# 1 Relevant Background Material

## Caltech Course

Fourier Decomposition - Take a signal apart into sine wave. The amplitudes and phases are needed once sampled, with the Nyquist frequency being the highest component. For a 10-pixel signal, 5 sine waves and one DC signal are required. The transform shifts from real to reciprocal space, where each sine wave is represented by a pair of dots.
A 2D sine wave is over x,y, with h,k being the miller frequencies, you can make a 2D grid of the component frequencies and their amplitudes. Each wave is 2 points, as you can't tell the direction.
The sum of the images is the plot of all the dots, intensity (amplitude$^2$) is based on the darkness of the dot, representative of the power spectrum. This shows the distribution of power into frequency components.
Dots close to the origin vary slowly and compose the low-resolution information, and further away are high-resolution, quickly varying stuff. You can use high-pass, low-pass or band-pass filters to gt specific image data, and then perform an inverse Fourier transform to get the image back.

Convolutions - $P \circledast Q = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} P(\tau)Q(t-\tau)d\tau$, where $\tau$ is the sliding window of the convolution.
This is the integral of the point-wise multiplication of two functions as one is translated. If you convolve a function with the Delta function, you get the waveform back. Microscopes have a convolution associated to them, which is similar to a convolution with a point-spread function. For example, blurring kernels use convolution.

Convolution Theorem -

$$\mathcal{F}[P \circledast Q] = \mathcal{F}(P)\mathcal{F}(Q)$$

A convolution in the spatial domain is a multiplication in the Fourier domain.

Cross-Correlation - Assesses the similarity between two functions, $P * Q = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f^*(\tau)g(t+\tau)d\tau$, in essence it is a sliding dot-product, which compares the values of a probe image position-by-position to values of a test image.
Cross-correlation coefficient is a 2D matrix. f is the probe image, while g is the test image. There is no flipping of the image like in convolution, but the easiest way to calculate it is via the convolution theorem.

Scattering - Transmission electron microscopy has corresponding contrasts
Amplitude - Caused by electrons behaving as particles that get scattered with some getting removed form the image place, so the total number of electrons gets reduced, creating contrast.
Phase - Due to electron's wave behaviour, as they undergo a phase-shift when scattered and recombine/interfere later on, creating an altered probability of the electron being detected, and contrast.
Electrons can be modelled as travelling in plane-waves, of an amplitude and a phase shift. As it moves downwards, the phase advances around a circle, and once it hits the sample, the wave scattered creating a phase ripple. Then all the scatterings interfere, like an infinite-slit experiment.
When an Z-ray hits an electron, it created an oscillating electric field, and the plane wave hits multiple. All the ripples constructively interfere in multiple directions at a specific angle $\alpha$. Each particle has certain scattering centres, and if they are closer together, the scattering angle is higher (between particles, between helices, between atoms). If $\sin\alpha = \frac{\lambda}{d}$, they interfere constructively. More scattering centres give a stronger amplitude.
Scattering reveals the amount of each Fourier component, in essence it splits the information up.
The diffraction pattern is a Fourier transform, and the inverse Fourier transform is present on the image plane.

Contrast Transfer Function - Not all the waves are delivered with the same strength, depending on the phase shift they are subject to. When the wave passes the same, some of it isn't scattered and continue to move down with the phase changing, and the part that was scattered interferes with the un-scattered part. These waves also both trace different paths, undergoing differing amount of phase shift. When they differ by 180°, the vector is shorter than it should be, and when it is 90°, there isn't much change and the wave isn't present in the image.
The CTF stated how much of the wave has an impact on the sum. It varies from -1 to +1, and undergoes a wave-like pattern as the frequency increases.
The higher the scattering angle, the higher the spatial frequency, if 360° it gets a contrast of 1. The function

starts at 0, and then goes to a minimum. It is the contrast into the image based on frequency.

Defocus - Thon rings are concentric circles, shows lots of info transfer at low-frequency. It is a complex band-pass filter.
The defocus is a term in the CTF calculation, in-focus means that the sample plane is conjugated to the image plane, but defocus is useful as it gets the wave function at different places, with different phase relationships, shifting the CTF and changing the focus planes.
At **high defocus, low frequencies are favoured** with high frequency information garbles. AT **low defocus, high frequencies are favoured**.

Envelope - The CTF undergoes damping at high frequencies, with a higher defocus having a tighter envelope, corresponding to less signal at higher frequencies due to the phase shifts being drastic and cancelling out.

The point-spread function and contrast-transfer functions are Fourier transforms of eachother, for a perfect CTF the PSF is a delta function.

$$\begin{aligned} \mathcal{F}(\text{image}) &= \mathcal{F}(\text{object})\mathcal{F}(\text{PSF}) \\ \text{object} &= \mathcal{F}^{-1}(\frac{\mathcal{F}(\text{image})}{\text{CTF}}) \end{aligned}$$

## Sigworth Paper

The goals of proposed methods are to

1. Detect individual particles in images with noise
2. Distinguish particles from artifacts

This approach uses a correlation-based detector followed by a classical mathed filter, which uses a standardized noise model vis prw-whitening filtering (taking the signal and producing a white one based off a predictor).The dot product of two images is the sum off the products over all the pixels, $X \cdot Y = \sum_1^{np} X_i Y_i$, with the squared norm, or power, of the image being the self dot-product.

Noise model - Assuming Gaussian noise for simplicity, which has a flat power-spectrum. We can state the image as a sum of particle motifs plus a noise term, X = mA + N, and the power of A = 1.
But the micrograph noise isn't white as in-elastically scattered electrons contribute at low frequencies and there is a fall-off at higher frequencies due to the PSF.

Pre-whitening filter - Get a circularly averaged power-spectrum, and fit the curve p(fr) to the spectrum, where fr is the radial frequency. Application of this filter makes the noise approximately white, and removes some low-frequency components. We assume X is normalized and pre-whitened.

The best filter to use is the matched-filter detector, or correlation detector (essentially cross-correlation).
The matched filter is followed by a threshold detector to filter the particle images. The underlying particle motif should also be pre-whitened, and if it isn't noiseless, pass it through a low-pass filter to reduce noise. Given a threshold value, you can calculated the density of expected peaks above that value.

Multiple References - Useful to have many projections with the same filtering as the image, to save time you can filter it before projecting, as it only depends on the radial distance. Try to reduce amount via clusstering, produce a set of images, eigenimage expansion of the set.

Summary:

1. Calculate the power spectrum to create a pre-whitening filter which is applied to the data and reference.
2. Create a set of reference projections, ideally an eigenimage expansion
3. Use the correlation detector
4. Iteratively find the maximal value location, and set the peak to 0 until the value falls below a threshold.

5. To check for false-positive, use squared-error test, and for an effective test you need a reduced representation of the images.

Subtract the refernece from the detected image, multiply it by circular mask M. From this, computer a rotationally avereged power spectrum and compute $t = \int_0^\infty P(f)w(f)df$, where w is a wighing function to accentuate frequencies near the particle. This t and the variable s, which is the amplitude of the peak, can be used to identify particles. Create a scatter plot with the good particles in a bounded region.
1. Standardize noise. 2. Fast algorithm for multiple correlations 3. Discriminant t, might determine s empirically, as it would be different for different projections. Ideally want local noise averaging.
Circularly averaged power spectrum - Choose a square, average the power spectrum in circles around it, creating a function for radius.

### Rickgauer Paper

Template matching - Compares computed templates to 3D sample reconstruction, but it is very difficult in crowded environment and for small particles. Using internal structure will be a more robust way to go about this, high-resolution info. At high resolution, the projections are unique, which is useful for detection but not reconstruction. Close-to-focus and a match filter aid when there is a background, as it limits the low-frequency information.

Cross-correlation allows matching, even in crowded environments.
Use hopf fibrations for rotations for close-to-focus.
The results weren't overly affected by the defocus of the microscope, template matching didnt work at high defocus (2000nm) but did a t low (70nm), probably due to low-frequency noise. Should use a full-matched filter with whitening.
Images from closer to focus had an increase in accuracy, which can be combined with whitening.

## 2 Lab work

### Undated Entries - Basic Coding Concepts

np.fft.rfft2(img), np.fft.irfft2(img), np.array([1,2],[3,4]) creates an array object. np.fft.fftshift shifts the zero-component to the centre.
Matplotlib - plt.plot(x,y), plt.ylabel(''), plt.imshow(), plt.subplot(121), plt.title(), plt.xticks([]), plt.colorbar()
img = scipy.misc.imread('file', 1(greyscale)). warning.filterwarnings['ignoer', category = 'DeprecationWarning']

### February 8$^{\text{th}}$

Mask image is a zero-padded array of 1s. Binned particles can be zero-padded by placing in the middle of a zero-array. Get the sizes of the image, and put it into the centre of a zero image.
Averaging - NCC - scipy.signal.fftconvolve(img, b[::-1,::-1]), convolves with the flipped image, as convolution automatically flips it. we normalize by diving by the norm of the template and the image.

$$\begin{aligned} imgnorm &= \sqrt{\mathcal{I}^2 \circledast (paddedmask)} \\ tempnorm &= \sqrt{\sum \text{temp}^2} \end{aligned}$$

imgnorm is the same as the co-variance, it takes a pixel in the image, adds it and takes sqrt. Convolution of image with a template of 1s.

pickling is a way to store numpy arrays. np.roll(fc, -int(height/2 + 1), axis = 0) is a way to shift elements along an axis.

### April 1$^{\text{st}}$

- As a sanity check, by writing your own version of a code, you can subtract one result from the other to see if/how they differ. You can use min/max of a well-behaved image to set bounds for one that isn't
- By convention fft is such that DC components are at the edge of resulting matrix, while Nyquist is in the middle. Need to swap the quadrants with fftshift

- The fft of a real signal is a Hermatian, same frequency components but different signs. rfft only does half.

Code so far: Can calculate the normalized cross correlation between an image and a template.

**normalized_cc (image, template, use_numpy_ver = True**

1. T = T - np.mean(T), subtracting off the mean of the template
2. Zero pad the template to the same size as the image
3. Compute the fast_convolve of the image with the flipped padded template
4. Compute the normalization denominators as described above
5. Calculate the normalized cross correlation = crossc/(denom1*denom2)

**fast_convolve(I,T, use_numpy_ver**)

1. Compute the 2D Fourier transforms of I and T (the padded template)
2. Perform the convolution, which is a multiplication of the transforms
3. Perform the inverse transform and shift the result.

The scipy version has boundary issues.

## April 26th

We want to optimize the code, by cutting down of the calls to the Fourier transform, we can do this by calling it earlier in the function stack and passing the result forward as a variable. Same with creating masks.

Next is to check multiple templates against a single micrograph, which would return N normalized cross correlations.

The way to write code is to first write it the simplest way that you know is correct, and only then go through it to work on efficiency.
#!/usr/bin/env python3 is a hashbang, tells linux how to run the script.

## April 30th

For a mask/image indexing mask[a1:a2,b1:b2], a is the vertical component, and b is the horizontal, starting from the top-left corner. Can bin particles by cutting out the image, b = img[particle location].
os.listdir() shows everything in your current working directory, you can filter out files using os.path or os.walk.
for part in os.listdir() -¿ os.path.join(folder,part)
Testing out speed, my function is twice as fast as scipy, possibly due to the real fft being used.
Tested whether to take the absolute value of the numpy normalized cross correlation. Checked with the scipy version and decided to go with it.

## May 1st

Quickly tested if there was a relationship between the Fourier transform of an image squared to the Fourier transform squared.
keyword arguments are useful for passing values to a function.

## May 7th

I want to be able to combine the images and determine which has the greatest value/likelihood of being the correct particle.
Trying to detect the peaks using maximal suppression.
Played around with an adjustable mask size, thinking of creating a dictionary for each particle with the location data linked to the sum of the masked image, but it might not be enough to just have the mean or the sum, as the resultant values are all similar. Can change the size summed over to get more of what is expected.

The indices don't line up well for tha maximal detections of varying particles, need to account for this when comparing the dictionaries.
i want to create a dictionary for each particle with coordinates and mean value of a smaller area of the detection, then looping through the keys with a coordinate tolerance threshold and compare the values.
my_list = [elem[0] for elem in dict.keys()]

## May 9th

## June 12 th

### Meeting Recap

- We went over the probability calculations, paying close attention to the notation. j = particle view, k = particle configuration, s = particle.

$$\begin{aligned} e_j k &= -log\mathcal{P}(I|T_j^k) \\ \mathcal{P}(S_k|I) &= \frac{\mathcal{P}(I|S_k)\mathcal{P}(S_k}{\mathcal{P}(I)} \\ \mathcal{P}(I|S_k) &= \sum_j e^{-e_{jk}} \\ E_k &= -log\mathcal{P}(I|S_k) - log\mathcal{P}(S_k) \\ \mathcal{P}(I) = \sum_k \mathcal{P}(I|S_k)\mathcal{P}(S_k) &= \sum_k \mathcal{P}(S_k)e^{-E_k} \end{aligned}$$

- Might not want $-log\mathcal{P}(S_k|I)$, and instead give the regular probability via $\frac{1}{2\sigma^2}$, where implicit as of now $\sigma = \sqrt{1/2}$ but can be calculated.
- Gaussian Noise Estimation:

## June 13th

- One of the images from my probabilities is inverted, and I don't know the cause. For now I am imply swapping the black and white.
- Regardless of how many template stacks i use (went up to 5) always one of them is 'inverted'
- I implemented the sigma functionality using an estimator.

## June 18th

- Can return multiple things from a function via return(x, x0, x2), and access by index. Or can use a dictionary return 'y0':yo...
- I fixed up the code to work with method 2, where probabilities vary
- what I want returned:

$$\begin{aligned} \mathcal{P}(S_k|I) &= (probstack) \\ -log\mathcal{P}(I|S_k) &= -log\sum_j e^{-ejk} \\ -log\mathcal{P}(I) &= -log\sum_k \mathcal{P}(S_k)e^{-E_k} \end{aligned}$$

- Returned these values, but one of the $\mathcal{P}(S_k|I)$ is 'inverted', not sure why
- ek stack and ejk stack are heavily blocked by some artifact.
- Need to figure out how to combine all these things and pick out the particles
- The CTF is a function of how information is transmitted as a function of frequency, equivalent to a point-spread-function, it spreads the information around the image plane in a global manner.
- $\sum(I_i - T_i^{j,k})^2 \rightarrow \sum(I_i - CT_i^{j,k})^2$, where C is a dense array representing convolution.

## June 20th

- Tracked down the blocking my images were getting to the convolution of the image with a passed square in the normalized cross-correlation, the $\sum I^2 = \mathcal{IF}[\mathcal{F}(I) * \mathcal{F}(\text{padding})]$ term.

## June 21th

- Abbas sent me files with various configurations of particles from the stack or not, pre and post CTF noise, supplied me with the noise values and particle counts. Figured out how to use CTF code, next implementing normal microscope parameters: psize = 2.8, akv = 200, csf = 2.0, wgh = 0.07, df = 10k:15k, ang = 0, dscale = 1, bfactor = 500.

## June 25th

- Get PNGs of all the micrographs, implement basic CTF functionality
- Abbas: CTF is applied to a padded area twice the length of the projection. It is applied to each template: projection and the pre-CTF noise. A 260*260 structure gets padded to 520*520 and CTF applied. The parameters are: defocus = 12000, astig = 1000, angle = 0, akv = 300, wgh = 0.1, cs = 2, psize = 2, bfactor = 10, dscale = 1
- Particles: 27001 is small and irregularly shaped. 28001 is spherical and has a diffuse center, symmetric. 29001 is spherical and has a dense center.
- CTF Functionality:

$$\sum(||\tilde{I}(\omega) - \tilde{C}(\omega)\tilde{T}(\omega)||^2$$
$$\sum \tilde{I}^2(\omega) - 2\sum I(\omega)\tilde{C}(\omega)\tilde{T}(\omega) + \sum(\tilde{C}(\omega)\tilde{T}(\omega))^2$$
$$(E)_{jk} = \sum_i (I_i - T_i^{j,k})^2 = \sum I_i^2 - 2\sum I_i T_i + \sum T_i^2$$

- Set values to None instead of 0, better form
- the blocking the I see doesn't really matter as it is constant, and I should just leave it out of my calculations
- the regular (non-log) probability for the micrographs not fro the stack doesn't turn out well, so using the negative log instead. Can change values of corner pixels to have all images with same scale.

## June 26th

Flow: Using $-(log)\mathcal{P}(I)$ to determine particle location, use $-(log)\mathcal{P}(S_k|I) - (log)\mathcal{P}(I|S_k)$ to determine which particle it is.
Possible to save the whole calculated array with np.save and np.load.

## June 27th

Assuming particles not from the stack are the micrographs that should be focused on, working with 30004 and 30006.

1. Create empty lists, variables, dictionaries
2. Loop through $-(log)\mathcal{P}(I)$, apply a maximal threshold, take location and apply a mask
3. At that point, calculate some value from the $-(log)\mathcal{P}(S_k|I)$ and $-(log)\mathcal{P}(I|S_k)$ to be able to pick the particle.

Will process first image separately, as the template gives weird results. Was masking the corresponding image and looping through it separately.

## June 28th

Can correctly identify particles in 30004, 30005, 30006, 30010. For 30007 and 30008 the detection of the smaller particle needs much work.

## June 29th

Right now my code has four outputs,

$$\begin{aligned} z1[0] &= -(log)\mathcal{P}(S_k|I) \\ z1[1] &= -(log)\mathcal{P}(I|S_k) \\ z1[2] &= -(log)\mathcal{P}(I) \\ z1[3] &= \mathcal{P}(S_k|I) \end{aligned}$$

The code takes z1[2], goes and finds maxes, applies a circular mask of the minimal value onto it until the maximal value is less than the median. For the non-zero z1[0]'s a dictionary is created with the location as the key and np.mean(z1[0][mask])/np.median(z1[1][mask]). A mask is also applied onto the $-(log)\mathcal{P}(S_k|I)$ which didn't load properly. It then loops through until the maximal value is less than np.mean($-(log)\mathcal{P}(I|S_k)$).

## July 4th

### Meeting Recap

Corrected my calculation for the $\sigma$, as I was computing $\sigma^2$. Also began using Overleaf to write my notes on, allowing meetings to be more conducive.The problems I had were about implementing the CTF function, the difficulties I had with the probability always being maxed. The solution to the second problem was to remember you can subtract the median from a -log probability and it will give you the same result, also to just use the -log probability instead of calculating the regular probability. Also, Marcus suggested to include a "fourth template" consisting of an array of zeros which should match to the background, allowing one of the particles not to match. There is probably a good way to implement this, but for now I just created a blank array and run the code with it. The calculated $\sigma$ should relate to those before and after the CTF by the formula $\frac{1}{2}\sigma^2_{pre} + \sigma^2_{post} = \hat{\sigma}^2$

For the Fourier part of the CTF,in real-space the CTF is a dense matrix representing convolution. We know that the Fourier transform is linear

$$\begin{aligned} \mathcal{F}(A) + \mathcal{F}(B) &= \mathcal{F}(A+B) \\ \mathcal{F}(A-B) &= \mathcal{F}(A) - \mathcal{F}(B) \\ ||A^2|| &= ||\mathcal{F}(A)||^2 \\ ||A-B||^2 &= ||\mathcal{F}(A) - \mathcal{F}(B)||^2 \end{aligned}$$

where A is I and B is C*T

$$||\mathcal{F}(I) - \mathcal{F}(C)\cdot\mathcal{F}(T)||^2$$

So now, expand the bracket, and we might need to zero-pad slightly, but this should be able to be done per pixel of the micrograph, as this is calculated per pixel.The CTF should be calculated being the same size as T.

### Today's work

I learned how to use LaTex, and I ran the code with the new sigma, as well as with a template of zeros for 30004 and 30008. The only downside to include a zero stack is that the -logP(I) image doesn't help find particles anymore, as it is mostly zero.

## July 5th

### Goals for Today

I will begin by focusing on image 30004, which has noise, SNR = 10, and chosen not from stacks, then will move to 30010 and 30008, and hopefully work on the CTF implementation. Also begin to write an abstract.

**Work**

Testing out $-(log)\mathcal{P}(I)$ shows that it can be used to detect the large particles, 34 in 30004 and 40 in 30008. However that is the limit, and it cannot detect the other 20 in each. $-(log)\mathcal{P}(S_k|I)$ of the zero template can be used and yields the exact same results.
Trying to accomplish detecting particle location by multiplying all the $-(log)\mathcal{P}(S_k|I)$ together, it works for 30004 but not for 30008.Can't figure out proper thresholding for 30008. I am trying to threshold it by stopping it once it hit the mean on the multiplied image, which works for 30004 but not for 30008. Maybe it will be easier once I implement the CTF functionality.

Playing with the CTF padding. Attempting to pad C and T before taking the Fourier transforms.
The three components of the extended bracket are on very different magnitudes, so I chose solely to focus on -2*C*T*I.

I've been playing with it a lot, trying to remove the zero stack, and on a few different micrographs, but I don't really know how to interpret the results. They are not consistent with anything expected, seemingly random if the particle is a maximum or a minimum, and difficult to detect faint particles.

## July 7th

Tried to use the logsumexp over the probabilities to try and get an image that can be picked easily. Seems to work fairly well for -lse(-z1[0][0:3]). Trying also with the CTF variant.Want to get CTF array for 30008 and test the method on it.

## July 8th

## Goals

Today I want to search on-line for some resources and see different methods for peak detection, as I have hit a wall. I might also manually input the amount of particles, and spend the day developing a way to detect which particle it is instead of finding them in the first place. The locating problem is difficult due to one of the particles being a different size, which throws everything off.

## On-line reading

- Detecting dog paws, 2D array with values of maximal pressure. Could break down into two problems: Detection and sorting. Uses scipy maximum filter, generated a binary structure (2,2), then applied local maximum filter, where all pixels of maximal value in neighbourhood are set to 1. local_max = maximum_filter(image, footprint=neighborhood)==image, where local_max is now a mask with peaks but also the background. Need to remove the background from the mask.Create a mask of the background, background = (image==0), need to erode background to successfully subtract it, eroded_background = binary_erosion(background, structure=neighborhood, border_value=1). Then use xor to remove the background peaks = local_max êroded_background

## Work

- First tried to copy what was in the on-line reading, which resulted in boxes where the particles were, but I had no way to actually mark the place of the particle and accurately count it
- I then tried doing something similar to my other attempts where I used the logsumexp of the negative P(Sk—I). But this time, I filtered it with a Gaussian filter with $\sigma = 25$, which accurately determined particle locations for 30004, 30008, 30010, and 30012. Looks promising. Will now try to classify particles. The lower the sigma, the more 'particles' are detected. 30007 gives 65 particles instead of 59, which is great as it doesnt hit the limit of 70. Sigma of 26 gives 62 and 28 gives 58.For 30008: 26 gives 61 and 28 gives 60. For 30012, up to 30 gives 58. For 30011, sigma of 25 gives 63, 28 gives the correct 62.
- Using this particle picking technique, I will now go back and use these locations and the probabilities to determine which particles are being detected.
- Started writing abstract.

## July 10th

### Goals

- Finish writing abstract
- Work on a way for particle sorting for 30008 and 30012 as the prime focus
- See what I can do with the CCTF stuff

### Work

- Finished first draft of abstract, will send to Marcus for review. He reviewed it and will send in shortly.
- Trying to find some criteria as to judging the likelihoods. Difficult for particle 1. For particles 2 and 3,

  $$\text{dictlist[xx].append(np.mean}(z1[0][xx][y-10:y+10, x-10:x+10]))$$

  works well for 30004, 30008 and 30012. With this, particle 1 is being included in the dense-centered particle (particle 2). I cannot seem to find a good picking algorithm.

## July11th

I realized that ive been choosing the max of the comparing list when i should be choosing the minimum, as the lower the number the higher the probability. I realized the P(I—Sk) for the first particle goes into the negatives, need to see how I can deal with that and get rid of the negatives. Maybe incorporate P(I) into it, and treat the non-overlapping ones differently. dictlist[xx].append(np.mean(z1[0][xx][mask2]*(np.sum(z1[2][mask2])))) works for sorting between 2 and 3

## July 12th

### Goals for Today

Having a meeting later. I want to work on the CTF stuff and see if I can get something working and find where things are breaking down. Not having much luck at the particle picking, so I might just leave that for now and bring it up at the meeting.

### Work

- Surprisingly the code I have for detecting the peaks works for the CTF generated probabilities. Just switching around the min/max lets accurate picking for the CTF z1[0] of 30004, 30008, 30010 and for 30012 it detects 57 (not 58). For 30008 it isn't entirely accurate with the smaller particles (to the best of my knowledge).
- Using z1[1] of the CTF produces better results, better centered on the particles. array = -scipy.misc.logsumexp(array, axis=0). But this way only detects 48 for 30008 and 47 for 30012, missing approximately 20 particles.

## July 13th

### Meeting Recap

- Met yesterday and discussed the problems I was having with detection and picking. Marcus suggested to go back and look at when I omitted the "blocking" part of the code before. Print out the image, the img norm, the cross correlation, and the combined template. Also, check this for the plain micrograph and an image generated with Gaussian noise of $\sigma = 1$, which shouldn't show any blocking. Do the same with the CTF micrographs.
- CTF has a global effect. The $\sigma$ varies with noise. Up until now we working with white noise, which is just a constant. Different 'coloured' noises are present at different frequencies, and we will eventually have to incorporate this noise model into the model. AS the $\sigma$ is frequency dependent, it can be brought into the sum over all frequencies.
- The cross correlation for the regular templates has a smooth function drop off from the true position, but for the CTF, it drops off into the negatives sharply after the peaks, producing a pattern with peaks/ripples.

**Work**

- Checking the old code. I produced the micrograph, one image from the template stack, a white-padded square, an the padded template. For a micrograph and a generated noisy 'micrograph', which had $\sigma = 1$ I calculated the image normalization, the template normalization, the cross correlation and the normalized cross correlation for regular and CTF-affected templates. Posted onto slack, as per request.

## July 19th

**Meeting Recap**

- When we deal with statistics, we have no absolute scale, it is all relative. We are trying to explain the pixels in the sliding block region, but certain regions, distinct from particle presence, are hard to explain, which leads to blocking. This is based on our image model of uniform noise.
- With the smaller particles, the issue is that once we get rid of the img_norm, we dont see the compensation for the low signal level, so they get lost. If we drop that term we just prefer larger particles.
- We need to be looking at the relative likelihood, compared to a different model. Can't say if it explains it well, only if it does it better than a different model.
- Leave the terms in there, but when you visualize, subtract off the log likelihood of the zero template, which should make img norm disappear, and gives flexibility for adding different models later.
- Sort out the math for the zero template instead of including as a physical template
- The relationship between A and B is only that B is divided by P(I), which gets rid of the blocking.
- Try to plot them all together using subplot with a fixed intensity range passed as an argument of vmax and vmin.
- Start explaining the code in a scientific way in overleaf, give enough info in the intro for a reader to understand.
- Looks like fftshift is missing from CTF code, but go through the math, not trying out fftshift everywhere..

## July 23th

**Goals**

- Work out calculations for zero template.
- Begin "formal" write up
- Figure ut how to use subplot properly
- Attend meeting for presentation guidelines

**Work**

- In solving the zero_template problem, I found the only contribution it makes is that one of the entries into ek_sum is a matrix composed of the -lse over the -img_norm, which is the image squared convolved with a white square. This turns out just to be appending $\frac{1}{2\sigma^2} * (img^2 \circledast padding) + \log(z)$ to ek_sum.
- I also changed the code around a bit, and managed to get rid of one of the function calls, Error_Template.
- For the CTF function, it seems like the contribution to ek_sum is -lse of -R, where R is np.fft.irfft2($\frac{1}{2\sigma^2} *$ np.fft.rfft2$(img)^2) - np.\log(z)$. Also was able to simplify code and redcue run time/function calls.
- Figuring out subplot:
- As a sanity check i should ad up the negative exponents of all the probabilities and see that they equal uniform 1s, which they don't. Now it is time to bug-hunt.

## July 24th

- Today I spent time writing up everything I have done in a formal manner in LaTeX, Still need to write the CTF part, but it was mainly to look for errors in the code
- I found the error: I was adding the same template twice, so I changed it and got a matrix of uniform ones.

## July 25th

- Ran the CTF code, and got the sum of the probabilities to be equal to 1 as expected
- I want to put a check in to see if the list of passed P(Sk) is equal to the amount of templates for method 2
- I want to do the formal write up for the CTF code, and try to figure out where the mistake is with regards to fftshift.
- I did the write-up. I think the error lies in the normalizations in the code, as I am just multiplying them, which seems almost too simple.

## July 26th

**Work**

- Testing out different combinations for the calculation of $\mathcal{E}_{jk}$ for the CTF function, hoping to figure out why all I get is a centred circle. Tries changing the img_norm factor from $\mathcal{F}(I)^2$ to $\mathcal{F}(I^2)$.
- It wasn't the img_norm term that was making the weird results, it was the temp_norm. Changing the way img_norm was caluculated had no effect. Maybe temp_norm should be a single value like in the non-CTF case, so I will np.sum it.
- I tried many different things, but none of the

## Meeting Recap

- We spoke about how an absolute value of a complex number squared is really itself times the conjugate, $||c^2|| = (a+bi)(a-bi) = c*c$. This leads to a different expanison of the $\sum_\omega ||(I_\omega - \tilde{C}_\omega \tilde{T}_\omega)||^2$, which ends up being $I^*I + C^2T^*T - 2cRe[T^*I]$
- The calculation should be left in Fourier space, and should switch over the full Fourier transform instead of just the real one.
- I should also use unitary normalization in the calculations, as it corrects the scaling factors between the calcultion in Fourier space and the sigma parameter.
- He thinks there might be some template scaling when the micrograph is projected which would cause the weirdities seen in the regular car.

## August 1st

- Starting to put together my presentation. Still not sure about how the CTF stuff should be properly incorporated, as I don't know how the calculations in Fourier space relate to particle picking in real space without an inverse transform. Also, Marcus is still working on why the results aren't what is expected. I've continued writing in the lab report what Ive implemented, but dont really see the issue. Its pretty much the exact same output as a few weeks prior. For my presentation, I feel like it is going to be mostly "What I am trying to do", and "What comes next", and talk about the problems we have run into.
- Marcus said that the "biased" way of performing this is to process all the particles and perform 3D classification where the images are classified to structures and the structures being estimated. two problems: 1. The original selection of images might be biased, meaning the set of images isn't fully representative. 2. This processing involves somewhat arbitrary discarding sets of particles based on human decision making.
- I want to update the CTF part of the method formal thingy.

## 2.1 August 2nd

- Meeting today at 12 downtown, will recap. I have some questions regarding the CTF functionality in converting back to usable information. Also about the poster presentation, and if i can continue working next semester. Maybe we should just focus on same particles but diferetn configurations, as they would all be the same size and it seems like that has a major role to play in the detection.
- When individually plotting the particles, the range for 27001 is -20 to 15, and for the larger ones is -50 to 40.
- Also, for the CTF stuff, the CTF part of the error sum has complex values, should they just be left?

## August $10^{\text{th}}$

- A few things to change for the poster: Some notational aspects, getting rid of the algorithm contast vs. intensity, moving formulas around.
- If more templates are used, the threshold used to detect particles is dragged up.
- To sample a function need x points, then a 3D one need $x^3$ points
- Univariant v. multivariant Gaussian, gets canonincal form o exponential family in hwich most probability distributions fall.
- The calculation is done for a tiny square, and to do it over the entire micrograph we use convolution.
- Try different strategy for picking, taking the max around the detected peak. Try to get a 30008-esque one with ground truths from Abbas.