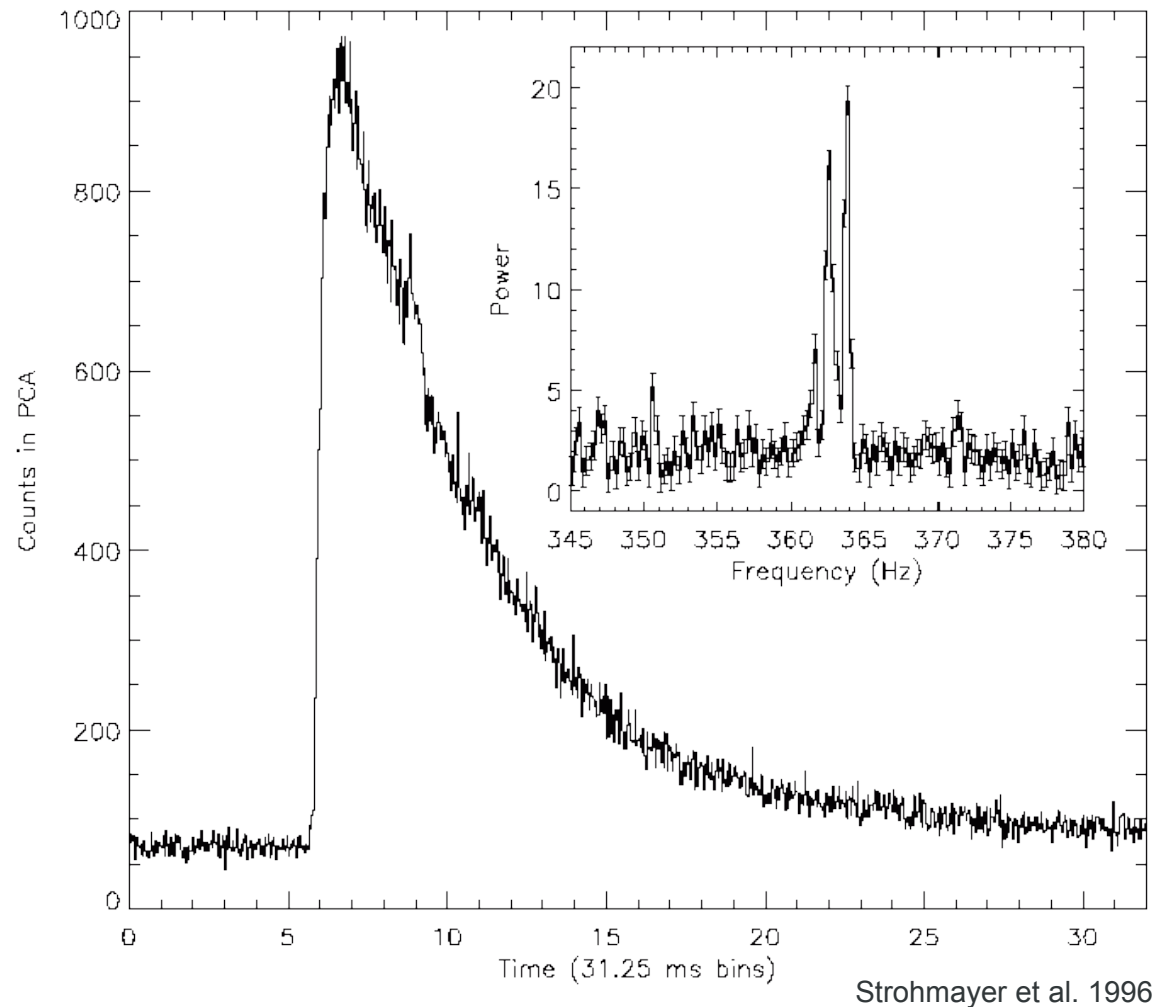
- Power spectrum of the first 5 periods
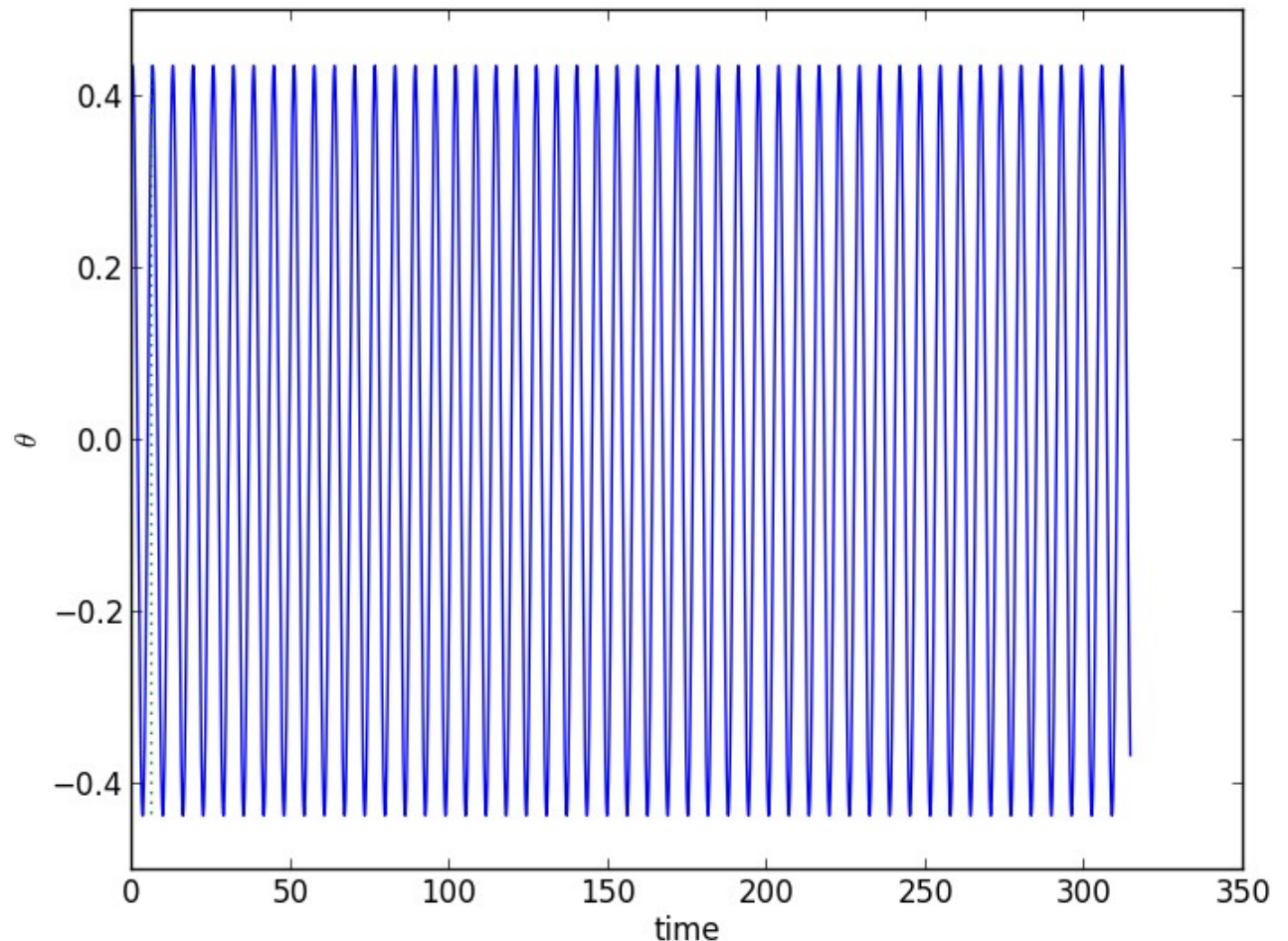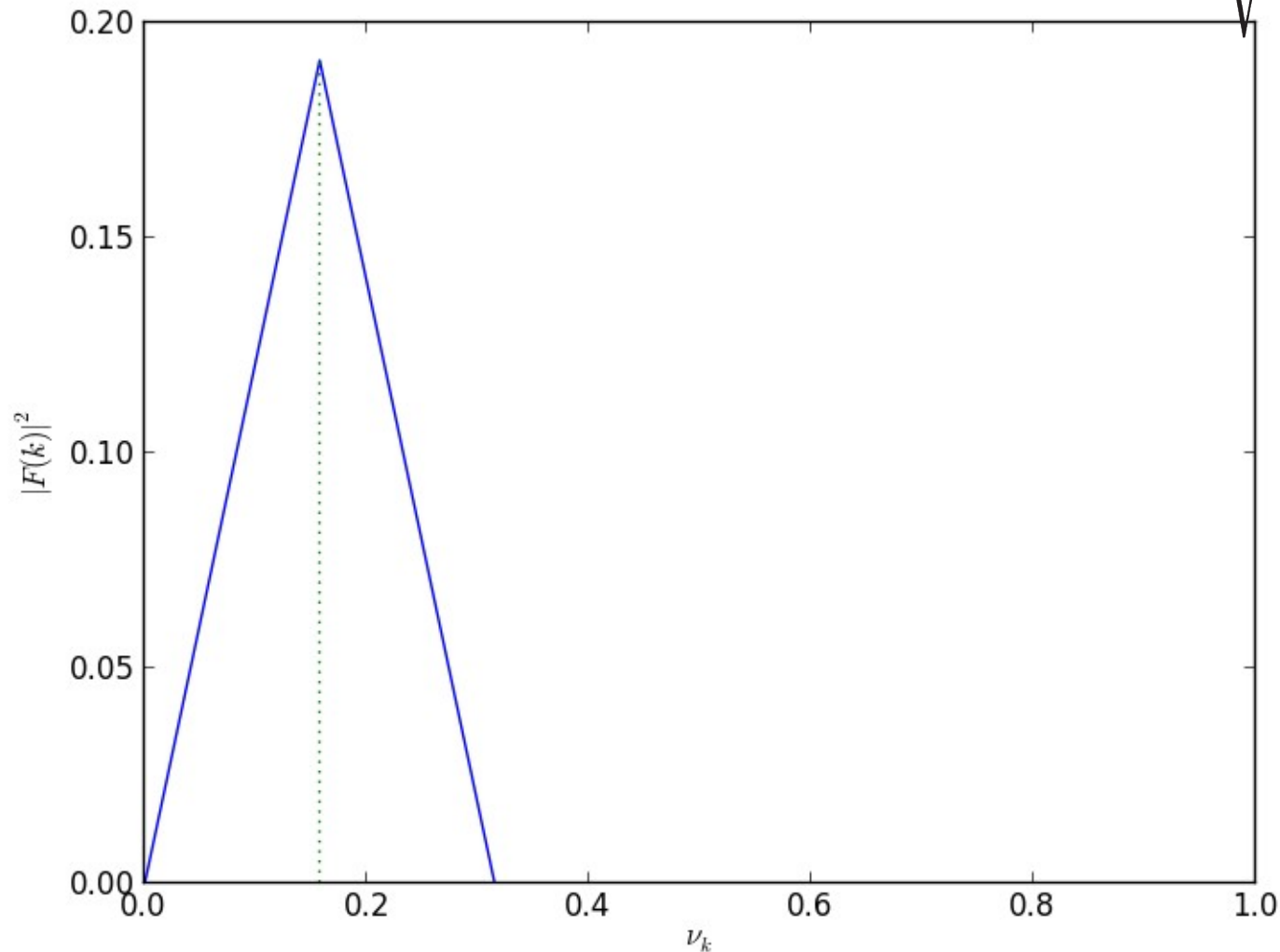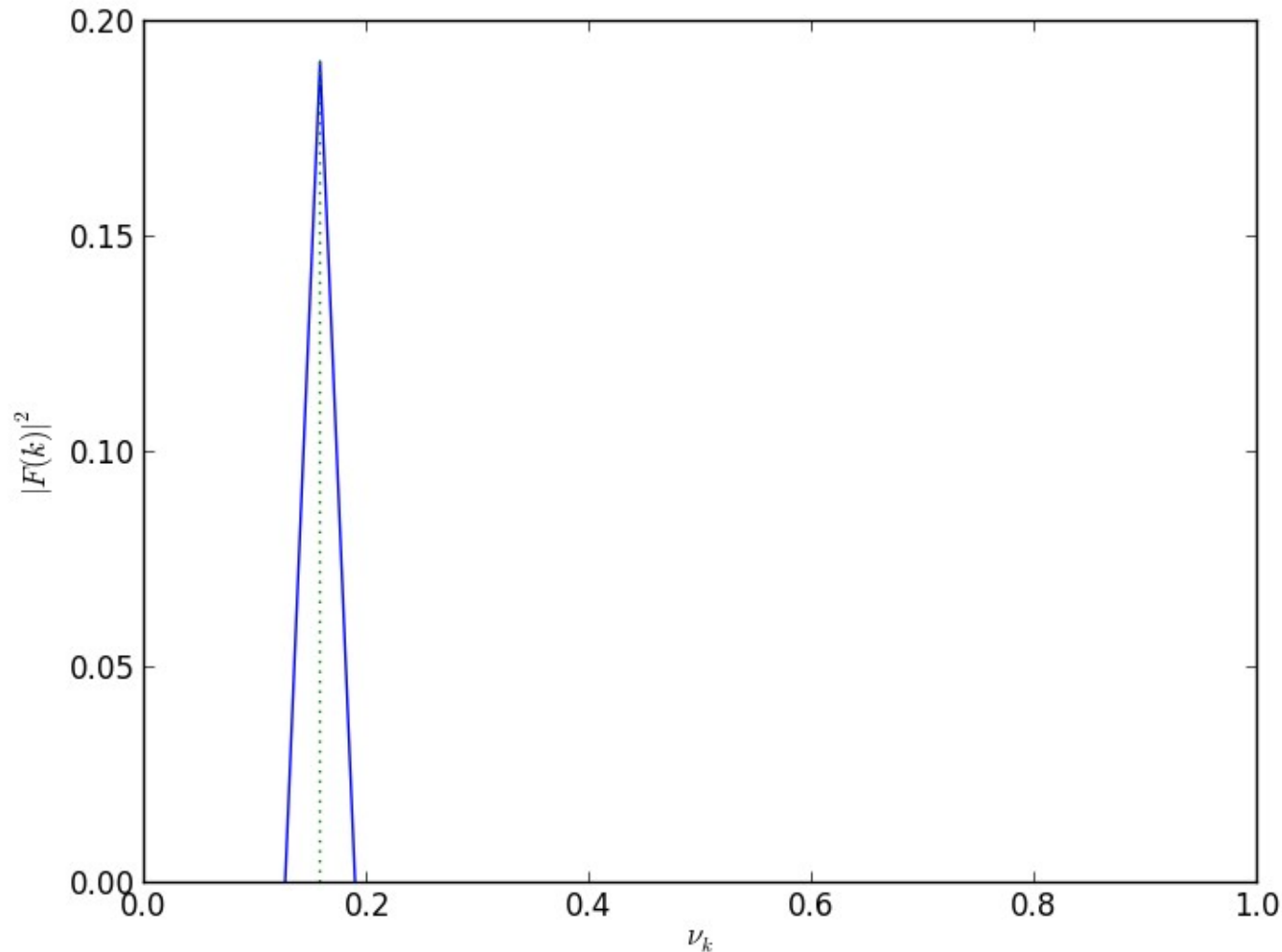
# Example: Time-Series Analysis

- Power spectrum of the all 50 periods



Smoothing at the base is from numerical error

- Here's an astronomical example from an X-ray burst time-series



Strohmayer et al. 1996

# Multidimensional FFT

- Multidimensional FFTs decompose into a sequence of one-dimensional FFTs

$$F_{k_x,k_y} = \sum_{m=0}^{N_x-1} \sum_{n=0}^{N_y-1} f_{mn} e^{-2\pi i(k_x m/N_x + k_y n/N_y)}$$

$$= \sum_{m=0}^{N_x-1} e^{-2\pi i k_x m/N_x} \underbrace{\sum_{n=0}^{N_y-1} f_{mn} e^{-2\pi i k_y n/N_y}}$$

This is the transform
in the y-direction

# Multidimensional FFT

- To visualize what is going on, we need to look at both the amplitude and the phase

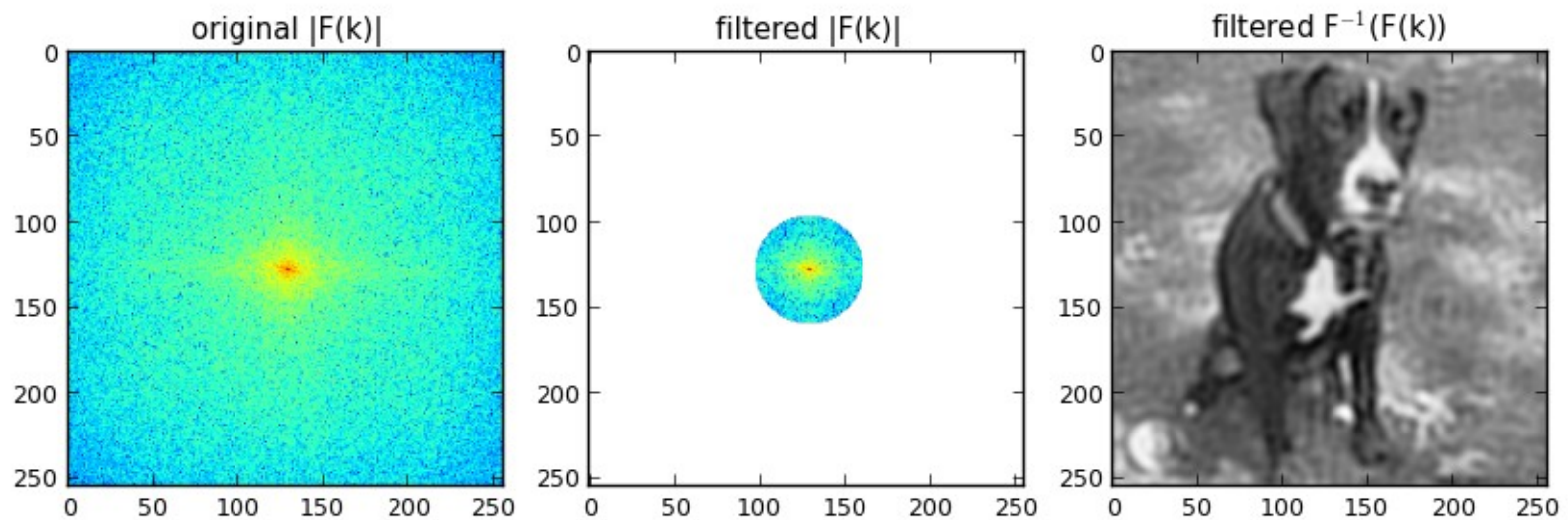  – Note that only 1 quadrant is significant, because our input was real-valued.
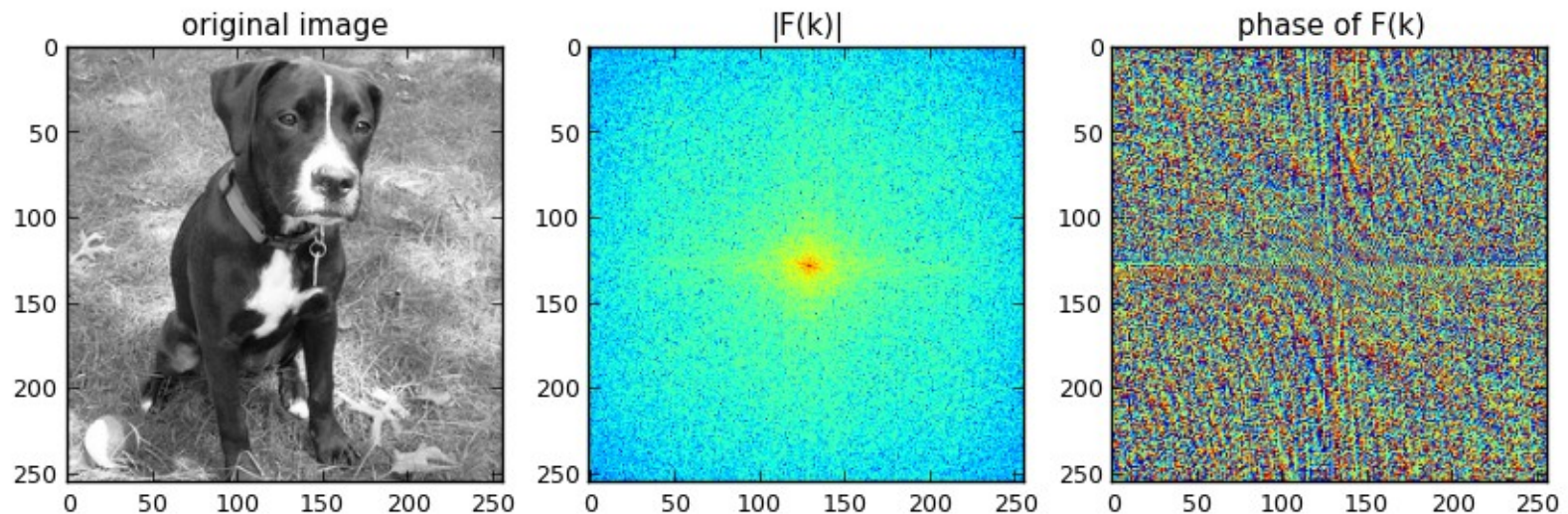
# Multidimensional FFT

- To understand what the magnitude and phase influence, here we scramble each of these in turn
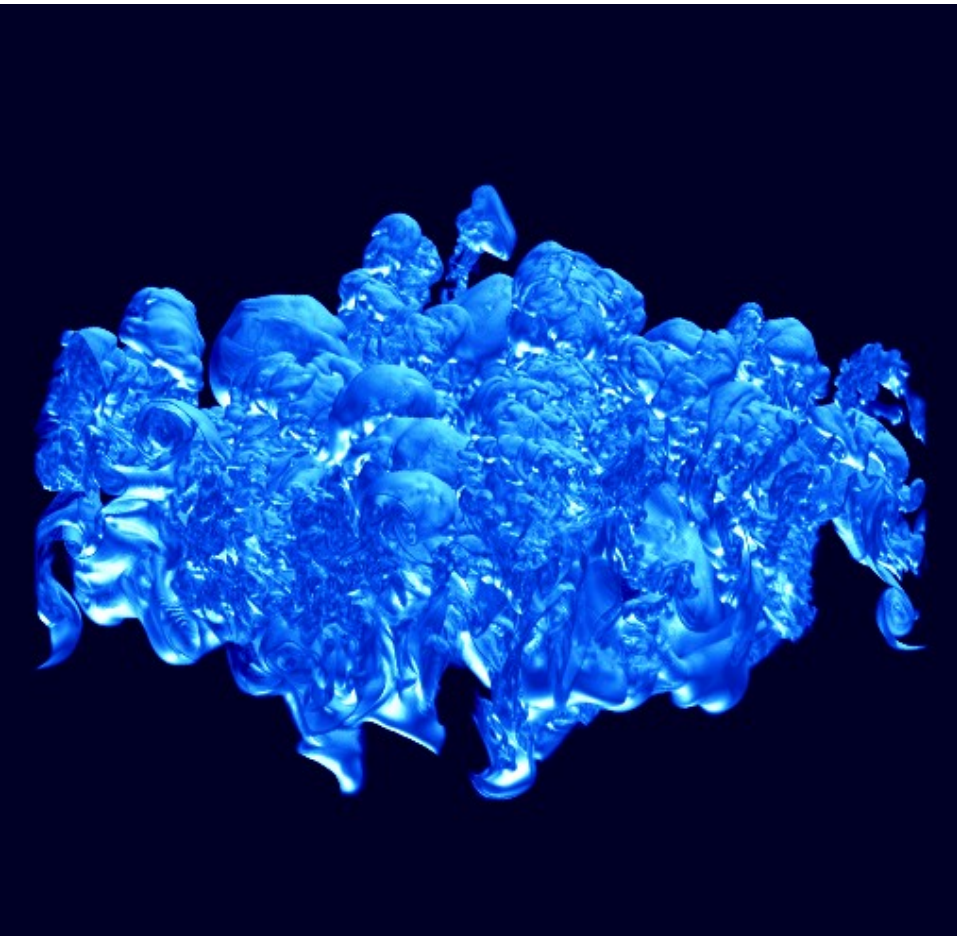


$F^{-1}(F(k))$ with scrampled amplitudes

$F^{-1}(F(k))$ with scrampled phases
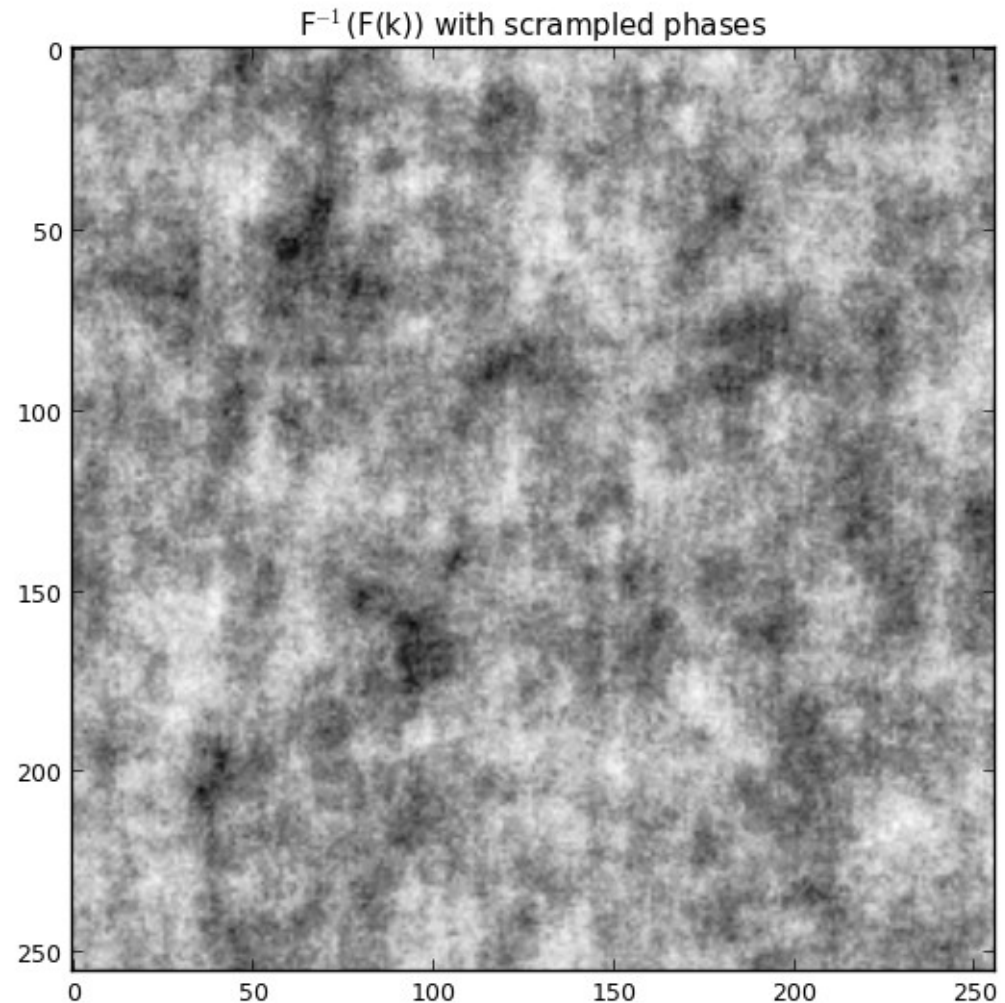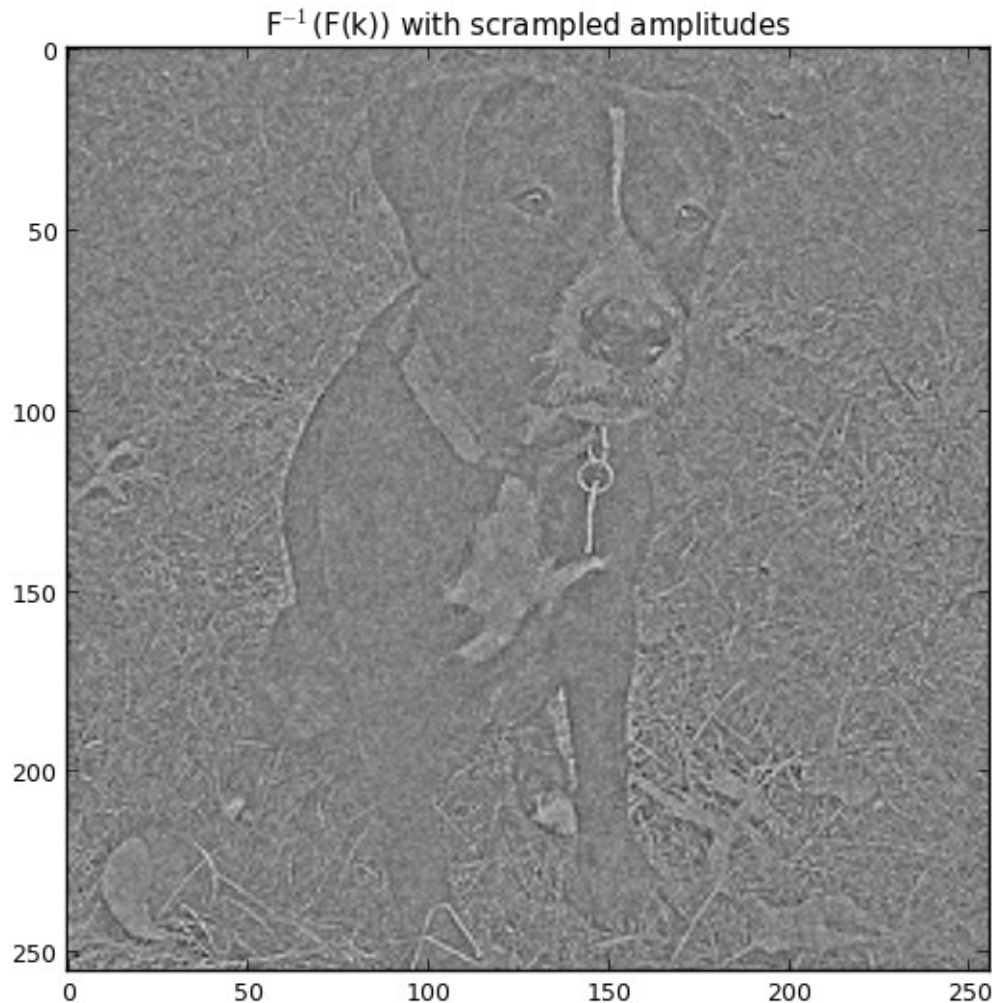
- In phase space we can filter out frequency components to do image processing



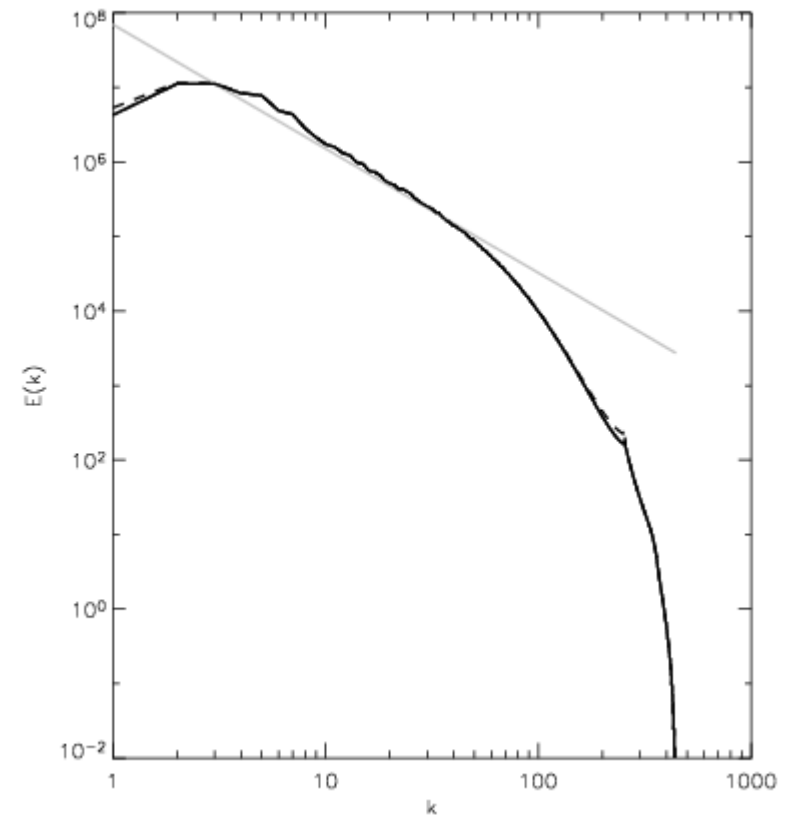original |F(k)|      filtered |F(k)|      filtered $F^{-1}(F(k))$

# Example: Turbulence

- Power spectrum of the velocity field is used to understand the turbulent energy cascade



(Zingale et al. 2004)

$$E(K) = \int_{K=|k|} dk[\hat{u}(k)^2 + \hat{v}(k)^2 + \hat{w}(k)^2]$$

$$k^2 = k_x^2 + k_y^2 + k_z^2$$

# Beyond Data Analysis

- So far we've focused on using the FFT for data analysis

- We can also use it directly for solving (some) PDEs.

- Consider the Poisson equation:

$$\frac{d^2\phi_n}{dx^2} = f_n$$

  - Express things in terms of the transforms

$$\phi(x) = \int \Phi(k)e^{-2\pi ikx}dk$$

$$f(x) = \int F(k)e^{-2\pi ikx}dk$$

# Beyond Data Analysis

- Easy to differentiate:

$$\frac{d^2\phi(x)}{dx^2} = -4\pi^2 k^2 \int \Phi(k) e^{2\pi i k x} dk$$

- Then:

$$-4\pi^2 k^2 \Phi(k) = F(k)$$

  – Easy to solve:

$$\Phi(k) = -\frac{F(k)}{4\pi^2 k^2}$$

- Solve algebraically in Fourier space and then transform back

  – Only works for certain boundary conditions