Computational Physics
Lecture 4

sieversj@ukzn.ac.za

git clone https://github.com/ukzncompphys/lecture3.git

Tutorial Problems

- How can you list all files in a directory, with the most recent one displayed last? (5)
- How can you display the first 25 lines of a file? How can you display just lines 16 through 25? (5)
- How can you search for all commands relating to python? Hint - it will be a use of "man". (5)
- Make a new text file with your name, email address, what you're working on for your honours project, plus a few things you'd like to learn in the course. Initialize a git repository and commit this file. (5)
- Put your answers to the other tutorial questions into another text file. Add this to the repository also. (5)
- Make a github account and push the repository onto github. Email me your github name so I can have a look at your file/answers. (10)

Fourier Transforms

- Functions can be represented in many different ways
- We normally use "real" space - $f(x)$
- Generally, arbitrarily many transforms exist to represent functions in different spaces - $F(y)=Af(x)$ for some matrix A and some new variable *y*. Iff A is invertible, $f(x)=A^{-1}F(y)$
- One important basis nature has picked out is complex exponentials/sines and cosines. Fundamental across physics, particularly quantum mechanics.

Fundamental Definition

- $F(k)=\int f(x)*\exp(2\pi ikx)dx$ (where $k=1/\omega$)
- Integral gets rid of *x*, replaces with *k*. New function has amplitude and phase as a function of *k*.
- Quantum mechanics - de Broglie says $p = \hbar k$. So, Fourier transform position to get momentum.
- Fourier transform electric field $E(t)$ to get frequency spectrum.
- Fourier transform to get fast correlations, convolutions, many other things.

DFT (Discrete FT)

- Computers don’t to continuous. Not enough RAM for starters…
- Function exists over finite range in x at finite number of points.
- If input function has *n* points, output can only have *n* k’s.
- Gives rise to discrete Fourier Transform (DFT)
- $F(k)=\sum f(x)\exp(2\pi ikx/N)$ for N points and $0\leq k<N$
- What would DFT of $f(0)=1$, otherwise $f(x)=0$ look like?
- What would DFT of $f(x)=1$ look like?
- DFTs have subtle behaviours not seen in continuous, infinite FTs.

Inverse

- One way to think about DFT is as a matrix multiply.
- $F(k)=Af$, $A_{mn}=\exp(2\pi i\ mn/N)$
- But look - $A_{mn}==A_{nm}$, so matrix is symmetrix.
- Also, columns are orthogonal under conjugation:
  $\sum\exp(-2\pi ikx)\exp(2\pi ik'x)=\sum\exp(2\pi i(k'-k)x)$. N if $k'==k$, otherwise 0.
- So, $A^{-1}=1/N*\mathrm{conj}(A)$. $\mathrm{IFT}=1/N\ \sum F(k)\exp(-2\pi ikx)$.
- Get back to where we started by just doing another DFT with a sign flip, then divide by # of data points.
- Alternative: divide by $\sqrt{(N)}$ in both DFT and IFT, (not standard computationally)

Numpy Complex

```python
import numpy
def exp_prod(m,n,N):
    #define imaginary unity
    J=numpy.complex(0,1)
    #now rest of code is just like for real numbers
    x=numpy.arange(0.0,N)*2*J*numpy.pi/N
    return numpy.sum(numpy.exp(-1*x*m)*numpy.exp(x*n))
if __name__=="__main__":
    print exp_prod(0,0,8)
    print exp_prod(2,4,8)
    print exp_prod(3,3,8)
    print exp_prod(0,7,8)
```

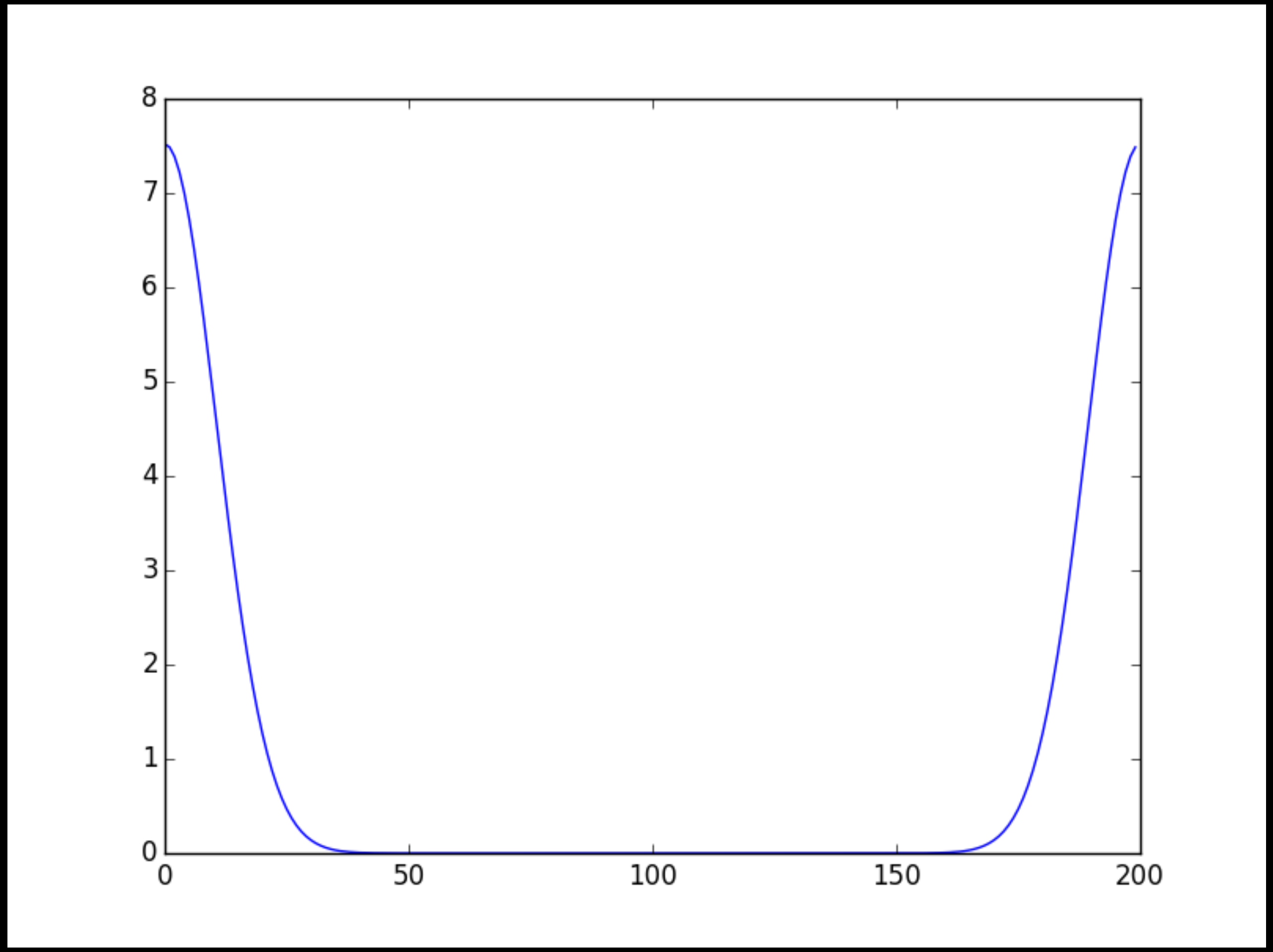
```
Jonathans-MacBook-Pro:lecture4 sievers$ python dft_columns.py
(8+0j)
(-4.28626379702e-16+4.4408920985e-16j)
(8+0j)
(3.44169137634e-15-1.11022302463e-15j)
Jonathans-MacBook-Pro:lecture4 sievers$
```

- Let's check orthogonality, need complex #'s.
- numpy.complex(re,im) will make a complex #
- numpy functions usually defined for complex #'s.

DFTs with Numpy

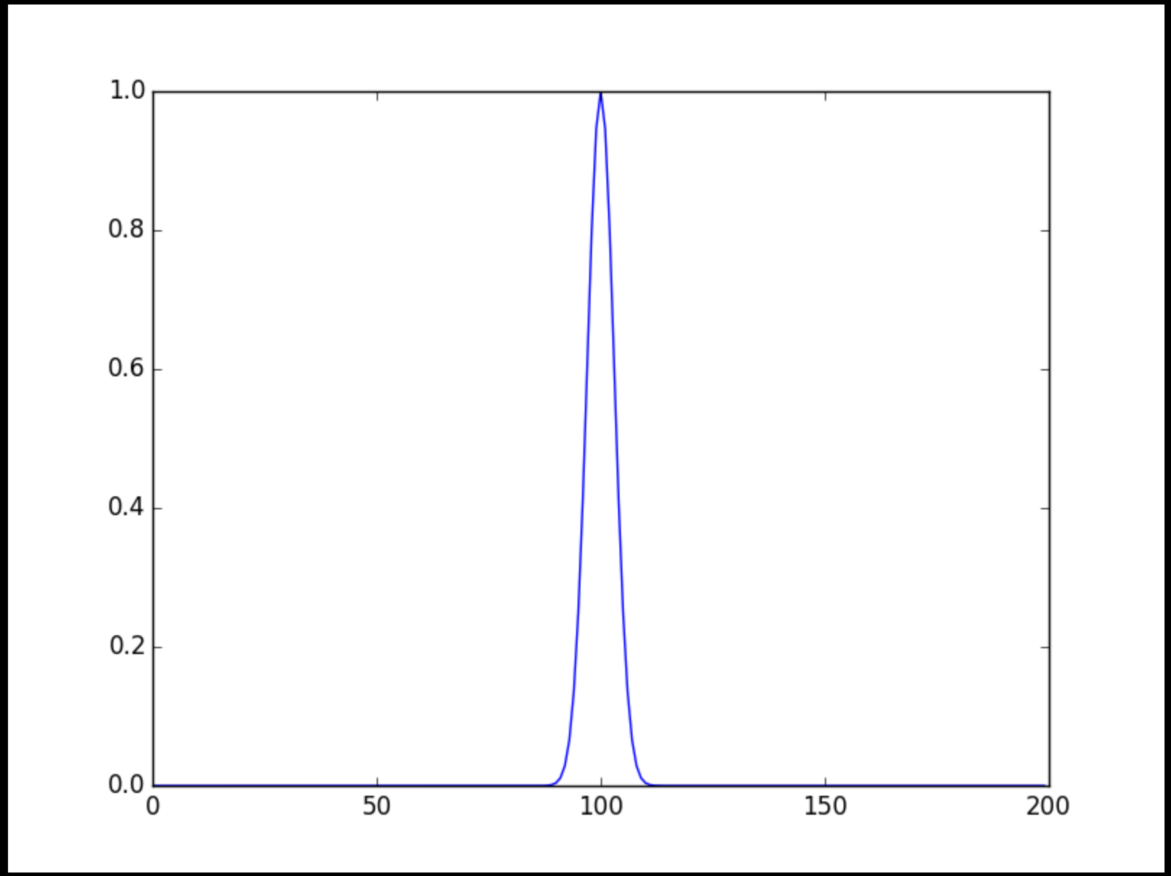
- Numpy has many Fourier Transform operations
- (for reasons to be seen) they are called *Fast* Fourier Transforms - FFT is one way of implementing DFTs.
- FFT's live in a submodule of numpy called FFT
- xft=numpy.fft.fft(x) takes DFT
- x=numpy.fft.ifft(x) takes inverse DFT
- Numpy normalizes such that f==fft(ifft(f))==ifft(fft(f))

DFT in Action




```
import numpy
from matplotlib import pyplot as plt

x=numpy.arange(-10,10,0.1)
y=numpy.exp(-0.5*x**2/(0.3**2))
yft=numpy.fft.fft(y)
plt.plot(numpy.abs(yft))
plt.savefig('gauss_dft')
plt.show()
```



- Left: input Gaussian
- Top: DFT of the Gaussian

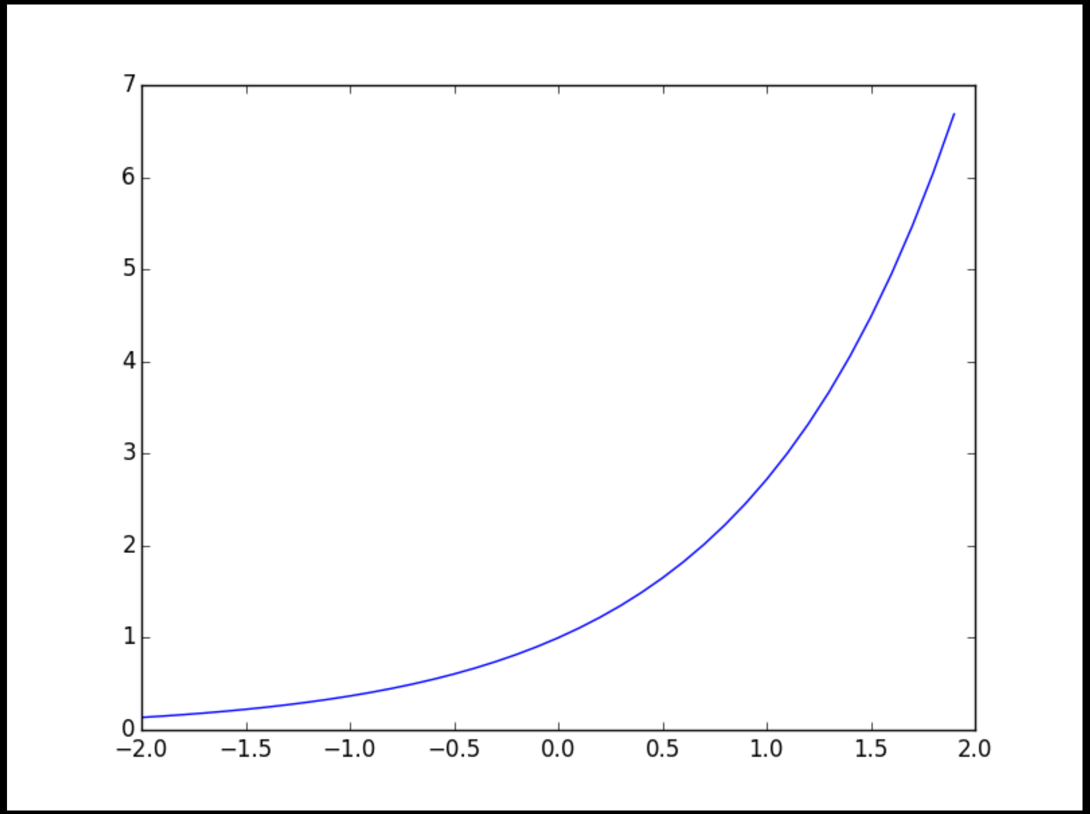
Periodicity

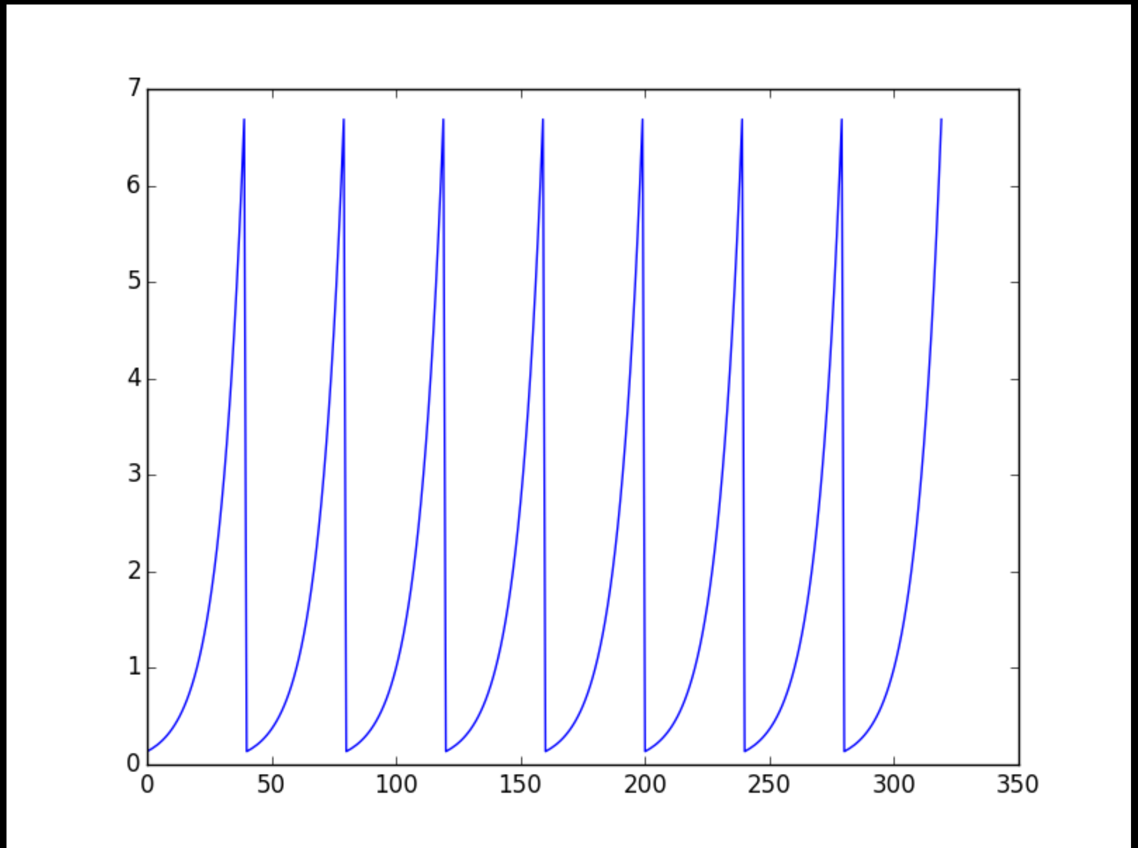
- $f(x)=\sum F(k)\exp(2\pi ikx/N)$
- What is $f(x+N)$? $\sum F(k)\exp(2\pi ik(x+N)/N)$
- $=\sum F(k)\exp(2\pi ik)\exp(2\pi ikx/N)$.
- $\exp(2\pi ik)=1$ for integer k, so $f(x+N)==f(x)$
- DFT's are periodic - they just repeat themselves ad infinitum.

Periodicity

```python
import numpy
from matplotlib import pyplot as plt

x=numpy.arange(-2,2,0.1)
y=numpy.exp(x)
plt.plot(x,y)
plt.savefig('fft_exp')
plt.show()
yy=numpy.concatenate((y,y))
yy=numpy.concatenate((yy,yy))
yy=numpy.concatenate((yy,yy))
plt.plot(yy)
plt.savefig('fft_exp_repeating')
plt.show()
```





- You may think you're taking top transform. You're not - you're taking the bottom one.
- In particular, jumps from right edge to left will strongly affect DFT

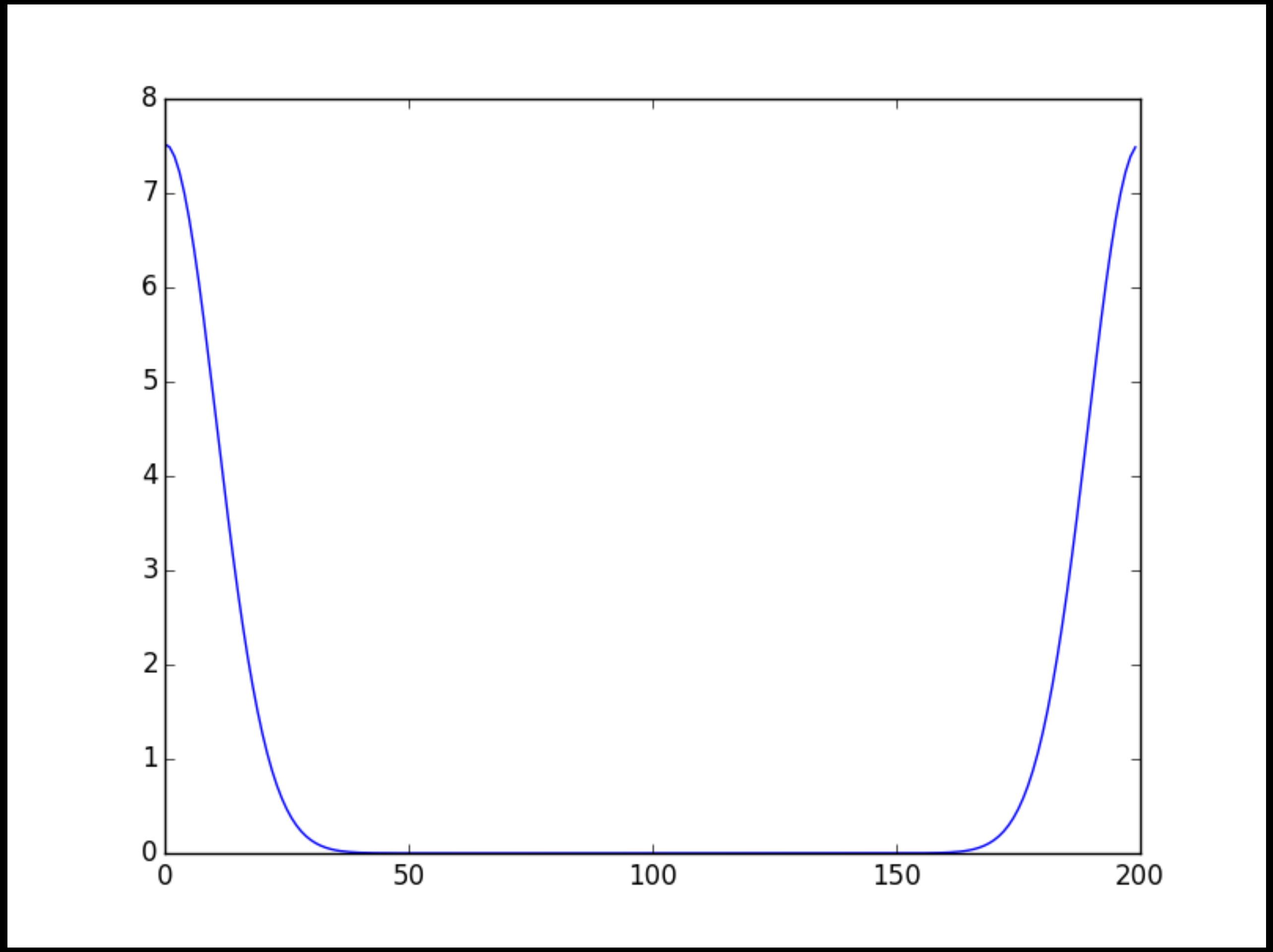
Aliasing

- $f(x) = \sum F(k) \exp(-2\pi ikx/N)$
- What if I had higher frequency, k>N? let $k^* = k - N$ (i.e. k* low freq.)
- $f(x) = \sum F(k) \exp(-2\pi i(k^*+N)x/N = \sum F(k)\exp(-2\pi ix)\exp(-2\pi ik^*x/N)$
- for x integer, middle term goes away: $\sum F(k^*+N)\exp(-2\pi ik^*x/N)$
- High frequencies behave exactly like low frequencies - power has been *aliased* into main frequencies of DFT.
- Always keep this in mind! Make sure samples are fine enough to prevent aliasing.

Negative Frequencies

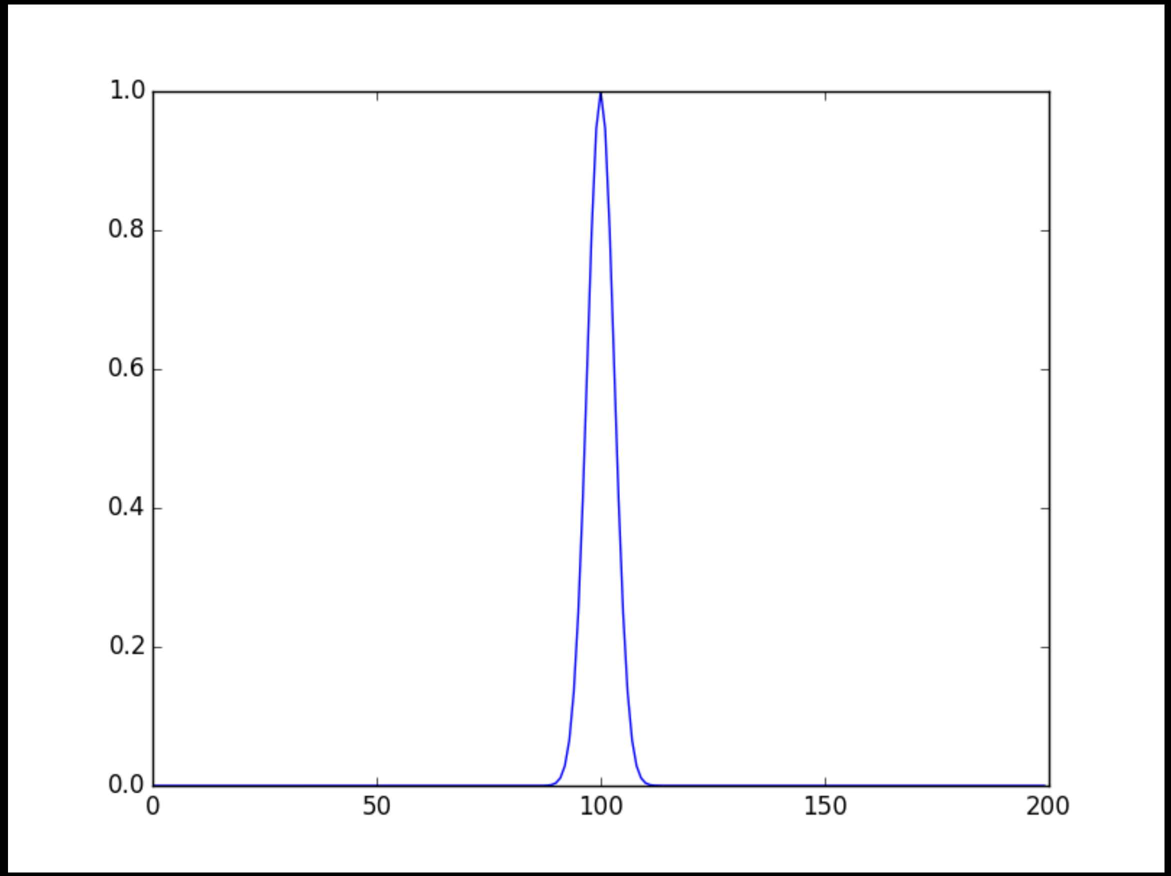
- All frequencies that are N apart behave identically
- DFT has frequencies up to (N-1).
- Frequency (N-1) equivalent to frequency (-1). You will do better to think of DFT as giving frequencies (-N/2,N/2) than frequencies (0,N-1)
- *Sampling theorem*: if function is band-limited - highest frequency is ν - then I get full information if I sample *twice* per frequency, $dt=1/(2\nu)$. Factor of 2 comes from aliasing.

DFT in Action, Redux




```
import numpy
from matplotlib import pyplot as plt

x=numpy.arange(-10,10,0.1)
y=numpy.exp(-0.5*x**2/(0.3**2))
yft=numpy.fft.fft(y)
plt.plot(numpy.abs(yft))
plt.savefig('gauss_dft')
plt.show()
```



- FFT makes more sense now - negative frequencies have been aliased to high frequency.

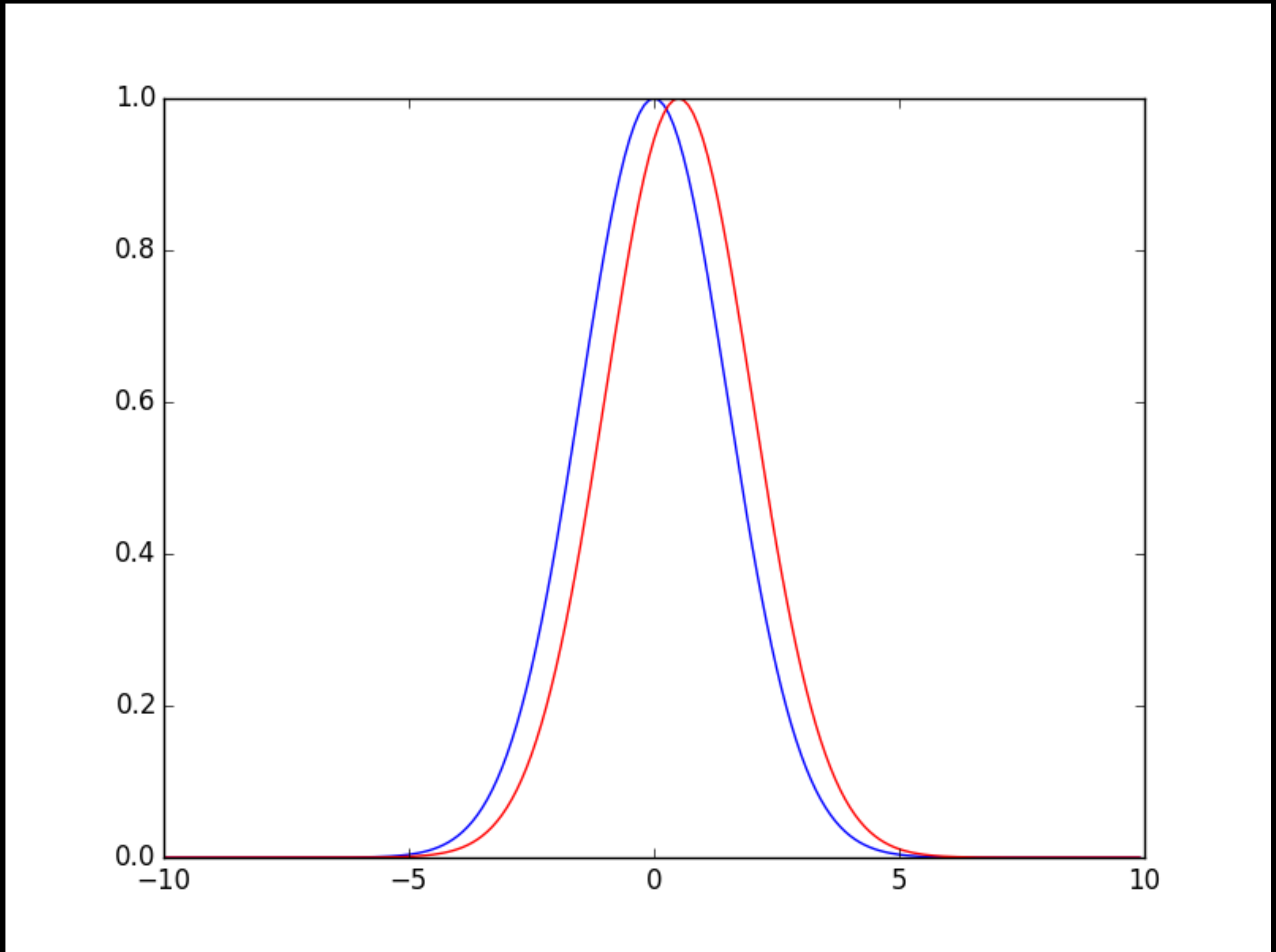
Flipping

- What is DFT of $f(-x)$?
- $\sum f(-x)\exp(2\pi ikx/N)$, $x^*=-x$, $\sum f(x^*)\exp(2\pi ik(-x)/N)$
- $DFT(f(-x))=\sum f(x)\exp(-2\pi ikx/N) = conj(F(k))$

Shifting

- What is FFT(x+dx)? $\sum f(x+dx)\exp(2\pi ikx/N)$.
- $x^*=x+dx$: $F(k)=\sum f(x^*)\exp(2\pi ik(x^*-dx)/N)$
- $F(k)=\exp(-2\pi ikdx/N)\sum f(x^*)\exp(2\pi ikx^*/N)$
- So, just apply a phase gradient to the DFT to shift in x

Shifting Example




```
import numpy
from matplotlib import pyplot as plt

x=numpy.arange(-10,10,0.1)
y=numpy.exp(-0.5*x**2/(1.5**2))
N=x.size
kvec=numpy.arange(N)
yft=numpy.fft.fft(y)
J=numpy.complex(0,1)
dx=5.0;
yft_new=yft*numpy.exp(-2*numpy.pi*J*kvec*dx/N)
y_new=numpy.real(numpy.fft.ifft(yft_new))
plt.plot(x,y)
plt.plot(x,y_new,'r')
plt.savefig('shifted_gaussian')
plt.show()
```

Real Data Symmetry

- If I know F(k), what is F(N-k) is f(x) is real?
- F(N-k)=F(-k) (from alias theorem)
- $F(-k)=\sum f(x)\exp(2\pi i(-k)x/N$. let $x^*=-x$
- $F(-k)=\sum f(-x^*)\exp(2\pi i k x^* N)$ = conj(F(k)) by flipping
- So, if f(x) is real, F(k)=conj(F(N-k))
- If N even, F(N/2)=conj(F(N/2)), so F(N/2) must be real.

```
>>> x=numpy.random.randn(8)
>>> xft=numpy.fft.fft(x)
>>> for xx in xft:
...   print xx
...
(-4.53568815727+0j)
(-0.174046761579+2.08827239558j)
(2.15348308858+2.32162497273j)
(-0.423040513854-3.72126858798j)
(2.75685372591+0j)
(-0.423040513853+3.72126858798j)
(2.15348308858-2.32162497273j)
(-0.174046761579-2.08827239558j)
>>>
```

Convolution Theorem

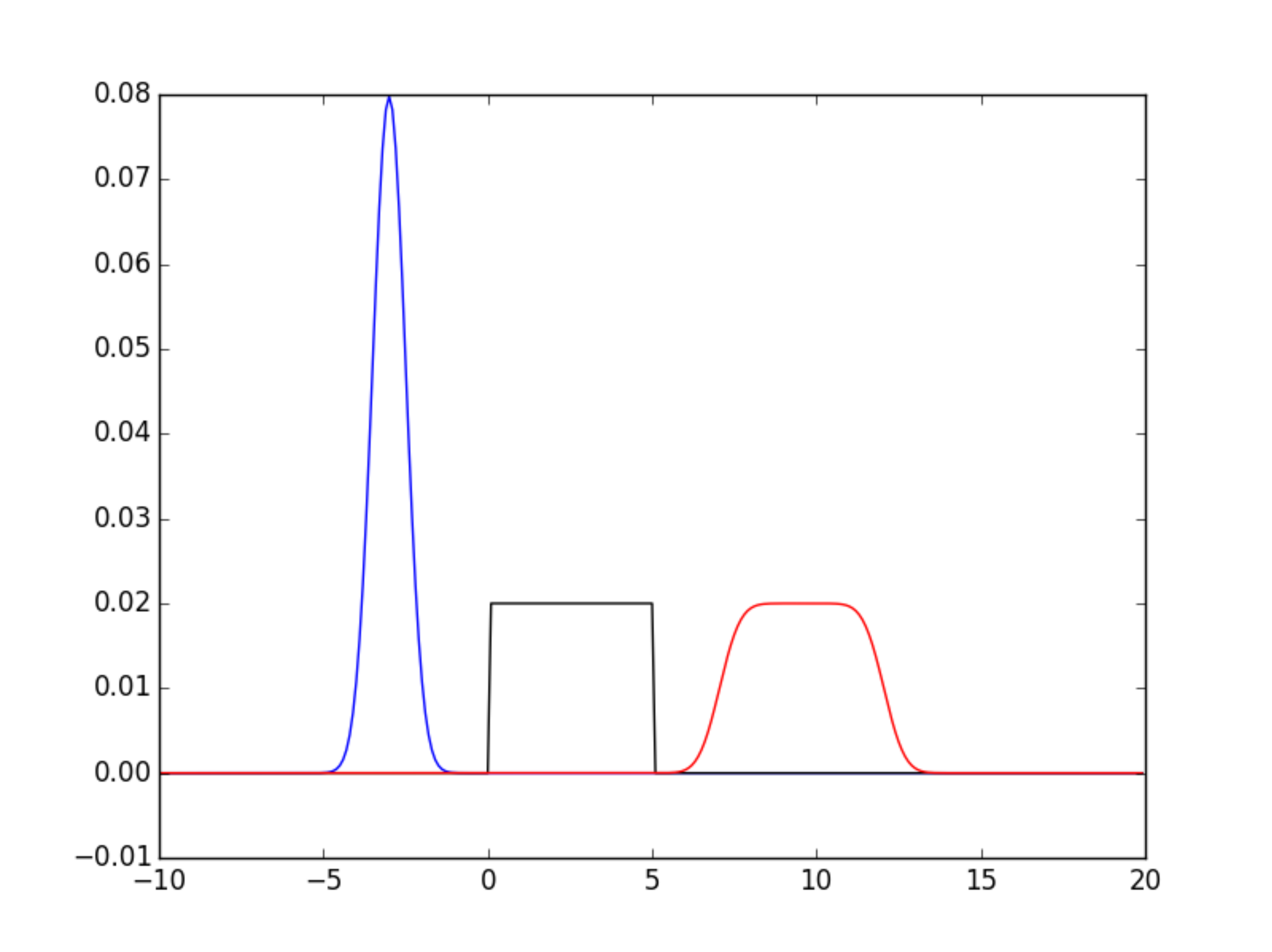
- Convolution defined to be $conv(y)=f\otimes g==\int f(x)g(y-x)dx$
- $\sum_x \sum F(k)\exp(2\pi ikx) \sum conj(G(k'))\exp(2\pi ik'x)\exp(2\pi ik'y/N)$
- Reorder sum: $\sum\sum F(k)conj(G(k'))\exp(2\pi ik'y/N)\sum_x \exp(2\pi i(k+k')x)$
- equals zero unless k'==-k. Cancels one sum, conjugates G
- $f\otimes g=\sum F(k)G(k)\exp(-2\pi iky/N)=ift(dft(f)*dft(g))$
- So, to convolve two functions, multiply their DFTs and the IFT

Convolution Example

```
from numpy import arange,exp,real
from numpy.fft import fft,ifft
from matplotlib import pyplot as plt
def conv(f,g):
    ft1=fft(f)
    ft2=fft(g)
    return real(ifft(ft1*ft2))

x=arange(-10,20,0.1)
f=exp(-0.5*(x+3)**2/0.5**2)
g=0*x;g[ (x>0)&(x<5)]=1
g=g/g.sum()
f=f/f.sum()
h=conv(f,g)

plt.plot(x,f,'b')
plt.plot(x,g,'k')
plt.plot(x,h,'r')
plt.savefig('convolved')
plt.show()
```



Fast Fourier Transform

- How many operations does a DFT take?
- Have an N by N matrix operating on a vector of length N - clearly N^2 operations, right?
- Nope! Otherwise we'd never use them. What's actually going on?
- Note DFT=$\sum f(x)\exp(2\pi ikx/N) = \sum f_{even}(x)\exp(2\pi ik(2x)/N)+\sum f_{odd}(x)\exp(2\pi ik(2x+1)/N)$
- $=F_{even}+\exp(2\pi ik/N)F_{odd}$. Let $W_k=\exp(2\pi ik/N)$
- if $k>N/2$, then $k^*=k-N/2$ and DFT=$F_{even}+\exp(2\pi ik^*/N+i\pi)F_{odd}=F_{even}-W_kF_{odd}$.

FFT cont'd

- So $F(k)=F_{even}(k)+W_kF_{odd}(k)$ $(k<N/2)$ or $F_{even}(k)-W_kF_{odd}(k)$ $(k>=N/2)$
- So, can get *all* the frequencies if I have 2 half-length FFTs.
- Well, just do the same thing again. FFT of a single element is itself.
- This algorithm works for arrays whose length is a power of 2
- Popularized by Cooley/Tukey in early computer days. Later found to go back to Gauss in 1805. Changes computational work from n^2 to *nlogn*.

Sample FFT

- Routine uses *recursion* - function calls itself. Recursion can be very powerful, but also easy to goof.
- numpy.concatenate will combine arrays - note that they have to be passed in as a tuple, hence the extra set of parenthesis
- Modern FFT routines deal with arbitrary length arrays. Fastest Fourier Transform in the West (FFTW) standard packaged - usually used by numpy.

```
from numpy import concatenate,exp,pi,arange,complex
def myfft(vec):
    n=vec.size
    #FFT of length 1 is itself, so quit
    if n==1:
        return vec
    #pull out even and odd parts of the data
    myeven=vec[0::2]
    myodd=vec[1::2]

    nn=n/2;
    j=complex(0,1)
    #get the phase factors
    twid=exp(-2*pi*j*arange(0,nn)/n)

    #get the dfts of the even and odd parts
    eft=myfft(myeven)
    oft=myfft(myodd)

    #Now that we have the partial dfts, combine them with
    #the phase factors to get the full DFT
    myans=concatenate((eft+twid*oft,eft-twid*oft))
    return myans
```

```
>>> import myft
>>> x=numpy.random.randn(32)
>>> xft1=numpy.fft.fft(x)
>>> xft2=myft.myfft(x)
>>> print numpy.sum(numpy.abs(xft1-xft2))
2.33937690259e-13
>>>
```

Tutorial Problems

- Write a function that will shift an array by an arbitrary amount using a convolution (yes, I know there are easier ways to do this). The function should take 2 arguments - an array, and an amount by which to shift the array. Plot a gaussian that started in the centre of the array shifted by half the array length. (10)

- The correlation function $f\bigstar g$ is $\int f(x)g(x+y)$. Through a similar proof, one can show $f\bigstar g$=ift(dft(f)*conj(dft(g))). Write a routine to take the correlation function of two arrays. Plot the correlation function of a Gaussian with itself. (10)

- Using the results of part 1 and part 2, write a routine to take the correlation function of a Gausian (shifted by an arbitrary amount) with itself. How does the correlation function depend on the shift? Does this surprise you? (10)

- The circulant (wrap-around) nature of the dft can sometimes be problematic. Write a routine to take the convolution of two arrays *without* any danger of wrapping around. You may wish to add zeros to the end of the input arrays. (10)

Tutorial Bonus Problem

- You have a sample code that calculates an FFT of an array whose length is a power of 2. Using that routine as a guideline, write an FFT routine that works on an array whose length is a power of 3 (e.g. 9, 27, 81). Verify that it gives the same answer as numpy.fft.fft (10)