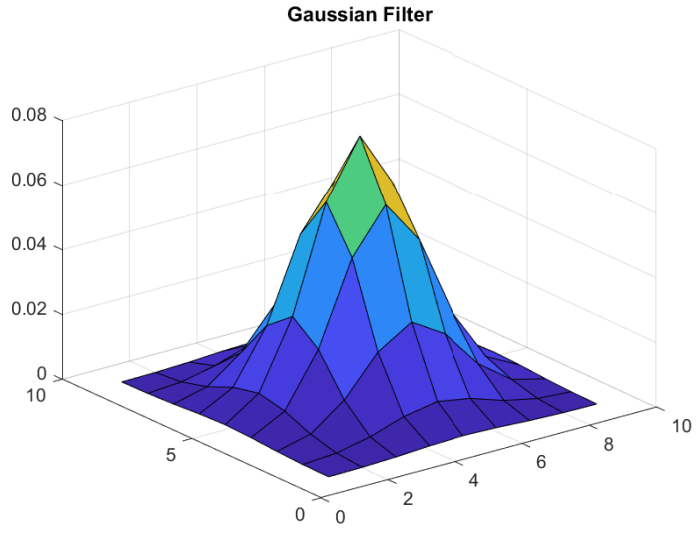
Blurring Filter

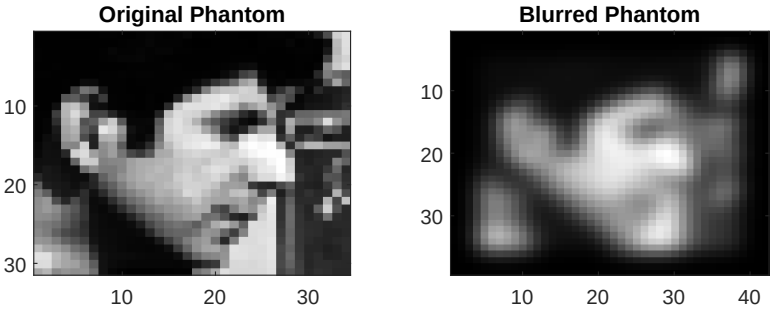
Setup

```
close all;
clc;
```

Part a

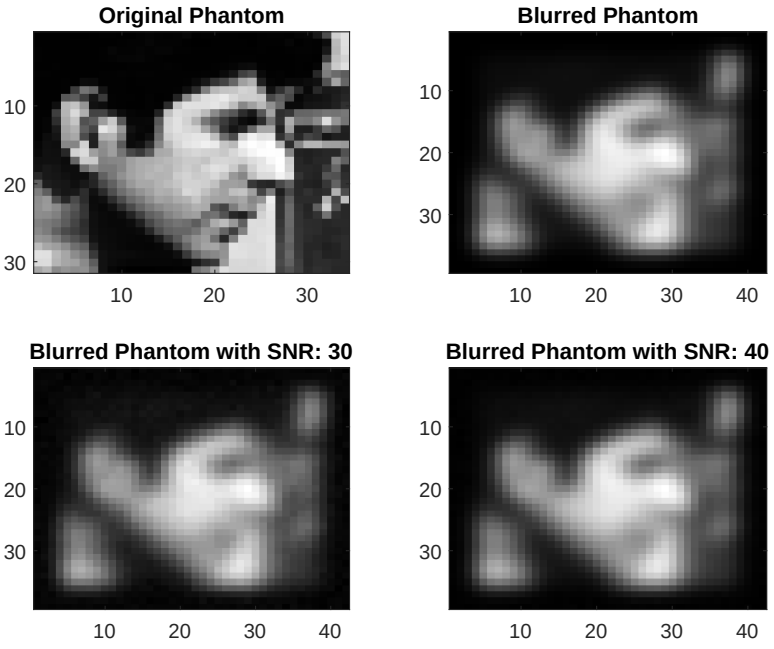
```
I = imread('cameraman.tif');
X = im2double(I);
X = X(50:80,105:138);
F = gauss2d(9,9,0,0,2,2);
figure
surf(F)
title('Gaussian Filter')
C = convmtx2(F,size(X,1),size(X,2));
X_vec = X(:);
y = C*X_vec;
f1 = figure;
ax = gca;
subplot(2,2,1,ax)
imagesc(X), colormap gray
title('Original Phantom')
subplot(2,2,2)
Y_blurred = reshape(y,size(F)+size(X)-1);
imagesc(Y_blurred)
title('Blurred Phantom')
```





Part b

```
for ratio = [3 4]
    variance = var(y)/10^ratio; % SNR adjustment
    sigma_n = sqrt(variance);
    noise = sigma_n * randn(size(y)); % N~(0,sigma_n^2)
    y_n = y + noise;
    y_img = reshape(y_n,size(F)+size(X)-1);
    subplot(2,2,ratio)
    imagesc(y_img)
    title(['Blurred Phantom with SNR: ',num2str(10*ratio)])
end
```

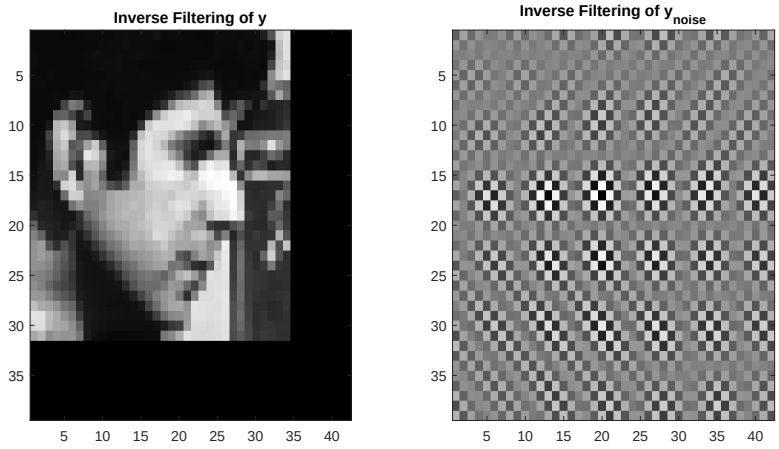


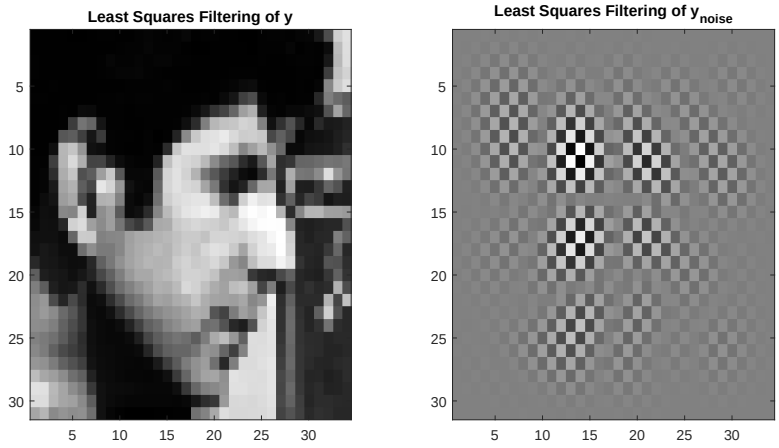
Comments

The effect of the noise is not observable to human eye under the effect of blurring, *i.e.* it is difficult to distinguish the noise from the blurred image.

Part c: Inverse and LS Filtering in Frequency domain

```
figure
size_Y = size(Y_blurred);
X_inv = ifft2(fft2(Y_blurred,size_Y(1), ...
    size_Y(2))./fft2(F,size_Y(1),size_Y(2)),size_Y(1),size_Y(2));
X_inv_noisy = ifft2(fft2(y_img)./fft2(F, ...
    size_Y(1),size_Y(2)),size_Y(1),size_Y(2));
subplot(1,2,1)
imagesc(X_inv), colormap gray
title('Inverse Filtering of y')
subplot(1,2,2)
imagesc(X_inv_noisy)
title('Inverse Filtering of y_{noise}')

figure
C = full(C); % The LS solution outputs wrong results if C remains sparse!
X_inv = reshape(inv(C'*C)*C'*y,size(X));
X_inv_noisy = reshape(inv(C'*C)*C'*y_n,size(X));
subplot(1,2,1)
imagesc(X_inv), colormap gray
title('Least Squares Filtering of y')
subplot(1,2,2)
imagesc(X_inv_noisy)
title('Least Squares Filtering of y_{noise}')
```





Comments

$$\tilde{y} = y + n \tag{1}$$
$$A = U\Sigma V^H \tag{2}$$
$$A^T = V\Sigma U^H \tag{3}$$
$$\hat{x_{LS}} = (A^T A)^{-1} A^T \tag{4}$$

Inserting Eqn. 2 and 3 in 4, results in following expression:

$$\hat{x_{LS}} = \underbrace{V\Sigma^{-1}U^H}_{A^\dagger}(y+n) \tag{5}$$

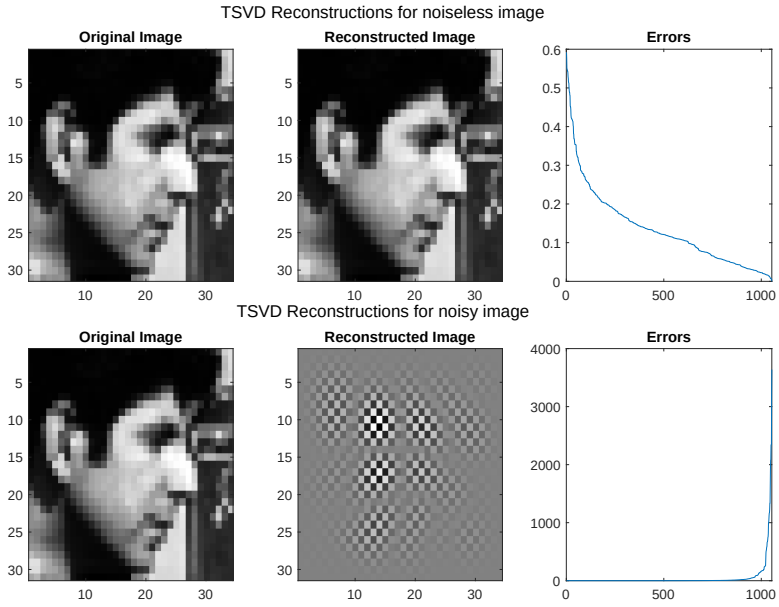
The least squares solution amplifies the noise with pseudo-inverse of A (in SVD sense) hence results in the checkerboard pattern in the reconstruction as an implication of amplified high frequency noise. Since A is an ill-conditioned matrix and hence has really small singular values, noise is amplified with $\frac{1}{\sigma}$ which is a huge amplification factor.

Part d

y = U S VH x; x = V 1/S UH y;

```
[U,S,V] = svd(C);
singular_values = diag(S);
X_back_vec = zeros(size(X_vec));
Errors = zeros(size(singular_values));
for i = 1:length(singular_values)
    s = singular_values(i);
    v = V(:,i);
    u = U(:,i);
```

```
    X_back_vec = X_back_vec + 1/s * v * u' * y;
    Errors(i) = relative_error(X_vec, X_back_vec);
end
X_back = reshape(X_back_vec,size(X));
f = figure;
f.Position = [100 100 900 300];
colormap gray
subplot(1,3,1)
imagesc(X)
title('Original Image')
subplot(1,3,2)
imagesc(X_back)
title('Reconstructed Image')
subplot(1,3,3)
plot(Errors)
title('Errors')
sgtitle('TSVD Reconstructions for noiseless image')

[U,S,V] = svd(C);
singular_values = diag(S);
X_back_vec = zeros(size(X_vec));
Errors = zeros(size(singular_values));
for i = 1:length(singular_values)
    s = singular_values(i);
    v = V(:,i);
    u = U(:,i);
    X_back_vec = X_back_vec + 1/s * v * u' * y_n;
    Errors(i) = relative_error(X_vec, X_back_vec);
end
Full_Errors = Errors;
X_back = reshape(X_back_vec,size(X));
f = figure;
f.Position = [100 100 900 300];
colormap gray
subplot(1,3,1)
imagesc(X)
title('Original Image')
subplot(1,3,2)
imagesc(X_back)
title('Reconstructed Image')
subplot(1,3,3)
plot(Errors)
title('Errors')
sgtitle('TSVD Reconstructions for noisy image')
```



Comments

For this question a distance error metric is defined:

$$RelativeError = \frac{||\hat{x} - x||}{||x||} \tag{6}$$

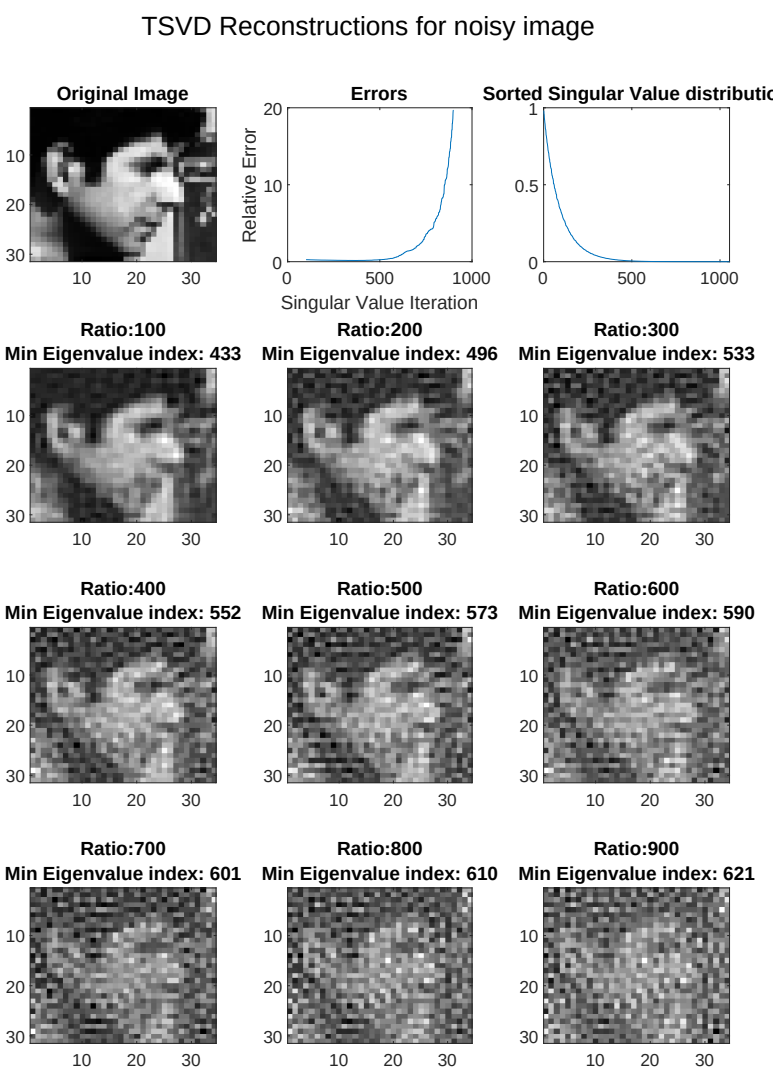
In the figure above, we observe that error is growing after the 900th singular value. Hence, to find a suitable threshold, this interval is scanned.

Find threshold for noisy reconstruction

A ratio of $\frac{1}{200}$ in singular value amplitude provided a satisfactory amount of reconstruction.

```
f = figure; colormap gray;
f.Position = [100 100 600 800];
plot_idx = 4;
for ratio = 100:100:900
    max_sing_value = max(singular_values);
    threshold = max_sing_value / ratio;
    [minimum, index] = min(abs(singular_values-threshold));

    X_back_vec = zeros(size(X_vec));
    Errors = zeros(index,1);
    for i = 1:index
        s = singular_values(i);
```

```
        v = V(:,i);
        u = U(:,i);
        X_back_vec = X_back_vec + 1/s * v * u' * y_n;
        Errors(i) = relative_error(X_vec, X_back_vec);
    end
    X_back = reshape(X_back_vec,size(X));
    subplot(4,3,plot_idx)
    imagesc(X_back)
    title({['Ratio:',num2str(ratio)] ...
        ,['Min Eigenvalue index: ',num2str(index)]})
    plot_idx = plot_idx + 1;
end

colormap gray
subplot(4,3,1)
imagesc(X)
title('Original Image')

subplot(4,3,2)
plot(100:900,Full_Errors(100:900))
title('Errors')
xlabel('Singular Value Iteration')
ylabel('Relative Error')

subplot(4,3,3)
plot(singular_values)
title('Sorted Singular Value distribution')
sgtitle('TSVD Reconstructions for noisy image')
```



Comments

Experiments showed that applying a threshold around the 1% of the maximum singular value resulted in moderate success when compared to the other reconstruction thresholds. The corresponding singular value index is found to be 433. This can also be seen in the error plot given in the top of the above figure, where error gets drastically higher after the inclusion of 500 singular values.

Conjugate Gradient

a

Assuming the vector-vector product has computational cost of N operations, the total cost of CG for N iterations and direct inversion is calculated as follows: For one iteration of Conjugate Gradient:

- $\alpha_k : 2N^2 + N \approx 2N^2$
- $x_{k+1} : \approx 1$
- $r_{k+1} : N^2$
- $\beta_k : 2N$
- $p_{k+1} : \approx 1$

Therefore, total cost per iteration can be approximated as $3N^2 + 2N$. For N_{iter} iterations, the overall cost is $(3N^2 + 2N)N_{iter}$. On the other hand, for direct inversion, requiring the computation of $(C^TC + \lambda D^TD)^{-1}$. The computation of the forward matrix costs $2N^2$ operations. Inversion of this matrix costs approximately N^3 operations, resulting in an overall cost of $N^3 + 2N^2$. Assuming $N >> 3$, then CG requires $3N^2N_{iter}$ and direct inversion requires N^3 operations. Hence, for small N_{iter}, CG requires less computation.

b

Comments

As stated, the main intensive task in the Conjugate gradient method is the repetitive forward operators. Tikhonov regularized solution can be found by assigning $A = (C^TC + \lambda D^TD)$. However, this may not be useful in memory-limited systems. Hence forward projections of A should be replaced with another operation requiring less memory. This can be done as follows:

$$Az = (C^TC + \lambda D^TD)z \tag{7}$$
$$= C^T(Cz) + \lambda D^T(Dz) \tag{8}$$

This type of formulation allows us to replace Az with forward operators Cz, C^Tz, Dz and D^Tz for some arbitrary vector z. For some particular C and D, the quantities Cz and C^Tz in Eqn. 8 can be computed using fast-forward routines, without forming the huge matrices that operate on flattened vector representations of the images. Keeping in mind that the operator C represents convolution operation, hence it can be computed via FFTs. C^T is the adjoint operator of convolution and corresponds to the convolution of input with flipped or inverted filter $H(-x, -y)$. In the case of matrix C representing Radon transform, the quantities Cz and C^Tz can be computed by utilizing the fast-forward routines `radon.m` and `iradon.m` in MATLAB. This result is

already derived in Homework 2. Similarly, discrete derivative operation can be computed by convolving the image with the first order derivative kernels, without the need of calculation the matrices R_1 and R_2. Hence, one only needs to store the discrete derivative operator and using the separability feature of the 2D discrete derivative operator, reduces memory requirements drastically. The derivative operator D is operated on the input vector x using the given stacked decomposition as follows:

$$Dx = \begin{bmatrix} R_1 x & R_2 x \end{bmatrix} \tag{9}$$

$$D^T z = \begin{bmatrix} {R_1}^T {R_2}^T \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = {R_1}^T z_1 + {R_2}^T z_2 \tag{10}$$

$$D^T D x = {R_1}^T R_1 x + {R_2}^T R_2 x \tag{11}$$

where ${R_N}^z$ corresponds to convolution with first order derivative kernel in dimension N and ${R_N}^T z$ is convolution with flipped derivative kernel. Using these fast-forward operations, the following scripts are written to perform Tikhonov regularization via Conjugate Gradient.

Forward Convolution A

```
function res = forward_A(H,lambda,D1,D2,p)
H_flipped = flip(flip(H,1),2);
D1_flipped = flip(D1);
D2_flipped = flip(D2);
Cp = conv2(p,H,'same');
CtranposeCp = conv2(Cp,H_flipped,'same');
R1 = conv2(conv2(p,D1,'same'),D1_flipped,'same');
R2 = conv2(conv2(p,D2,'same'),D2_flipped,'same');
res = CtranposeCp + lambda*(R1+R2);
res = res(:);
end
```

Forward Radon A

```
function res = forward_radon(lambda,D1,D2,p)
D1_flipped = flip(D1);
D2_flipped = flip(D2);
theta_step = 180/size(p,1);
thetas = 0:theta_step:180-theta_step;
Cp = radon(p,thetas);
CtranposeCp = iradon(Cp,thetas);
CtranposeCp = CtranposeCp(2:end-1,2:end-1);
R1 = conv2(conv2(p,D1,'same'),D1_flipped,'same');
R2 = conv2(conv2(p,D2,'same'),D2_flipped,'same');
```

```
res = CtranposeCp + lambda*(R1+R2);
res = res(:);
end
```

CG Convolution Tikhonov

```
function estimate = cgconvtik(H, D1, D2, Y, x_0, lambda, n_iter)
size_Y = size(Y);
Y = Y(:);
Ax_0 = forward_A(H,lambda,D1,D2,x_0);
x_0 = x_0(:);
r = Y - Ax_0;
p = r;
for k = 1:n_iter
    Ap = forward_A(H,lambda,D1,D2,reshape(p,size_Y));
    alpha = (r'*r)/(p'*Ap);
    x_0 = x_0 + alpha*p;
    r_new = r - alpha*Ap;
    beta = (r_new'*r_new)/(r'*r);
    p = r_new + beta*p;
    r = r_new;
end
estimate = x_0;
end
```

CG Radon Tikhonov

```
function estimate = cgradontik(D1, D2, Y, x_0, lambda, n_iter)
size_Y = size(Y);
Y = Y(:);
Ax_0 = forward_radon(lambda,D1,D2,x_0);
x_0 = x_0(:);
r = Y - Ax_0;
p = r;
for k = 1:n_iter
    Ap = forward_radon(lambda,D1,D2,reshape(p,size_Y));
    alpha = (r'*r)/(p'*Ap);
    x_0 = x_0 + alpha*p;
    r_new = r - alpha*Ap;
    beta = (r_new'*r_new)/(r'*r);
    p = r_new + beta*p;
    r = r_new;
end
estimate = x_0;
end
```

Q4 d

```
close all; clc; clear;
X = mat2gray(double(imread('cameraman.tif')));
H = gauss2d(15,15,0,0,2,2);
D1 = [-1 0 1];
D2 = D1';
H_flipped = flip(flip(H,1),2);
Y = conv2(X,H,'same');
sigma_s = var(Y(:));
sigma_n = sigma_s * 1e-3;
noise = sigma_n * randn(size(Y));
Y = Y + noise;
b = conv2(Y,H_flipped,'same');
n_iter = 250;
x_0 = randn(size(X));
i = 1;
f1 = figure('units','normalized','outerposition',[0.0823 0.1111 0.7063 0.7963]);
exponents = linspace(-5,-1,25);
for lambda = power(10,exponents)
    estimate = cgconvtik(H, D1, D2, b, x_0, lambda, n_iter);
    X_back = reshape(estimate,size(X));
    subplot(5,5,i)
    imagesc(X_back), colormap gray
    title({['\lambda = ',num2str(lambda)],['Exponent of 10: ',num2str(exponents(i))]})
    i = i+1;
end
sgtitle(['CG Reconstructions for N_{iter} = ',num2str(n_iter)])


X = phantom('Modified Shepp-Logan',256);
D1 = [-1 0 1];
D2 = D1';

theta_step = 180/(size(X,1));
thetas = 0:179;
Y = radon(X);
sigma_s = var(Y(:));
sigma_n = sigma_s * 1e-3;
noise = sigma_n * randn(size(Y));
Y = Y + noise;
b = iradon(Y,thetas);
b = b(2:end-1,2:end-1);
```

```
n_iter = 250;
x_0 = randn(size(X));
i = 1;
f2 = figure('units','normalized','outerposition',[0.1964 0.2250 0.4896 0.6019]);
exponents = linspace(-1,2,20);
for lambda = power(10,exponents)
    tic
    estimate = cgradontik(D1, D2, b, x_0, lambda, n_iter);
    X_back = reshape(estimate,size(X));
    subplot(4,5,i)
    imagesc(X_back), colormap gray
    title({['\lambda = ',num2str(lambda)],['Exponent of 10: ',num2str(exponents(i))]})
    i = i+1;
    toc
end
sgtitle(['CG Reconstructions for N_{iter} = ',num2str(n_iter)])
```

Comments

$$||y - Cx||_2^2 + \lambda||Dx||_2^2 \tag{12}$$

The Tikhonov Regularization uses the cost function given in Eqn. 12. The regularization parameter λ serves as a weighting parameter between prior and data-model-mismatch (effect of noise) that penalizes the quantity expressed by Dx. In our scenarios, this regularization enforces the reconstruction to be smooth. In figures Figures 1 and 2, the trade-off effect is observable when λ is swept through the given intervals. We see the effect of noise for small λ values where noise is amplified and a smoothness (low-pass effect) introduced by highly weighted prior in the larger λ cases.

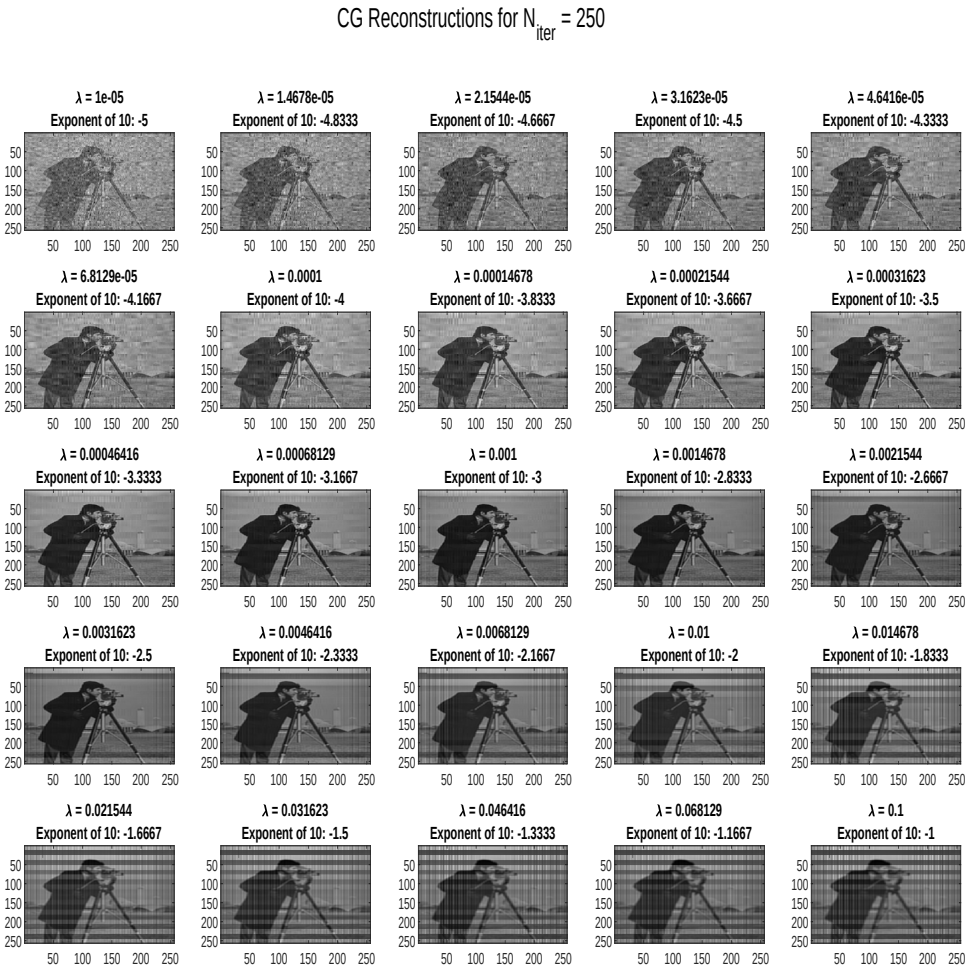


Figure 1: Cameraman Reconstructions

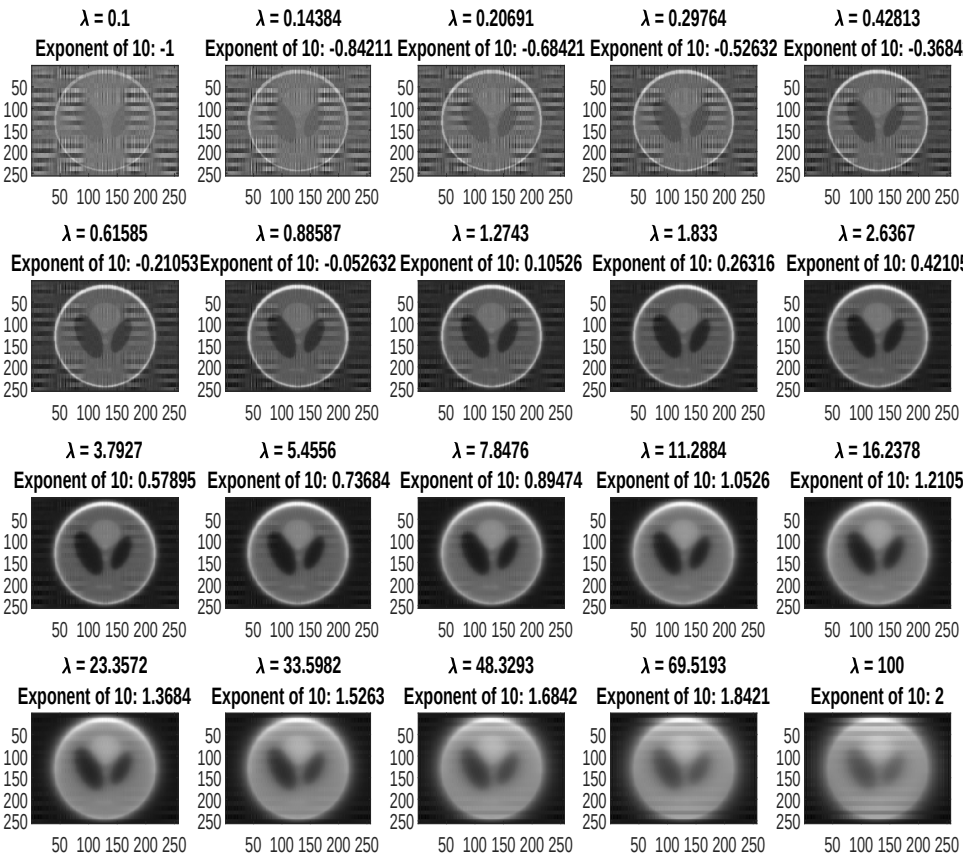


Figure 2: Shepp-Logan Reconstructions