

# Problem Session 3

# Topics

- Image Filtering
  - Spatial domain vs. Fourier Domain
  - Low pass and high pass
- Deconvolution and Inverse Filtering
  - Fourier-based
    - Standard
    - Wiener Deconvolution
  - Gradient Descent

# Task 1: Image filtering

Primal domain vs. Fourier domain

- Primal:  $I(x, y) \rightarrow I(x, y) * PSF(x, y)$   
Point spread function
- Fourier domain:  $\tilde{I}(\omega_x, \omega_y) \rightarrow \tilde{I}(\omega_x, \omega_y) \times OTF(\omega_x, \omega_y)$   
Optical transfer function

# Task 1: Image Filtering

- Primal Domain vs. Fourier domain
- **Helpful functions:** `scipy.signal.convolve2d`, `pypher.psf2otf`, `numpy.fft.fft2`, `numpy.fft.ifft2`
- Notice that the time of the convolution increases as the PSF becomes larger, while the time of the Fourier domain computation remains similar and independent of kernel size
- Normalize the filter so it sums to 1
- You can implement high pass filtering as:

$$I - I * PSF_{LP}$$

Primal Domain

$$\tilde{I} \times (1 - OTF_{LP})$$

Fourier Domain

# Task 1: Image filtering

Example of results:

- Primal and dual results look similar

Spatial blur with  $\sigma=15$



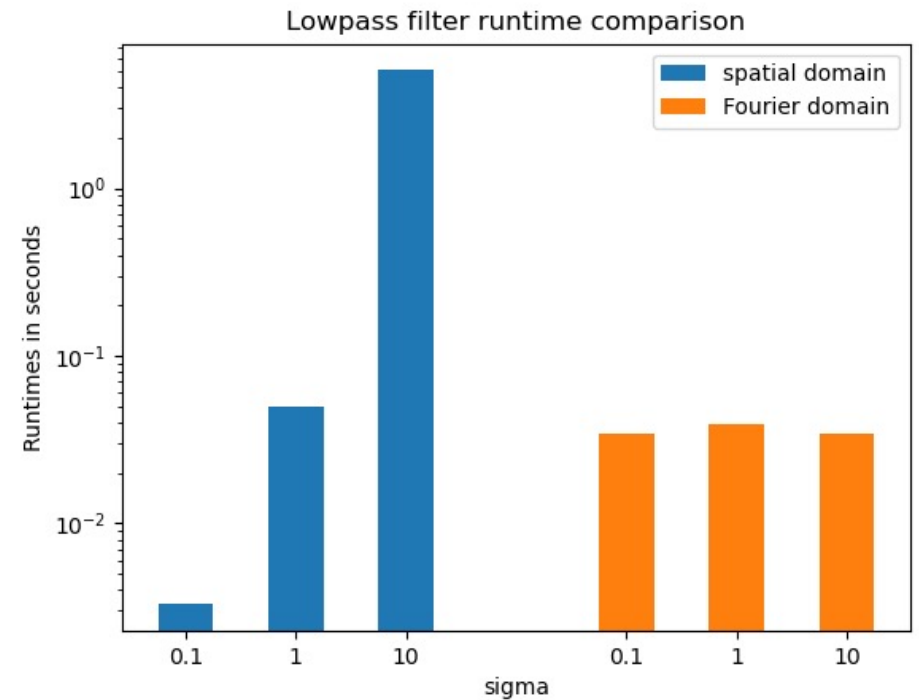
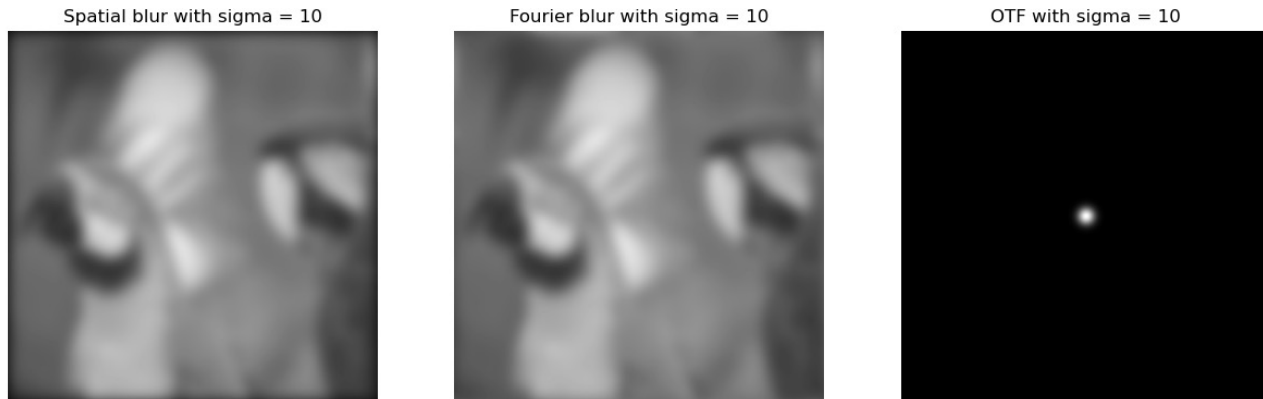
Fourier blur with  $\sigma=15$



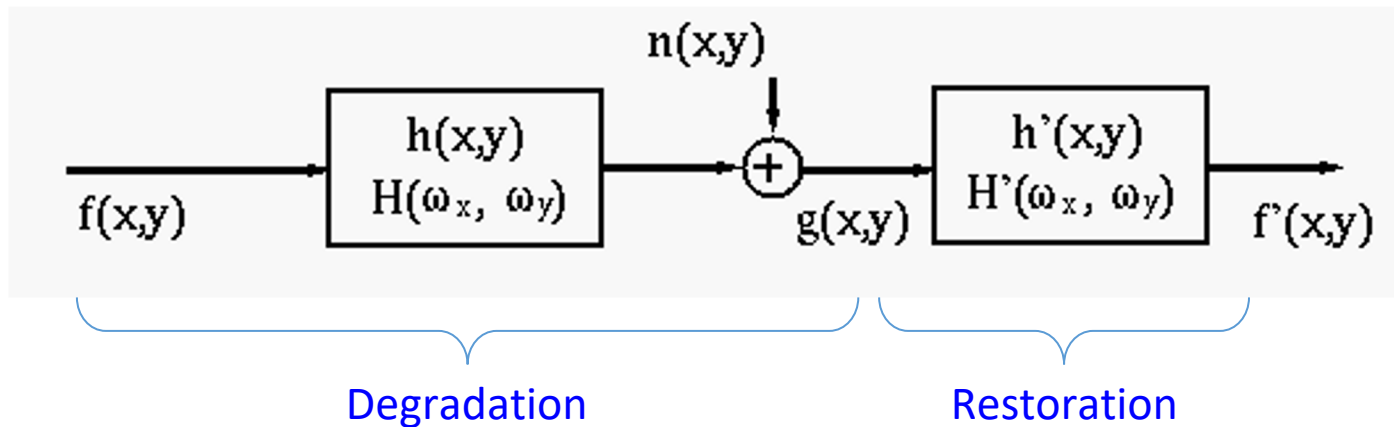
# Task 1: Image filtering

Why are we seeing this behavior?

Example of results



# Task 2: Deconvolution and Inverse Filtering

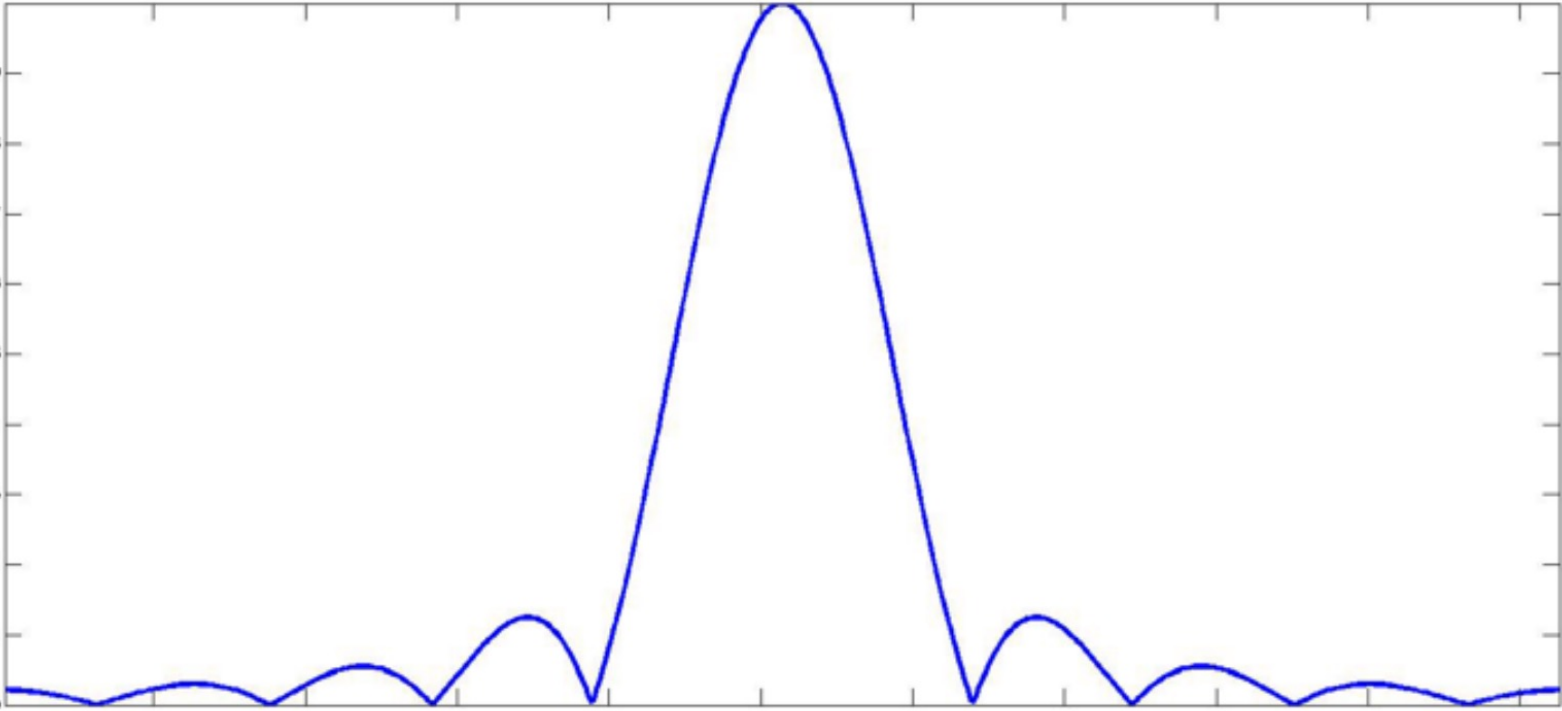
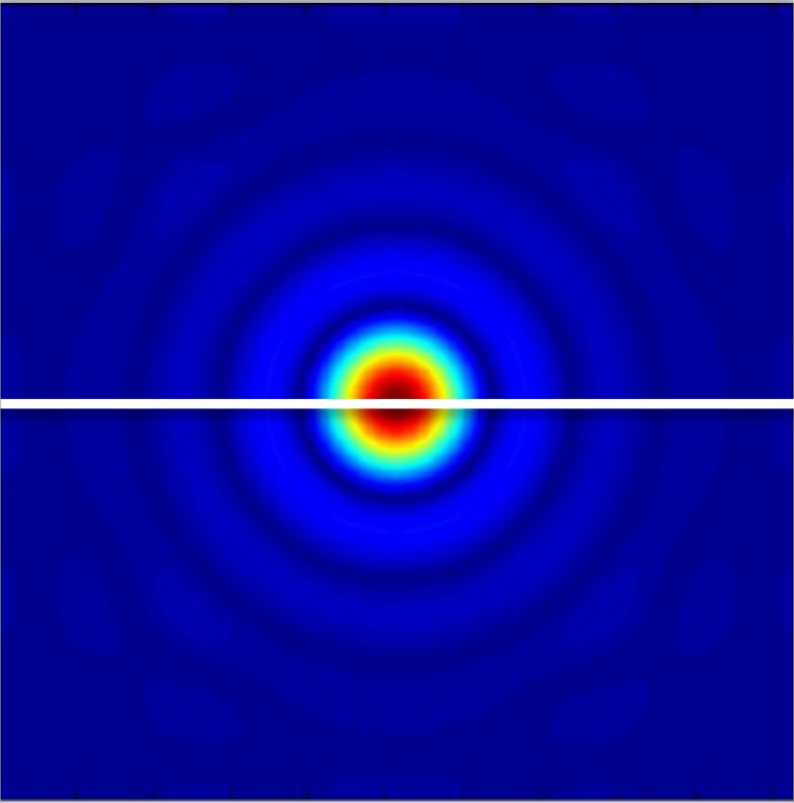


What is the best  $h'$  (or  $H'$ )?

Simply using  $H' = 1/H$  will (usually) amplify noise and destroy the image. **Why?**

# Task 2: Deconvolution and Inverse Filtering

OTF





# Task 2: Deconvolution and Inverse Filtering

For HW:

- First, blur the image with a Gaussian kernel (primal or Fourier domain)
- Add random noise:  $I = I + \text{sigma} \cdot \text{randn}(\text{size}(I))$  ;
- Reconstruct the image by
  1. Dividing by the blur kernel (OTF) in Fourier domain (simple inverse filtering)
  2. Wiener deconvolution, which is almost the same as inverse filtering, but uses a damping factor in the Fourier domain that depends on the noise

$$H' = \frac{1}{H} \cdot \frac{|H|^2}{(|H|^2 + 1/SNR)}$$

$$SNR = \frac{\bar{I}}{\sigma_{noise}}$$

Average pixel value of noisy image

# Task 2: Results

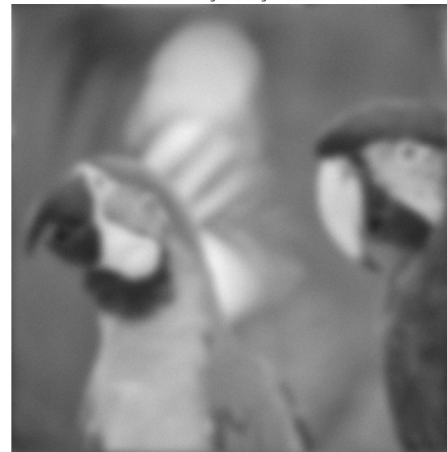
Blurred image with sigma = 0



Wiener filtering, PSNR = 106.3 dB



Blurred image with sigma = 0.01



Wiener filtering, PSNR = 25.9 dB



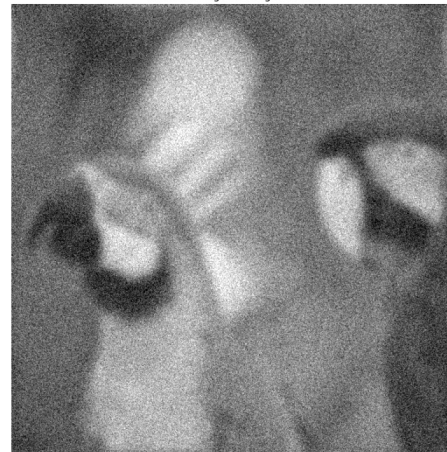
Blurred image with sigma = 0.001



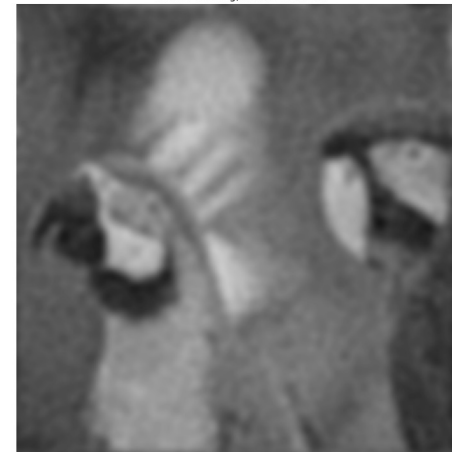
Wiener filtering, PSNR = 26.6 dB



Blurred image with sigma = 0.1



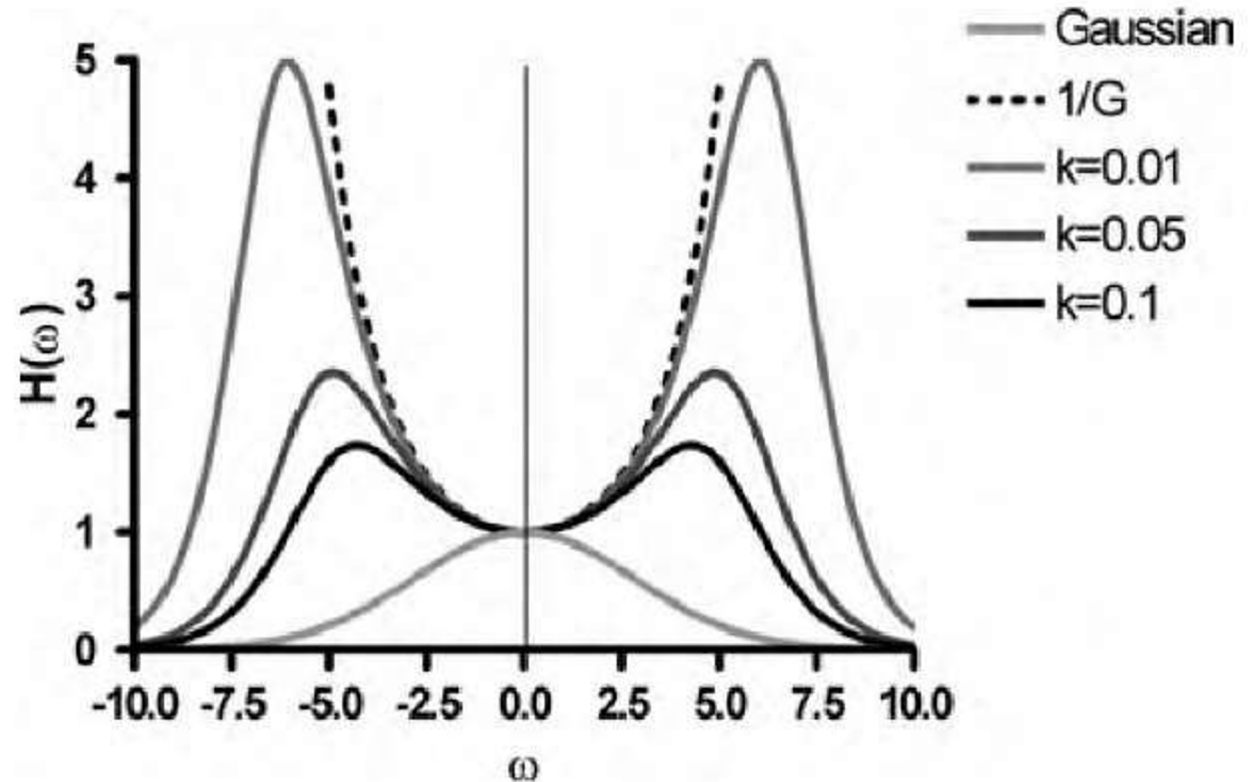
Wiener filtering, PSNR = 19.7 dB



# Task 2: Deconvolution and Inverse Filtering

Frequency response of a Wiener filter

$$G' = \frac{1}{G} \cdot \frac{|G|^2}{(|G|^2 + k)}$$



Higher Noise  $\rightarrow$  Lower SNR  $\rightarrow$  More damping  $\rightarrow$  less noise amplification

# Task 2: Deconvolution and Inverse Filtering

The Wiener filter is the solution ( $x$ ) that minimizes the mean square error between the image and its estimation:

$$E\|x - \hat{x}\|_2^2$$

an analytical derivation results in the Wiener filter.

# Task 2: Deconvolution and Inverse Filtering

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

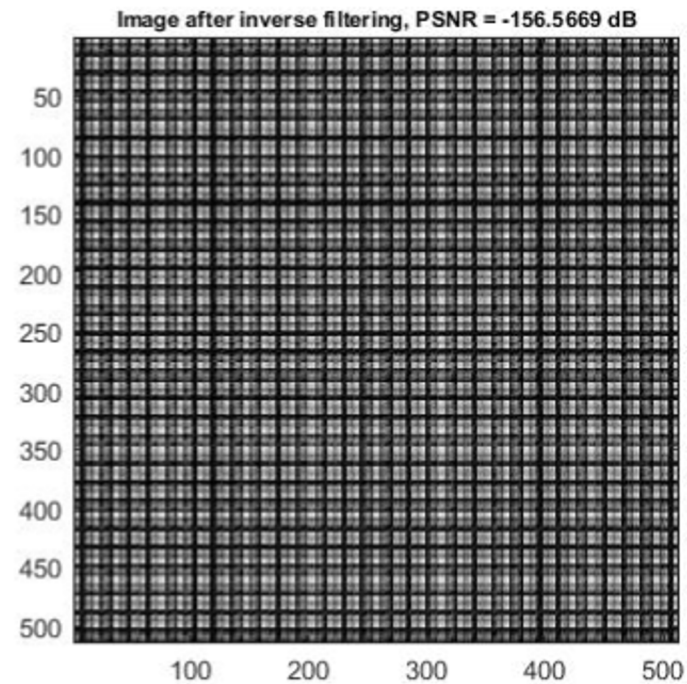
$$MSE = \frac{1}{mn} \sum_{i=1}^m \sum_{j=1}^n [I_{original}(i, j) - I_{restored}(i, j)]^2$$

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

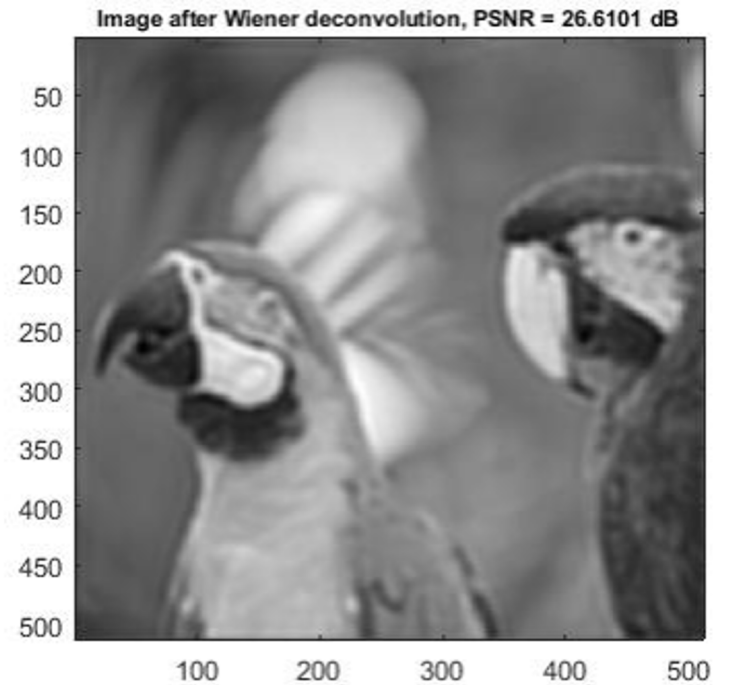
# Task 2: Deconvolution and Inverse Filtering

Example of results

Inverse



Wiener deconvolution



# Task 3: Gradient Descent

- A general algorithm for solving an optimization problem of the form

$$\underset{x}{\text{minimize}} \quad f(x)$$

- Idea: Move in the direction of the **negative gradient**
  - the direction in which the function is most steeply decreasing
  - Alpha ( $\alpha$ ) is the step size (or the “learning rate”)

$$x^{(k+1)} \leftarrow x^{(k)} - \alpha \nabla f(x^{(k)})$$

# Task 3: Gradient Descent

- Apply to the equation

$$\underset{x}{\text{minimize}} \quad \frac{1}{2} \|Ax - b\|_2^2$$

- A: Linear operator representing the image formation model (or **forward model**)
- b: Observed measurements (noisy image)
- x: Desired reconstruction variable



# Task 3: Gradient Descent

Residual  $\frac{1}{2} \|Ax - b\|_2^2 = \frac{1}{2} x^T A^T Ax - x^T A^T b + \frac{1}{2} b^T b$

$\nabla_x \left[ \frac{1}{2} \|Ax - b\|_2^2 \right] = A^T Ax - A^T b$  Gradient

```
def grad_l2(A, x, b):  
    # TODO: return the gradient of 0.5 * ||Ax - b||_2^2  
    return None  
  
def residual_l2(A, x, b):  
    return 0.5 * np.linalg.norm(A @ x - b)**2
```

@ = matrix multiply

$$x^{(k+1)} \leftarrow x^{(k)} - \alpha \nabla f(x^{(k)})$$

# Task 3: Stochastic Gradient Descent

- General case:  $x^{(k+1)} \leftarrow x^{(k)} - \alpha g(x^{(k)})$        $\mathbb{E}[g(x)] = \nabla f(x)$
- In the context of least squares, can express the objective as a sum of scalar residuals:
 
$$\|Ax - b\|_2^2 = \sum_{i=1}^n (a_i^T x - b_i)^2$$
- Choosing a **subset of rows** of A and b === descending on a **subset of these residuals**
  - The number of rows is the **batch size**.
- Use `np.random.randint` to select random indices for the A matrix

# Task 3: Gradient Descent

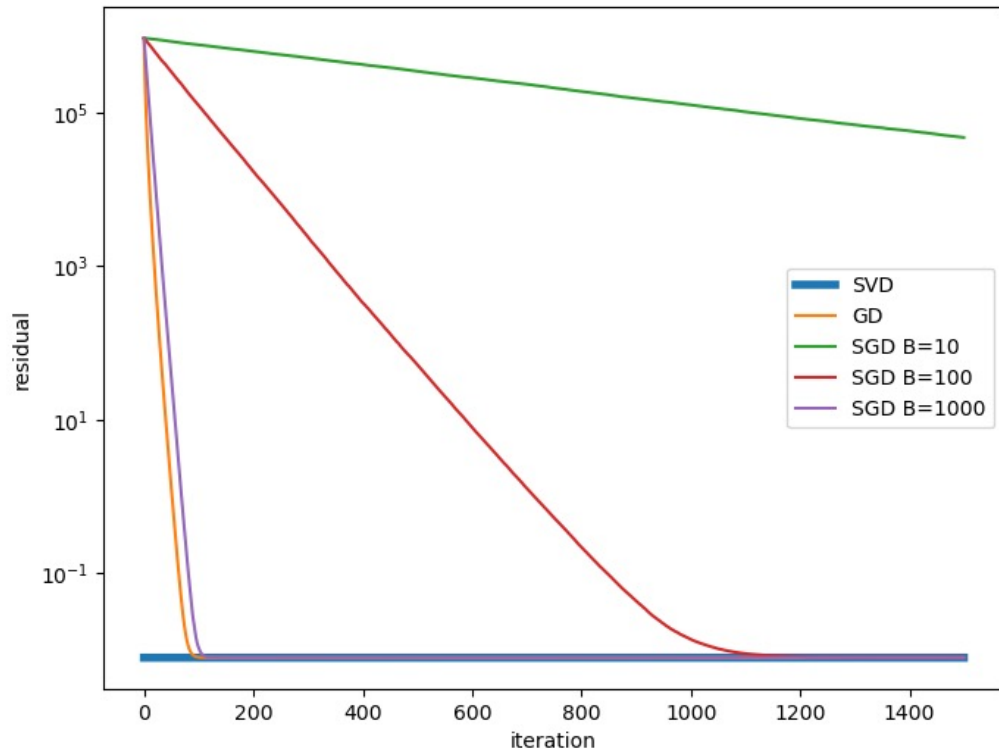
Pass functions as arguments

- Python stuff:
  - Functions as arguments to other functions
  - Multiple return values with `return`  
`a, b`
    - and unpacking with  
`a, b = func()`
  - `time.time()`

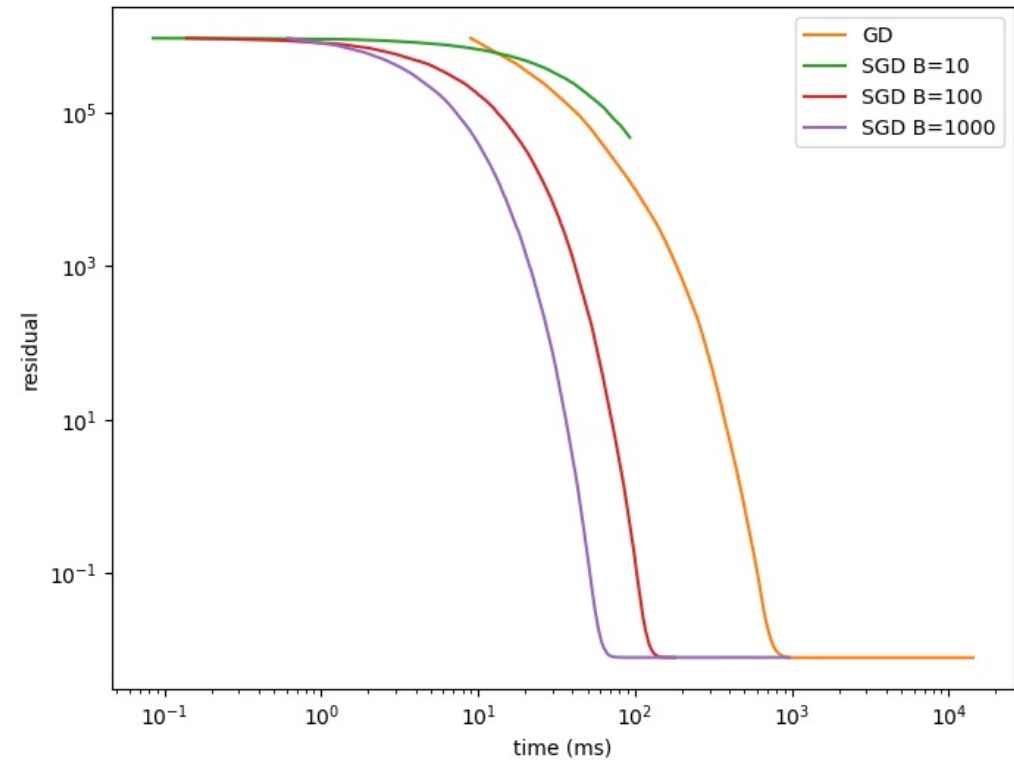
```
def run_gd(A, b, step_size=1e-4, num_iters=1500, grad_fn=grad_l2, residual=residual_l2):  
    ''' Run gradient descent to solve  $Ax = b$   
  
    Parameters  
    -----  
    A : matrix of size (N_measurements, N_dim)  
    b : observations of (N_measurements, 1)  
    step_size : gradient descent step size  
    num_iters : number of iterations of gradient descent  
    grad_fn : function to compute the gradient  
    residual : function to compute the residual  
  
    Returns  
    -----  
    x  
        output matrix of size (N_dim)  
    residual  
        list of calculated residuals at each iteration  
    timing  
        time to execute each iteration (should be cumulative to each iteration)  
    ...  
'''
```

Multiple return values

# Task 3: Gradient Descent



Use full A matrix, not subsampled A, to compute residual.



Note: Exact runtimes and order of convergence in wall clock time may vary!

**Have a nice weekend!**

And good luck with the homework!