any questions?

Imaging

- If we build telescopes, we usually want to take pictures of things…
- If I measure FT of sky, I can just IFT to get map, right?
- Well… There are usually gaps in UV coverage. If baselines sampled more densely than dish diameter, this might work.
- Not possible for high-resolution. To make a map of sky, we must fill in UV plane with some guess.
- Unobserved parts of UV could be anything! The art of imaging is sensibly filling in those unobserved areas.
- Remember - we understand very well how to predict data/measure χ^2 given a map of the sky. It's only inverse problem that is ill define.

CLEAN

- One standard way to do this is CLEAN. Pretend the sky is full of point sources.
- Make a dirty map - direct FT of visibilities. Requires putting visibilities on a grid - how fine does that grid need to be?
- Look for brightest peak (or peaks).
- Subtract from data. Repeat.
- Experience has shown that subtracting fraction of peak brightness works better in practice - say 30-50% of peak.
- When should we stop?

Bayesian

- General imaging problem is usually under constrained. We have to “make up” data to make an image.
- Amongst all the possible maps that agree with the data, *you* need to decide which one you think makes sense.
- Which brings us to the Reverend Bayes. $P(a|b)P(b)=P(b|a)P(a)$. $P(m|d)=P(d|m)P(m)/P(d)$ for data d and map m.
- Drop $P(d)$ because we already have the data. $P(d|m)$ is straightforward to calculate. Effectively, mapping problem is deciding on $P(m)$
- There are many ways to do this - multiscale clean where you fit Gaussians instead of sources, maximum entropy… Do think about what you’re looking at as it will guide choice of imaging.

Everyone is Bayesian.

- Some are just in denial…
- If I were to run an MCMC fitting a Gaussian, I could either use σ or σ^2 as one of my independent variables.
- Would I get $<\sigma>^2=<\sigma^2>$?
- No! $P(m|d)=P(d|m)P(m)$. In one case, probability of model is flat in σ, in the other case it is flat in σ^2.

Example Code

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

x=np.arange(-20,20,0.1)
s_true=1
y_true=5*np.exp(-0.5*x**2/s_true**2)
y=y_true+np.random.randn(len(x))


npt=5000
s_linear=np.linspace(0.1,4,npt)
chisq_linear=0*s_linear
for i in range(npt):
    pred=np.exp(-0.5*(x**2)/s_linear[i]**2)
    lhs=np.dot(pred,pred)
    rhs=np.dot(pred,y)
    amp=rhs/lhs
    delt=y-amp*pred
    chisq_linear[i]=np.sum(delt**2)

var=np.linspace(s_linear[0]**2,s_linear[-1]**2,npt)
chisq_var=0*var
for i in range(npt):
    pred=np.exp(-0.5*(x**2)/var[i])
    lhs=np.dot(pred,pred)
    rhs=np.dot(pred,y)
    amp=rhs/lhs
    delt=y-amp*pred
    chisq_var[i]=np.sum(delt**2)

sigma_marg=np.sum(s_linear*np.exp(-0.5*chisq_linear))/np.sum(np.exp(-0.5*chisq_linear))
var_marg=np.sum(var*np.exp(-0.5*chisq_var))/np.sum(np.exp(-0.5*chisq_var))

print 'fitting for sigma gives me ',sigma_marg
print 'fitting for variance gives me ',var_marg,' which works out to a sigma of ',np.sqrt(var_marg)
```

```
>>> execfile('gauss_prior.py')
fitting for sigma gives me  1.11204375505  with data amplitude  1
fitting for variance gives me  1.97240902231  which works out to a sigma of  1.40442480123
>>> execfile('gauss_prior.py')
fitting for sigma gives me  1.00478226771  with data amplitude  5
fitting for variance gives me  1.0228440997  which works out to a sigma of  1.01135755285
>>> execfile('gauss_prior.py')
fitting for sigma gives me  1.00461092104  with data amplitude  20
fitting for variance gives me  1.0101087314  which works out to a sigma of  1.00504165655
```

Keep in Mind…

- This became important once we switched to marginalizing over parameters. σ vs. σ^2 have the same peak likelihood, but different marginalized values.
- This makes the biggest difference if the errors are fractionally large.
- Bayes is always there once you average over distributions. Don't forget!

Frequency Resolution

- Monochromatic phase is $2\pi b\cdot\theta$, where $b=d/\lambda$
- What happens if we have finite frequency resolution?
- $\theta\sim\lambda_0/D$ at edge of FOV, so phase $\sim 2\pi(d/\lambda)\cdot(\lambda_0/D)\sim 2\pi(d/D)(1+d\lambda/\lambda_0)\sim 2\pi(d/D)(1+d\nu/\nu)$
- Phase difference has to be order unity or better, so we want $2\pi(d/D)d\nu/\nu<1$, $d\nu/\nu<D/2\pi d$. Read out each baseline for many narrow-frequency *channels*.
- Somewhat surprising result - # of channels (for fixed *fractional* bandwidth) is independent of frequency, set just by separation in dish diameter.

VLA Example

- VLA changes configurations. Most compact is D array - about 1 km baselines. Most extended is A array - 30 km baselines.
- What frequency resolution do we need?
- 25m dishes. At D array, $d\nu/\nu<25/1000*2\pi$, or $\nu/d\nu\sim250$. At A array, $\nu/d\nu\sim7500$
- Higher resolution means more data

Extreme Example

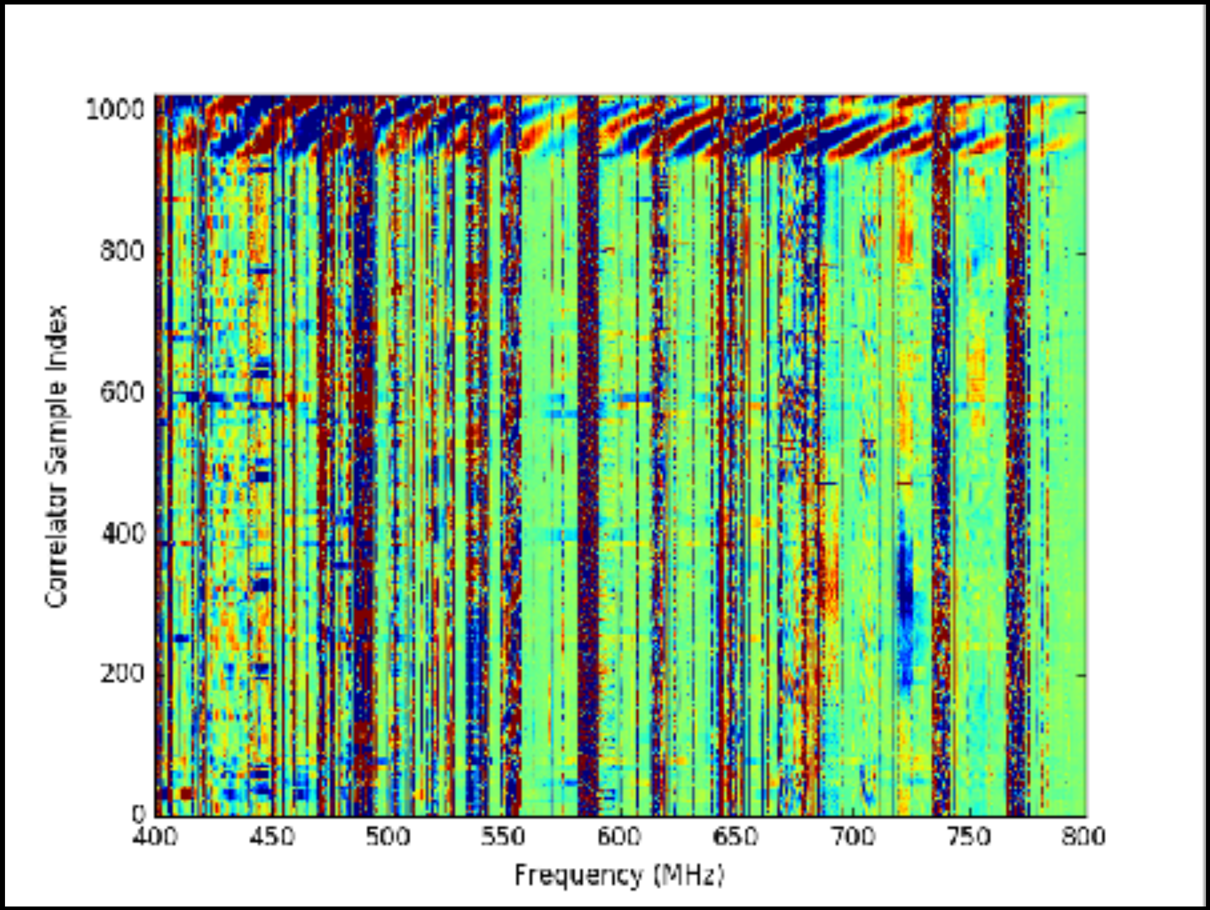
- Event Horizon Telescope (EHT) observes with ~10m dishes at 300 GHz with 10,000 km baselines.
- $\nu/d\nu \sim 2\pi * 1e7/10 \sim 6{,}000{,}000$ channels. This is a lot(!)
- What happens if we can’t handle that many?

Field of View

- We got frequency resolution by saying source at edge of field has to have phase coherence across channel.
- If I only care about things near the center, coarser channelizing will keep center in phase, but reduce field edges.
- If I know I only care about center of FOV (e.g. looking at black holes), then can get away with fewer.

Might Want Many Channels Anyways…

- People like to communicate!
- This is the bane of radio astronomy - radio-frequency interference (RFI)
- Communication often carried out over narrow bandwidths - what would typical audio channel width be?
- If our channels are wider than RFI, then we lose otherwise good data.



Above: HIRAX prototype spectrum from HartRAO outside Johannesburg
Axes are frequency (horizontal) and time (vertical). Tilted strips at top are source on sky going through beam. Everything else is interference.

RFI Width

- We can hear up to 20 kHz (if young. I can’t any more…)
- So, ~20 kHz should be enough to get voice across. Less if we ditch high frequencies. More if we want stereo.
- CB/AM radio - 10 kHz channels. FM - 200 kHz, only allowed to use 150 kHz
- Generically if RFI is a problem, 10 kHz would be nice. VLA D-array has many more channels than you might think because of RFI.

CB Radio Channels (FCC)[34]

Channel	Frequency	Channel	Frequency	Channel	Frequency	Channel	Frequency
1	26.965 MHz	11	27.085 MHz	21	27.215 MHz	31	27.315 MHz
2	26.975 MHz	12	27.105 MHz	22	27.225 MHz	32	27.325 MHz
3	26.985 MHz	13	27.115 MHz	23	27.255 MHz	33	27.335 MHz
4	27.005 MHz	14	27.125 MHz	24	27.235 MHz	34	27.345 MHz
5	27.015 MHz	15	27.135 MHz	25	27.245 MHz	35	27.355 MHz
6	27.025 MHz	16	27.155 MHz	26	27.265 MHz	36	27.365 MHz
7	27.035 MHz	17	27.165 MHz	27	27.275 MHz	37	27.375 MHz
8	27.055 MHz	18	27.175 MHz	28	27.285 MHz	38	27.385 MHz
9	27.065 MHz	19	27.185 MHz	29	27.295 MHz	39	27.395 MHz
10	27.075 MHz	20	27.205 MHz	30	27.305 MHz	40	27.405 MHz

Canadian Radio Spectrum Allocation

Channelizing

- So, how do we actually split up timestreams into channels?
- Simple, just FFT samples from ADC, right?
- Not so fast….
- We have a continuum of frequencies going into our telescope, which in general won't be an integer number of wavelengths per chunk.
- Edge effects are going to be important…

How to Channelize

- We’re going to have to do some sort of windowing. But, what do we want the output to look like?
- Within a channel, we want a flat response to all frequencies in it, with no response outside.
- So, our ideal frequency response is a boxcar.
- In a perfect world, we’d take infinitely long FT, convolve that output with a boxcar, and sample.

Windowed FFTs are not flat

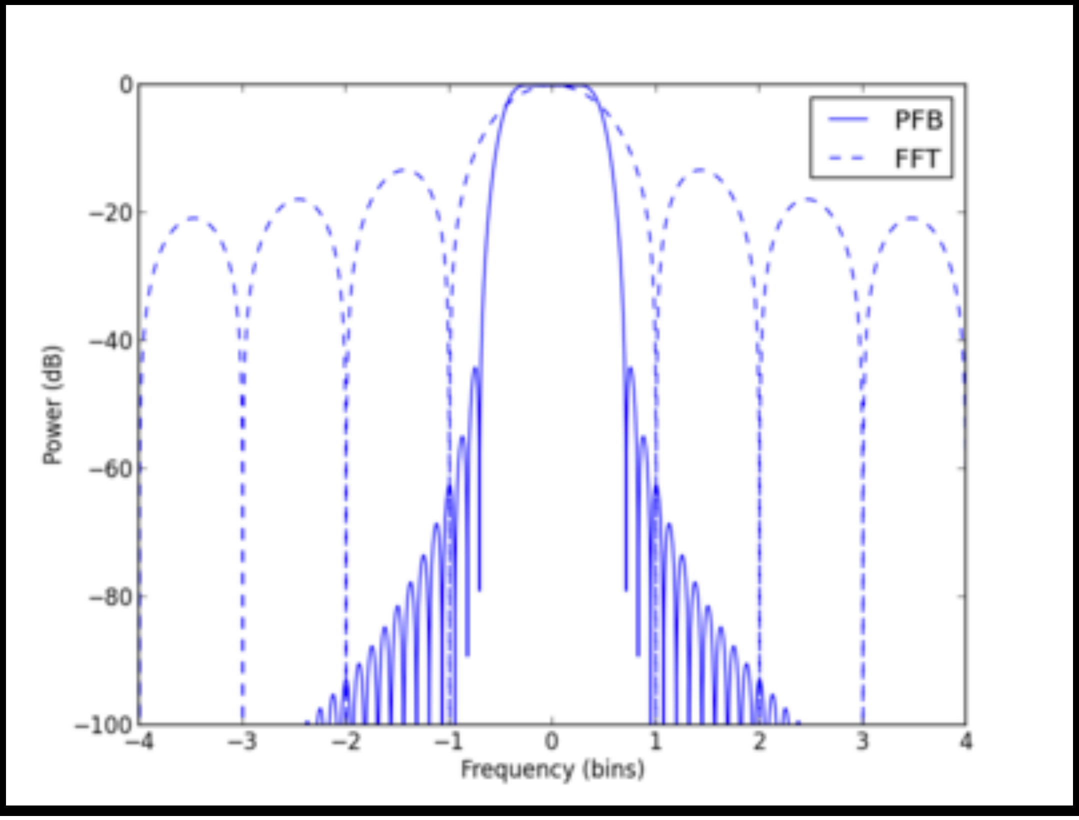
```
import numpy as np
from matplotlib import pyplot as plt
plt.ion()
npt=2048
x=np.linspace(0,1-1.0/npt,npt)


nu_range=np.linspace(305,306,11)
win=0.5-0.5*np.cos(2*np.pi*x)
for nu in nu_range:
    y=np.cos(2*np.pi*x*nu)
    yft=np.abs(np.fft.rfft(y))
    yft=yft/np.sqrt(np.sum(y**2))/np.sqrt(npt)
    y2=y*win
    y2ft=np.abs(np.fft.rfft(y2))
    y2ft=y2ft/np.sqrt((np.sum(y2**2)))/np.sqrt(npt)
    print 'max vals are for unwindowed/windowed are',yft.max(),y2ft.max(), ' with freq ',nu
```

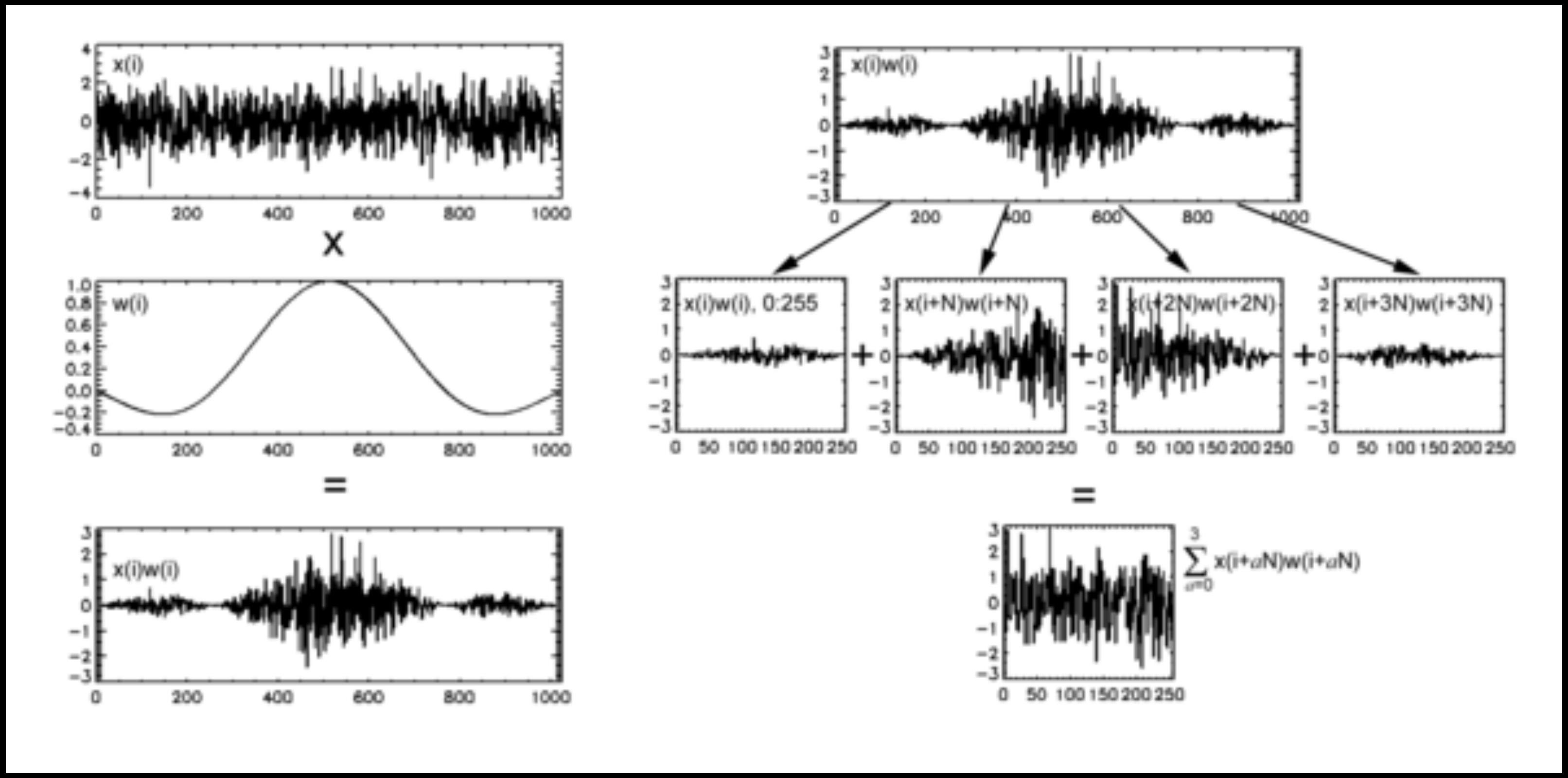
```
>>> execfile('spectral_leakage.py')
max vals are for unwindowed/windowed are 0.707106781187 0.57735026919  with freq  305.0
max vals are for unwindowed/windowed are 0.695540778175 0.573636357304  with freq  305.1
max vals are for unwindowed/windowed are 0.661549052614 0.562609364258  with freq  305.2
max vals are for unwindowed/windowed are 0.607076312371 0.544608603224  with freq  305.3
max vals are for unwindowed/windowed are 0.535155775712 0.520183471425  with freq  305.4
max vals are for unwindowed/windowed are 0.450411804613 0.490070130016  with freq  305.5
max vals are for unwindowed/windowed are 0.535454415294 0.520183470271  with freq  305.6
max vals are for unwindowed/windowed are 0.607138889807 0.544608602849  with freq  305.7
max vals are for unwindowed/windowed are 0.66152658314 0.562609364531  with freq  305.8
max vals are for unwindowed/windowed are 0.695532586344 0.573636357679  with freq  305.9
max vals are for unwindowed/windowed are 0.707106781187 0.57735026919  with freq  306.0
```

PFB

- Convolving an FT with a boxcar is the same as multiplying time series by a sinc.
- Want the boxcar to have finite width. Width is called number of *taps*.
- Turns out finite-width boxcare w/sample is equivalent to splitting sinc-multiplied timestream into # of taps, add together, and FFTing.
- This is called a *polyphase* filterbank or PFB.

PFB From Casper





PFB Steps

- Decide on frequency resolution/# of channels (boxcar width in frequency) and # of taps (boxcar width in samples).
- Take chunk of data n_{tap} times n_{chan} long. Multiply by sinc, and possibly extra window for more out-of-band rejections.
- Split into n_{tap} pieces.
- Add pieces together and FT.

How many operations to correlate?

- Correlation us
uniformly no
- VLA - 27 dua
split up). 54 i
complex mult
- CHIME - 102
crunching ne

19	GSIC Center, Tokyo Institute of Technology Japan	**TSUBAME3.0** - SGI ICE XA, IP139-SXM2, Xeon E5-2680v4 14C 2.4GHz, Intel Omni-Path, NVIDIA Tesla P100 SXM2 HPE	135,828	8,125.0	12,127.1	792
20	United Kingdom Meteorological Office United Kingdom	Cray XC40, Xeon E5-2695v4 18C 2.1GHz, Aries interconnect Cray Inc.	241,920	7,038.9	8,128.5	
21	DOE/SC/Argonne National Laboratory United States	**Theta** - Cray XC40, Intel Xeon Phi 7230 64C 1.3GHz, Aries interconnect Cray Inc.	280,320	6,920.9	11,661.3	
22	Barcelona Supercomputing Center Spain	**MareNostrum** - Lenovo SD530, Xeon Platinum 8160 24C 2.1GHz, Intel Omni-Path Lenovo	153,216	6,470.8	10,296.1	1,632
23	Forschungszentrum Juelich (FZJ) Germany	**JUWELS Module 1** - Bull Sequana X1000, Xeon Platinum 8168 24C 2.7GHz, Mellanox EDR InfiniBand/ParTec ParaStation ClusterSuite Bull, Atos Group	114,480	6,177.7	9,891.1	1,361
24	NASA/Ames Research Center/NAS United States	**Pleiades** - SGI ICE X, Intel Xeon E5-2670/E5-2680v2/E5-2680v3/E5-2680v4 2.6/2.8/2.5/2.4 GHz, Infiniband FDR HPE	241,108	5,951.6	7,107.1	4,407

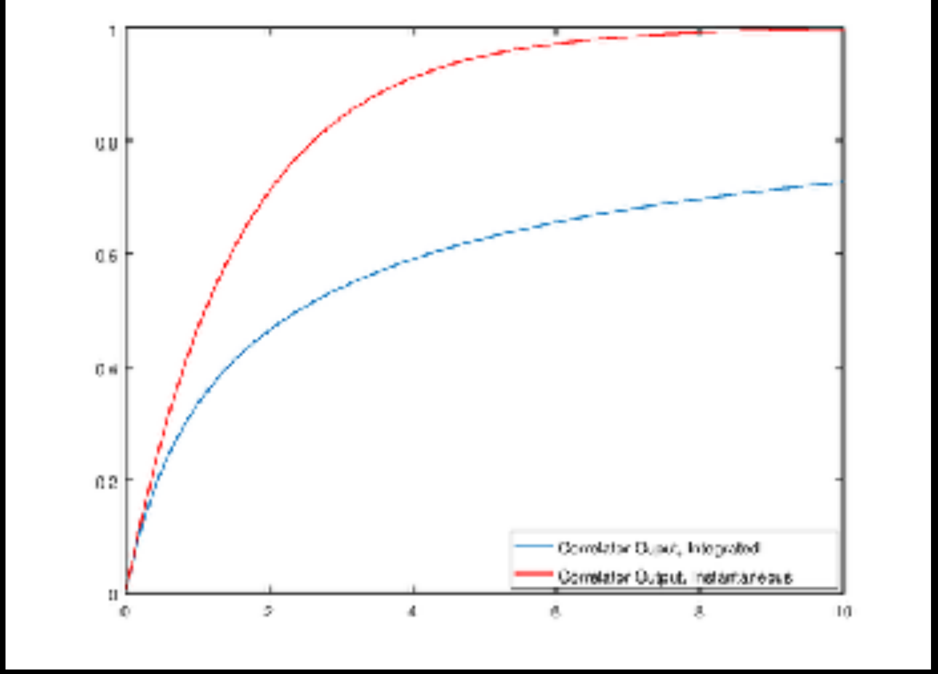
How Many Bits?

- At these rates, higher-precision operations cost more than lower-precision ones.
- Extreme limit is *1-bit* correlation. For each sample, I get +1 if both inputs are positive *or* both inputs are negative. I get -1 if both inputs are opposite sign.
- What happened to the amplitude(!)?

S<<N

- Take limit where signal is much less than noise.
- Noise is gaussian-distributed w/ zero-mean (very good approximation after channelizing)
- If signal is small, adding signal to two different channels increases the probability that both signals are either positive (for positive signal) or negative (for negative signal).
- In fact, can write this down in terms of *erf*s.
- As signal gets large, correlator output saturates.

1-bit Output



- Right: 1-bit correlator output as function of signal-to-noise power.

- Response (from erfs) is 2/π, whereas response from ideal is 1. Noise in both is 1, so SNR of a 1-bit correlator is 2/π vs. ideal.

- If I’m limited by ops/storage, maybe this is a win

- More bits does better - 2 bits ~80%, 3 bits ~95%, 4 bits ~99% of ideal.

Back-End Signal Path

- People moving to FX correlators - split timestreams into channels (F), then cross-correlate (X).
- RFI means channelizing often happens at many bits (why? intermodulation if digital effects become important).
- Cross-correlation can happen at fewer bits.
- Standard GPU can do ~10 TFlops of single-precision math.
- BUT, NVidia has tensor cores that do 4x4 multiplies, and can do so at 16bit float, 8-bit int, and 4-bit int. At 4-bit ints, 1 card can do ~500 Tops! (not really flops)
- In principle, ~16 $1000 GPUs has enough horsepower to correlate CHIME. Would be top-20 system at double precision.