Recitation will be this Friday (9/21) in Jolley 309.
  - Will go over topics relevant to Pset.
Problem Set due next Tuesday
Notes on Parallelization

Say your simple code looked like this

```python
for i in Indices:
    y = func1(x[i])
    z = func2(x[ifunc(i)])
    p = func3(y, z)
    out[i] = func4(p)
```

Replace as

```python
y = func1(x[Indices])
z = func2(y[ifunc(Indices)])
p = func3(y, z)
out[Indices] = func4(p)
```

- Most numpy functions that act on single numbers can be used elementwise on arrays.
- Think about all the steps you would do for each number in a loop: Can these steps can be carried out independently for different loop indices?
- If so, replace them with array operations.
\[ \hat{X} = \arg \min_X \|X - Y\|^2 + (X - \mu)^T \Sigma^{-1} (X - \mu) \]

where, \( \mu = W^T \mu_c, \Sigma = W^T D_c W. \)

The solution is:

\[ \hat{X} = W^T (I + D_c^{-1})^{-1} W \ (Y + W^T D_c^{-1} \mu_c) \]

How would you code this up?
\[ \hat{X} = W^T (I + D_c^{-1})^{-1} W (Y + W^T D_c^{-1} \mu_c) \]

\[ = W^T (I + D_c^{-1})^{-1} (WY + D_c^{-1} \mu_c) \]

```python
xc = wvlt(Y)
xv += mu_c / sigma2c
xc /= (1 + 1/sigma2c)
X = invwvlt(xc)
```

Note that here mu_c and sigma2c are arrays the same size as xc.
More general optimization setting:
\[
\hat{X} = \arg \min_X D(X; Y) + R(X)
\]
\[
\hat{X} = \arg \min_X \sum_n \|X[n] - Y[n]\|^2 + \lambda \sum_n \left( ((G_x * X)[n])^2 + ((G_y * X)[n])^2 \right)
\]

Let $A_x$ and $A_y$ be matrices corresponding to convolution with $G_x$ and $G_y$.
\[
\hat{X} = \arg \min_X \|X - Y\|^2 + \lambda \left( \|A_x X\|^2 + \|A_y X\|^2 \right)
\]
\[
= \arg \min_X (X - Y)^T (X - Y) + \lambda X^T (A_x^T A_x + A_y^T A_y) X
\]

Let's change this to our standard quadratic form:
\[
= \arg \min_X X^T \left( I + \lambda (A_x^T A_x + A_y^T A_y) \right) X - 2X^T Y + Y^T Y
\]
And so,
\[
\hat{X} = \left( I + \lambda (A_x^T A_x + A_y^T A_y) \right)^{-1} Y
\]

How do you do this matrix inverse?
\[ \hat{X} = (I + \lambda(A_x^T A_x + A_y^T A_y))^{-1} Y \]

Remember, (circular) convolution is diagonalized in the Fourier domain!

\[ A_x = S \, D_x \, S^* \]

- Multiplication by \( S \) is the inverse Fourier transform
- Multiplication by \( S^* \) is the forward Fourier transform
- \( SS^* = I \)
- \( D_x \) is a diagonal matrix with the Fourier transform coefficients of \( G_x \)

\[ A_x^T A_x = A_x^* A_x = S \, D_x^* D_x S^* = S \, \|D_x\|^2 S^* \]

\[ A_y^T A_y = S \, \|D_y\|^2 S^* \]

\[ I + \lambda(A_x^T A_x + A_y^T A_y) = I + \lambda S(\|D_x\|^2 + \|D_y\|^2)S^* \]

\[ = S \left( I + \lambda \left( \|D_x\|^2 + \|D_y\|^2 \right) \right) S^* \]
What is this doing?

It’s down-weighting frequency components by a (real) factor where $\|D_x\|^2$ and $\|D_y\|^2$ are high.

Those are high for higher frequencies, because $G_x$ and $G_y$ are derivative filters.

So this operation down-weights higher frequency components ⇒ Smooths the image.
IMAGE RESTORATION

Denoising

\[
\hat{X} = \arg \min_X \sum_n \|X[n] - Y[n]\|^2 + \lambda \sum_n \left( (G_x * X)[n]^2 + (G_y * X)[n]^2 \right)
\]

De-blurring

Say we know that our image has been blurred by a known blur kernel \( k \)

\[
Y[n] = (X \ast k)[n] + \epsilon[n]
\]

\[
\hat{X} = \arg \min_X \sum_n \|(X \ast k)[n] - Y[n]\|^2 + \lambda \sum_n \left( (G_x * X)[n]^2 + (G_y * X)[n]^2 \right)
\]

\[
\hat{X} = \arg \min_X \|A_k X - Y\|^2 + \lambda \left( \|A_x X\|^2 + \|A_y X\|^2 \right)
\]

where \( A_k \) represents the action of convolution by blur kernel \( k \).

\[
\hat{X} = (A_k^T A_k + \lambda (A_x^T A_x + A_y^T A_y))^{-1} A_k^T Y
\]

Note that there is now \( A_k^T Y \) instead of just \( Y \).
De-blurring / De-convolution

\[
\hat{X} = \arg \min_X \|A_k X - Y\|^2 + \lambda \left( \|A_x X\|^2 + \|A_y X\|^2 \right)
\]

\[
= \left( A_k^T A_k + \lambda (A_x^T A_x + A_y^T A_y) \right)^{-1} A_k^T Y
\]

We can do this in the Fourier domain again (assuming \(A_k\) represents circular convolution).

\[
\hat{X} = S \left( \|D_k\|^2 + \lambda \left( \|D_x\|^2 + \|D_y\|^2 \right) \right)^{-1} S^* A_k^T Y
\]

\[
\hat{X} = S \left( \|D_k\|^2 + \lambda \left( \|D_x\|^2 + \|D_y\|^2 \right) \right)^{-1} S^* (SD_k^*S^*)Y
\]

\[
\hat{X} = S \left( \|D_k\|^2 + \lambda \left( \|D_x\|^2 + \|D_y\|^2 \right) \right)^{-1} D_k^* S^* Y
\]

\[
X_f[u, v] = \frac{K_f[u, v]^*}{\|K_f[u, v]\|^2 + \lambda \left( \|G_{xf}[u, v]\|^2 + \|G_{yf}[u, v]\|^2 \right)} Y_f[u, v]
\]

If \(\lambda = 0\), this reduces to:

\[
X_f[u, v] = \frac{K_f[u, v]^*}{\|K_f[u, v]\|^2} Y_f[u, v] = \frac{Y_f[u, v]}{K_f[u, v]}, \quad \text{as } \|K_f\|^2 = K_f K_f^*
\]
De-blurring / De-convolution

\[
\hat{X} = \arg \min_{X} \|A_k X - Y\|^2 + \lambda \left( \|A_x X\|^2 + \|A_y X\|^2 \right) \\
= \left( A_k^T A_k + \lambda (A_k^T A_x + A_k^T A_y) \right)^{-1} A_k^T Y
\]

\[
X_f[u, v] = \frac{K_f[u, v]^*}{\|K_f[u, v]\|^2 + \lambda \left( \|G_{xf}[u, v]\|^2 + \|G_{yf}[u, v]\|^2 \right)} Y_f[u, v]
\]

If \( \lambda = 0 \), this reduces to:

\[
X_f[u, v] = \frac{K_f[u, v]^*}{\|K_f[u, v]\|^2} Y_f[u, v] = \frac{Y_f[u, v]}{K_f[u, v]}, \text{ as } \|K_f\|^2 = K_f K_f^*
\]

But \( K_f[u, v] \) may be zero or close to zero, in which case dividing will amplify noise.

So the Fourier transform of the kernel \( k \) is telling us which frequency components are severely attenuated by the kernel.

The "regularization" with \( \lambda > 0 \) helps stabilize the inversion, because if \( K_f[u, v] \) is low for some \((u, v)\), then the factor will downscale the input coefficient \( Y[u, v] \).

This is called Wiener filtering or Wiener Deconvolution.
If you're going to use this method to do de-convolution, you will have to account for the fact that your observed image was a "valid" convolution, and not a 'circular' convolution.

A good approximation is to do what's called "edge tapering". First pad the image $Y$ to a bigger image, where you make values go down smoothly to 0.

Then do the deconvolution, and crop out the central part.
We discussed cases when you know of a basis (wavelet / Fourier) where you can diagonalize your quadratic system matrix, and have a closed form expression for its inverse.

Not always the case. What if you wanted to exactly model valid convolution (not approximate it as circular) ? What if you observed values at a subset of pixels ?

Generally, what if you wanted to compute $X = Q^{-1}Y$ for some arbitrary symmetric positive-definite $Q$. 
Let's consider a case where you can form $Q$.

- Never compute $Q^{-1}$, and then multiply by $Y$.
  - Numerically unstable, more expensive.
- Call `scipy.linalg.solve`:
  - Cholesky / LDL Decomposition: $Q = L D L^T$
  - Always exists for a positive definite matrix. $L$ is lower triangular.
  - Solve $Qx = b \rightarrow LDL^T x = b \rightarrow Ly = b, L^T x = D^{-1}y$

\[
\begin{bmatrix}
a & 0 & 0 & 0 & \cdots \\
q & c & 0 & 0 & \cdots \\
d & e & f & 0 & \cdots \\
\vdots & \vdots & \vdots & \vdots & \ddots \\
\end{bmatrix}
\begin{bmatrix}
y_1 \\
y_2 \\
y_3 \\
\vdots \\
\end{bmatrix}
= 
\begin{bmatrix}
b_1 \\
b_2 \\
b_3 \\
\vdots \\
\end{bmatrix}
\]
More generally, when $Q = A_1^T A_1 + A_2^T A_2 + \ldots$, where $A_1, A_2, \ldots$ are sparse operations, that involve convolutions and element-wise operations.

- If $A_1$ is convolution with $k$, then you can get the effect of multiplying $A_1^T A_1$ with an image $Y$ by
  - Convolving with $k$ first
  - Then, convolving the result with the flipped version of $k$
- If $A_1$ is valid convolution, $A_1^T$ will correspond to "full" convolution with flipped version of $k$.
- If $A_2$ is convolution with $k$ followed by element-wise multiplication with a mask image, then $A_2^T A_2$ is
  - Convolution with $k$
  - Multiply by mask
  - Multiply by mask again
  - Convolution with flipped version of $k$
- So even when we can't form $Q$, we can carry out the actions $QY$, as well as $Z^T QY$
  - Compute $QY$
  - Take element-wise product of the result with $Z$ and sum.
Solve by the Conjugate Gradient method.

- Generic algorithm for solving $Qx = b$ for symmetric positive definite $Q$.
- Useful when you can multiply by $Q$ but not 'form' it.

**Basic Idea**

- For a given set of vectors $\{p_1, p_2, \ldots, p_N\}$
  - that are same size as $x$
  - linearly independent
  - $N = \text{dimensionality of } x$
- We can write any $x = \sum_i \alpha_i p_i$
- If we also choose the vectors to be 'conjugate' such that $p_i^T Q p_j = 0$ for $i \neq j$:

$$Qx = b \rightarrow p_k^T Qx = p_k^T b \rightarrow \alpha_i p_k^T Q p_k = p_k^T b \rightarrow \alpha_i = \frac{p_k^T b}{p_k^T Q p_k}$$
CONJUGATE GRADIENT

Iterative Algorithm

- Begin with some guess $x_0$ for $x$ (say all zeros)
- $k = 0$, $r_0 \leftarrow b - Qx_0$, $p_0 \leftarrow r_0$
- Repeat
  - $\alpha_k \leftarrow r_k^T r_k / p_k^T Q p_k$
  - $x_{k+1} = x_k + \alpha_k p_k$
  - $r_{k+1} = r_k - \alpha_k Q p_k$
  - $\beta_k = r_{k+1}^T r_{k+1} / r_k^T r_k$
  - $p_{k+1} = r_{k+1} + \beta_k p_k$
  - $k = k + 1$


Think about what you would do when: $Q = (A_k^T A_k + \lambda (A_x^T A_x + A_y^T A_y))$, $b = A_k^T Y$
$r[n] = \int L(\lambda, n)\Pi_r(\lambda)d\lambda$

g[n] = \int L(\lambda, n)\Pi_g(\lambda)d\lambda

b[n] = \int L(\lambda, n)\Pi_b(\lambda)d\lambda
Simple View:
Total / Average Intensity in "Green Part" of the spectrum.
Color

Metamers: Different L that have the same measured RGB values.

Simple View:
Total / Average Intensity in "Green Part" of the spectrum.
For simplicity,
Have discrete wavelengths
Approximate integration as summation

\[ r[n] = \int L(\lambda, n) \Pi_r(\lambda) d\lambda \]
\[ g[n] = \int L(\lambda, n) \Pi_g(\lambda) d\lambda \]
\[ b[n] = \int L(\lambda, n) \Pi_b(\lambda) d\lambda \]

\[ L(\lambda, n) \rightarrow L[\lambda, n] \text{ or } L[n] \in \mathbb{R}^B \]
\[
\begin{align*}
    r[n] &= \langle L[n], \Pi_r \rangle \\
    g[n] &= \langle L[n], \Pi_g \rangle \\
    b[n] &= \langle L[n], \Pi_b \rangle \\
\end{align*}
\]

Think of the incident light being a B (\(\gg 3\)) channel image \(L[n]\)

\[
L(\lambda, n) \rightarrow L[\lambda, n] \text{ or } L[n] \in \mathbb{R}^B
\]
There are cameras that actually capture such "hyperspectral" images.
\[ X[n] = \Pi^T L[n], \]
\[ \Pi = [\Pi_r \quad \Pi_g \quad \Pi_b] \]

(B x 3 Matrix)

Think of the incident light being a B (>> 3) channel image \( L[n] \)

- 3 Dimensional Projection from higher dimensional space
- Invariant to changes in the "null space" of \( \Pi \)