Problem Session 6
Topics

• Photo-electron shot-noise
• SNR calculations
• Deconvolution of an image with Poisson noise
  • Wiener deconvolution
  • Richardson-Lucy
  • Richardson-Lucy + TV prior
• Simulation of a single pixel camera
  • Least norm
  • ADMM + TV prior
  • ADMM + NLM
Photo-electron shot-noise

• Shot noise occurs during:
  1. Photon counting in optical devices, and associated with the particle nature of light.
  2. The conversion of photons to electrons

• There is statistical variability (=noise) when counting photo-electrons during a specific amount of time.

• Can be modelled by Poisson statistics.

• Special property of Poisson statistics:

  The variance equals the average number of events \( (N) \)

  \[ \text{Signal to noise (SNR)} = \frac{N}{\sqrt{N}} = \sqrt{N} \]
Task 1: SNR calculations

You’re asked to compare the SNR of two cameras:
• Consumer camera
• sCMOS

At different acquisition modes:
• Flutter shutter
• Burst

Calculate the SNR and discuss your results.
SNR calculations

**Consumer camera**: Lots of photons, moderate additive Gaussian noise.

**CMOS microscope sensor**: photon-limited imaging, only a few photons. The sensor is cooled, so no Gaussian noise, but there is photo-electron shot-noise, which has a Poisson distribution.
SNR calculations

**Flutter shutter**: temporally modulated aperture pattern. Used, for example, for better motion deblurring (see “Coded Exposure Photography: Motion Deblurring using Fluttered Shutter”, Raskar et al.). The result is a single image.

**Burst**: acquires multiple short-exposure images.
SNR calculations

\[
\text{SNR} = \frac{\text{mean number of photons}}{\text{standard deviation of noise}} = \frac{\mu}{\sigma}
\]

- Adding \( m \) signals of strength \( k \Rightarrow mk \)
- Adding independent Gaussian distributions
  \[ G(\mu_1, \sigma_1^2) + G(\mu_2, \sigma_2^2) \sim G(\mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2) \]
- Adding independent Poisson distributions
  \[ \text{Pois}(\lambda_1) + \text{Pois}(\lambda_2) \sim \text{Pois}(\lambda_1 + \lambda_2) \]

Variance = (standard deviation)\(^2\)
Both mean and variance
SNR calculations

For each of the four cases, calculate:
- The average number of photons (the signal)
- The standard deviation of the noise
- and divide them

Don’t forget to describe your results and conclusions.
Task 2: Poisson deconvolution

You’re provided with the original image of a cell and code for blurring it and adding Poisson noise.

Your task is to sharpen (deconvolve) the image using:

1. Wiener filtering
2. Richardson-Lucy
3. Richardson-Lucy + a TV prior (with several regularization factors $\lambda$)

Show the resulting image, the convergence of the residual and the MSE and calculate the PSNR.
Poisson deconvolution

Note:
Deconvolution as we saw in week 3 (using ADMM) will not work well. This is because the minimization objective (L2 norm) is optimal for Gaussian noise, not Poisson noise. See notes!
If you want to try ADMM for the Poisson case, complete the bonus question (worth 25 points!).
Poisson deconvolution

Mean squared error (MSE) and the peak signal-to-noise ratio (PSNR):

$$MSE = \frac{1}{3mn} \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{c=1}^{3} \left[I_{\text{original}}(i, j) - I_{\text{restored}}(i, j)\right]^2$$

$$PSNR = 10 \log_{10} \left( \frac{\max(I_{\text{original}})^2}{MSE} \right)$$

Use 1 if images are normalized to 1
Poisson deconvolution

Wiener deconvolution:

• Implement in the Fourier domain

$$H' = \frac{1}{H} \cdot \frac{|H|^2}{(|H|^2 + 1/SNR)} \quad SNR = \frac{\bar{I}}{\sigma_{noise}}$$

• What is $\sigma_{noise}$ for an image with Poisson noise? Consider the properties of the Poisson noise distribution...
Wiener deconvolution

Original

With blurring and noise

Wiener deconvolution

Still blurry...
Poisson deconvolution

**Maximum likelihood**

We want to find $x$ (the original image) such that it maximizes the probability of $b$ (our noisy & blurry image) given $x$, knowing the blur kernel and assuming Poisson noise.

$$b \sim \mathcal{P}(Ax)$$
Poisson deconvolution

Maximum likelihood
The probability of a measurement (pixel) \( i \) is given by

\[
p(b_i|x) = \frac{(Ax)^i e^{-(Ax)_i}}{b_i!}
\]

\( Ax \) is how the image appears on the sensor
\( \rightarrow \) It determines the average number of photons accumulated in the sensor

Poisson PMF

\[
P(k \text{ events in interval}) = \frac{\lambda^k e^{-\lambda}}{k!}
\]

where
- \( \lambda \) is the average number of events per interval
- \( e \) is the number 2.71828... (Euler's number) the base of the natural logs
- \( k \) takes values 0, 1, 2, ...
- \( k! \) is the factorial of \( k = k \times (k-1) \times (k-2) \times \ldots \times 2 \times 1 \)

This equation is the probability mass function (PMF) for a Poisson distribution.
Poisson deconvolution

Maximum likelihood

The joint probability of all measurements (pixels) is expressed by

\[ p(b | x) = \prod_{i=1}^{M} p(b_i | x) = \prod_{i=1}^{M} \frac{e^{\log((Ax)_i)b_i} e^{-(Ax)_i}}{b_i!} \]

and the log likelihood is:

\[ L(x) = \log(p(b | x)) = \log((Ax)^T b) - (Ax)^T 1 - \sum_{i=1}^{M} \log(b_i!) \]

We want to maximize \( L(x) \)!
Poisson deconvolution

The Richardson – Lucy algorithms

Assume we have some initial guess of our unknown image ($x$), we can apply an iterative correction to the image which will be converged at iteration $q$ if the estimate does not change after further iterations, i.e.

$$\frac{x^{(q+1)}}{x^{(q)}} = 1.$$ 

At this estimate, the gradient of the objective ($L(x)$) will also be zero. The gradient is calculated using matrix calculus.

$$\nabla L(x) = A^T \text{diag}(Ax)^{-1} b - A^T 1 = 0$$

Places the elements of $Ax$ on the diagonal 

Sum over the rows of $A$ (multiplication with ones)
Poisson deconvolution

**Richardson – Lucy**

\[
\frac{x^{(q+1)}}{x^{(q)}} = 1
\]

\[
\nabla L(x) = A^T \text{diag}(Ax)^{-1} b - A^T 1 = 0 \Rightarrow \text{diag}(A^T 1)^{-1} A^T \text{diag}(Ax)^{-1} b = 1
\]

\[
x^{(q+1)} = 1 \cdot x^{(q)} = (\text{diag}(A^T 1)^{-1} A^T \text{diag}(Ax)^{-1} b) \cdot x^{(q)}
\]

This is what you’ll implement

For any initial guess that is purely positive \((x^{(0)} > 0)\), the following iterations will also remain positive. Non-negativity of the estimated image is an important requirement for most imaging applications.
Poisson deconvolution

Richardson-Lucy

Very noisy!!
What should we do?

Add a regularization term (prior)!
Poisson deconvolution

Richardson – Lucy with TV prior

We add the TV term $(\lambda \|Dx\|_1)$ to $L(x)$. After derivation (see notes) we get this update equation:

$$x^{(q+1)} = \frac{A^T \left( \frac{b}{Ax} \right)}{A^T 1 - \lambda \left( \frac{D_{xx}x}{|D_{xx}|} + \frac{D_{yx}}{|D_{yx}|} \right)} \cdot x^{(q)}$$

You can get NaNs if you have values of $Ax == 0$, -> $b/(Ax) == \text{Inf}$. Just set NaN values to zero after calculating $b/(Ax)$.
Richardson-Lucy deconvolution

Original

Richardson-Lucy

Richardson-Lucy + TV prior
Poisson deconvolution

Not part of the HW
Poisson deconvolution

The residual is the log likelihood $L(x)$

For RL:

$$\log (Ax)^T b - (Ax)^T 1 - \sum_{i=1}^{M} \log (b_i!)$$

For RL + TV:

$$\log (Ax)^T b - (Ax)^T 1 - \sum_{i=1}^{M} \log (b_i!) - \lambda \|Dx\|_1$$

You can get NaNs if you have values of $Ax = 0 \rightarrow \log(0) = -\infty$.
Just set NaN values to zero after multiplying $\log(Ax)$ and $b$, but before taking the sum.
Poisson deconvolution

Hints:
• You are given a template which constructs the image. Read it and understand it.
• Start with 100 iterations for RL and RL+TV
• Perform the solution per channel (for loop on the colors)
• Initial guess for x can be anything positive (try uniform or random [0,1]: rand(imageSize))
• Don’t use matrix multiplications! \(Ax\) and \(A^Tx\) should be implemented as functions in Fourier domain.
• Implement point-wise division where needed.
• If you get NaNs, make them zero (a(isnan(a))==0)
• If you get negative values, make them zero (a(a<0)==0)
• We Provide a function for calculating \(D_x, D_y\) (opDx.m)

% define function handle for 2D convolution
Afun = @(x) ifft2( fft2(x).*otf, 'symmetric');
Atfun = @(x) ifft2( fft2(x).*conj(otf), 'symmetric');
Task 3: Single pixel camera
From denoising to solving the single-pixel camera

In the notes of lecture 10, we solve image denoising by maximizing the a-posteriori estimate of $x$ (MAP estimation).

With Gaussian noise and $\Gamma(x)$ as the prior, we derive that:

$$x_{\text{MAP}} = \arg \min_{\{x\}} \frac{1}{2\sigma^2} \|b - x\|_2^2 - \log(p(x))$$

Our prior $\Gamma(x)$ $p(x)$ can be interpreted as a prior on the latent image $x$.

For a generic reconstruction problem, we have an operator, $A$, in front of $x$. 

From denoising to solving the single-pixel camera

For a **self-similarity prior**, we can use non-local-means denoising:

\[
x_{\text{NLM}} = \arg \min_{\{x\}} \frac{1}{2\sigma^2} \| b - x \|_2^2 + \Gamma(x) = \text{NLM} \left( b, \sigma^2 \right)
\]

(This is more of a heuristic, or we can define NLM to be the solution to the denoising problem with a self-similarity prior)

Here the argument is the variance, but notice that in some implementations the argument will be the standard deviation
From denoising to solving the single-pixel camera

Now let’s look at the notes of lecture 11, the reconstruction problem is:

$$\min_{\{x\}} \frac{1}{2} \|Ax - b\|_2^2 + \lambda \Gamma(x).$$

A could be a blurring kernel or any other operator. 
$\Gamma(x)$ is the regularization which can be sparse gradients (TV prior) or self-similarity.
From denoising to solving the single-pixel camera

Formulating as an ADMM optimization problem (K is a general linear operator):

$$\min_{\{x\}} \underbrace{\frac{1}{2} \|Ax - b\|_2^2}_{f(x)} + \underbrace{\lambda \Gamma(z)}_{g(z)}$$

subject to $Kx - z = 0$

Now we will go through a generalized ADMM solution
Formulating as an ADMM optimization problem (K is a general linear operator):

\[
\begin{aligned}
\text{minimize} & \quad \frac{1}{2} \| Ax - b \|_2^2 + \lambda \Gamma (z) \\
\text{subject to} & \quad Kx - z = 0
\end{aligned}
\] (3)

Following the general ADMM strategy, the Augmented Lagrangian of Equation 3 is formed as

\[
L_\rho (x, z, y) = f (x) + g (z) + y^T (Kx - z) + \frac{\rho}{2} \|Kx - z\|_2^2
\] (4)

As discussed in more detail in Chapter 3.1 of [Boyd et al. 2001], using the scaled form of the Augmented Lagrangian, the following iterative updates rules can be derived:

\[
x \leftarrow \text{prox}_{\frac{1}{2} \rho} (v) = \arg \min_x L_\rho (x, z, y) = \arg \min_x \frac{1}{2} \| Ax - b \|_2^2 + \frac{\rho}{2} \|Kx - v\|_2^2, \quad v = z - u
\]

\[
z \leftarrow \text{prox}_{\lambda} (v) = \arg \min_z L_\rho (x, z, y) = \arg \min_z \lambda \Gamma (z) + \frac{\rho}{2} \|v - z\|_2^2, \quad v = Kx + u
\]

\[
u \leftarrow u + Kx - z
\]

where \( u = \frac{1}{\rho} y \). The \( x \) and \( z \)-updates are performed via the proximal operators \( \text{prox}_{\cdot, \rho} \). The interested reader is referred to [Boyd et al. 2001] for more details.
X – update

\[ \text{prox}_{\|x\|_2, \rho} (v) = \arg \min_{\{x\}} \frac{1}{2} \|Ax - b\|_2^2 + \frac{\rho}{2} \|Kx - v\|_2^2, \quad v = z - u \]

\[ \text{prox}_{\|x\|_2, \rho} (v) = \left( \underbrace{A^T A + \rho K^T K}_{\tilde{A}} \right)^{-1} \left( \underbrace{A^T b + \rho K^T v}_{\tilde{b}} \right) \]

We can use an iterative solver to compute this (see following slides).
Z – update

\[ z \leftarrow \text{prox}_{\Gamma,\rho} (v) = \arg \min_{\{z\}} L_\rho (x, z, y) = \arg \min_{\{z\}} \lambda \Gamma (z) + \frac{\rho}{2} \| v - z \|^2_2, \quad v = Kx + u \]

Looks exactly like the denoising formulation!

We can use any Gaussian denoising algorithm (MAP estimator for Gaussian noise) as the proximal operator!

That’s the big insight!

We can use denoising algorithms for image reconstruction

Total variation prior: \( \Gamma (x) = \| Dx \|_1 \)
The gradients are sparse

Self similarity prior:
Denoise with NLM

This function matches the prior:
z=Dx, the gradients, can have high values, but low values go to zero

\[ \text{prox}_{\Gamma,\rho} (v) = \text{NLM} \left( v, \frac{\lambda}{\rho} \right) \]
Single pixel camera

Formulating the problem:

Image formation for a noisy single-pixel camera:

\[ b = Ax + \eta, \]

where \( \eta \) is an additive, signal-independent Gaussian noise term with standard deviation \( \sigma \), \( b \in \mathbb{R}^{M} \), \( x \in \mathbb{R}^{N} \), and \( A \in \mathbb{R}^{M \times N} \).

For compression factors \( \frac{N}{M} = 1, 2, 4, \) and \( 8 \) \((M < N)\)

We want to reconstruct \( x \).
Single pixel camera

Formulating the problem:

\[ b = Ax + \eta, \]
\[ A \in \mathbb{R}^{M \times N}, M < N \]

→ More variables \((N)\) than equations \((M)\)

→ The system is undetermined, and there are infinite solutions

→ A prior will help us obtain a reasonable solution
  1. Least norm solution
  2. TV prior
  3. Self-similarity prior
Single pixel camera: Least norm solution

Use a prior that is the \( \ell_2 \)-norm on the image: \( \|x\|_2^2 \)

The reconstruction problem is

\[
\begin{aligned}
\min_{\{x\}} & \quad \frac{1}{2} \|Ax - b\|_2^2 + \|x\|_2^2 \\
\end{aligned}
\]

The solution is obtained by derivation, and the least norm solution is

\[
x_{ln} = A^T(AA^T)^{-1}b
\]
The conjugate gradient method

- An algorithm for the numerical solution of particular systems of linear equations, namely those whose matrix is symmetric and positive-definite: $Cx = b$, $C^T = C$, $x^T C x > 0$ for all $x \neq 0 \Rightarrow x = C^{-1} b$
- Used to calculate $(AA^T)^{-1} b$
- Useful for large matrices, when we want quick convergence
- Use the Matlab function pcg (Preconditioned Conjugate Gradients Method)
- Matlab’s pcg function also supports matrix-free operations, meaning, you can supply a function handle that computes $C x$ without forming the matrix $C$
- $x = \text{pcg}(C, b, \text{Tolerance}, \text{Max\_Iteration})$
Single pixel camera: Least norm solution

Results are not great...
Single pixel camera: ADMM + TV prior

Use a prior that is sparse gradients and minimize the TV norm

\[
\begin{align*}
\text{minimize} \quad & \frac{1}{2} \| Ax - b \|_2^2 + \lambda \| Dx \|_1 \\
\{ x \} \quad & \text{regularization}
\end{align*}
\]

\[
\Gamma(x) = \lambda \| Dx \|_1, \quad \text{with} \quad D = \begin{bmatrix} D_x^T & D_y^T \end{bmatrix}^T. \quad D \in \mathbb{R}^{2N \times N}
\]

Steps are similar to HW3, only the update step for \( x \) is different
Single pixel camera: ADMM + TV prior

$$\text{prox}_{f,\rho}(v) = \left( \frac{A^T A + \rho D^T D}{\tilde{A}} \right)^{-1} \left( \frac{A^T b + \rho D^T v}{\tilde{b}} \right)$$

$$\tilde{A}x = \tilde{b}$$

See notes for derivation!
Single pixel camera: ADMM + TV prior

Compression factor = 1

Compression factor = 2

Experiment with other values of $\lambda$, $\rho$
Single pixel camera: ADMM + TV prior

Compression factor = 4, $\lambda = 1$

Compression factor = 4, $\lambda = 10$

Experiment with other values of $\lambda, \rho$
Single pixel camera: ADMM + TV prior

Compression factor = 8, $\lambda = 1$

Compression factor = 8, $\lambda = 10$

Experiment with other values of $\lambda, \rho$
Hints for task 3 a, b

• Images are grayscale. 3rd dimension can be used for the N masks.
• Create N random masks (use rand), where N=(# of pixels in image)/(compression)
• Define functions for A, \( A^T \), and \( AA^T \):
  • \( A \) performs element-wise multiplication of the 2D input \( x \) (which has N elements) with M 2D masks, and sums the results. The output is a vector size M. \( (b = Ax) \)
  • \( A^T \) multiplies the M masks with the M values of the input and sums the results. The output is a 2D matrix with N elements. \( (A^T b) \)
  • \( AA^T \) can be defined using the previous functions.
• Noise is added after applying the masks: \( b = A \text{fun}(I) + \sigma \cdot \text{randn([N 1])} \);
• For conjugate gradients, use 50 iterations and a tolerance of \( 10^{-12} \)
• For ADMM, use the provided functions: \( \text{opDx}, \text{opDtx} \), which work on non-vectorized images.
• You can also define a handle for \( D^T D \): \( \text{opDtDx} = @(x) \text{opDtx(opDx(x))} \);
• What is the function handle for CG? Notice reshaping the function to the correct dimensions!
• The residual for ADMM is \( \frac{1}{2} \|Ax - b\|_2^2 + \lambda \|Dx\|_1 \), plot the log residual
Single pixel camera: ADMM + NLM

Compression factor = 1, \( \lambda = 0.1 \)

Compression factor = 2, \( \lambda = 0.1 \)

Experiment with other values of \( \lambda, \rho \)
Single pixel camera: ADMM + NLM

Compression factor = 4, $\lambda = 0.1$

Compression factor = 8, $\lambda = 0.1$

Experiment with other values of $\lambda, \rho$
Comparison

<table>
<thead>
<tr>
<th>Compression</th>
<th>PSNR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1X</td>
<td>51.18 dB</td>
</tr>
<tr>
<td>2X</td>
<td>43.65 dB</td>
</tr>
<tr>
<td>4X</td>
<td>17.95 dB</td>
</tr>
<tr>
<td>8X</td>
<td>15.68 dB</td>
</tr>
</tbody>
</table>

TV → target image

NLM →
Hints for task 3 c

• K is unity ($\rho K^T K = \rho, \rho K^T v = \rho v$)

• For a fast NLM implementation use the code attached to the problem set (based on: https://www.mathworks.com/matlabcentral/fileexchange/27395-fast-non-local-means-1d--2d-color-and-3d?s_tid=srchtitle)

• You’ll need to compile the c files to Matlab

• Use function: NLMF.m (read the documentation)

• Use the default NLM parameters, and only change the filter strength:

  kappa = lambda/rho;
  v = x+u;
  Options.filterstrength=sqrt(kappa);
  z = NLMF(v,Options);

• The residual is $\frac{1}{2} \|Ax - b\|_2^2$. It does not decrease monotonically because it’s not a convex problem, but final residual should be much lower than initial one
Single pixel camera: section d

• Apply ADMM + TV and ADMM + NLM to the provided 12 images
• For ADMM + TV: use $\lambda=1$, $\rho=10$
• For ADMM + NLM: use $\lambda=0.1$, $\rho=10$
• Present your results as a composite image (see next slides)
• Also present a table with the final PSNR values
• Discuss the results
Single pixel camera: ADMM + TV prior

<table>
<thead>
<tr>
<th>Compression factor</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>34.30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>27.98</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>24.79</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>21.24</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Single pixel camera: ADMM + NLM

**PSNR table**

<table>
<thead>
<tr>
<th>Compression factor</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>32.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>29.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>25.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>22.9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Have a nice weekend!

And good luck with the homework!