

---

# Assignment 5

CPSC 302 - 2022W1

Devam Sisodraker  
69899771

---

## Problem 1

### Problem 1

Consider the  $2 \times 2$  matrix

$$A = \begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix},$$

and suppose we are required to solve  $A\vec{x} = \vec{b}$ , where  $\vec{b}$  is an arbitrary right-hand side vector. Clearly, solving a  $2 \times 2$  linear system using a stationary scheme is an utterly ridiculous idea, but the following is helpful in understanding more about convergence of stationary methods.

- Find the spectral radius of the Jacobi and Gauss-Seidel iteration matrices and the asymptotic rate of convergence for these two schemes, namely  $-\log_{10} \rho(T)$ , where  $\rho(T)$  denotes the spectral radius of the corresponding iteration matrix,  $T$ .
- How much faster does Gauss-Seidel converge compared to Jacobi for a fixed reduction in the relative residual norm,  $\frac{\|\vec{r}_k\|_2}{\|\vec{b}\|_2}$ , in terms of iteration counts?
- Write down the SOR iteration matrix as a function of the relaxation parameter,  $\omega$ .
- Find the optimal SOR parameter,  $\omega_{\text{opt}}$ , and the spectral radius of the corresponding iteration matrix.
- Approximately how much faster does SOR with  $\omega_{\text{opt}}$  converge compared to Jacobi?

### Solution (a).

Given  $A$  let us first calculate the Jacobi iteration matrix of  $A$ , denoted  $D$ . Since  $D$  all zeros along the non-diagonal entries and shares the same diagonal entries as  $A$  then we conclude that  $D$  is the following. We also indicate its inverse below.

$$D = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} \qquad D^{-1} = \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{pmatrix}$$

Since we have a computed value for  $D^{-1}$  we can now calculate the iteration matrix  $T_{\text{Jacobi}}$ .

$$\begin{aligned} T_{\text{Jacobi}} &= I - D^{-1}A \\ &= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} - \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{pmatrix} \begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix} \\ &= \begin{pmatrix} 0 & \frac{1}{2} \\ \frac{1}{2} & 0 \end{pmatrix} \end{aligned}$$

*continued on next page...*

Now let us calculate the eigenvalues of  $T_{\text{Jacobi}}$  to obtain the spectral radius.

$$\begin{aligned}\det(T_{\text{Jacobi}} - I\lambda) &= 0 \\ \begin{vmatrix} -\lambda & \frac{1}{2} \\ \frac{1}{2} & -\lambda \end{vmatrix} &= 0 \\ \lambda^2 - \frac{1}{4} &= 0 \\ \lambda &= \pm \frac{1}{2}\end{aligned}$$

We know that the largest absolute eigenvalue is  $\frac{1}{2}$ .

$$\rho(D^{-1}) = \lambda_1 = \frac{1}{2}$$

Now that we have a value for  $\rho(D^{-1})$ , we can calculate the asymptotic rate of convergence for the Jacobi iteration matrix of  $A$  to be the following. *Note: The following results were calculated using Matlab.*

$$-\log_{10} \rho(D^{-1}) = -\log_{10} \left( \frac{1}{2} \right) \approx \mathbf{0.3010}$$

We have now calculated the asymptotic rate of convergence for the Jacobi iteration matrix of  $A$ . Now let us calculate the asymptotic rate of convergence and the spectral radius for the Gauss-Seidel iteration matrix of  $A$ .

$$E = \begin{pmatrix} 2 & 0 \\ -1 & 2 \end{pmatrix} \quad E^{-1} = \frac{1}{2 \cdot 2 - 0 \cdot (-1)} \begin{pmatrix} 2 & 0 \\ 1 & 2 \end{pmatrix} = \begin{pmatrix} 1/2 & 0 \\ 1/4 & 1/2 \end{pmatrix}$$

Since we now have a value for  $E^{-1}$  we can now calculate the iteration matrix.

$$\begin{aligned}T_{\text{GS}} &= I - E^{-1}A \\ &= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} - \begin{pmatrix} 1/2 & 0 \\ 1/4 & 1/2 \end{pmatrix} \begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} - \begin{pmatrix} 1 & -1/2 \\ 0 & 3/4 \end{pmatrix} \\ &= \begin{pmatrix} 0 & 1/2 \\ 0 & 1/4 \end{pmatrix}\end{aligned}$$

*continued on next page...*

Now that we have a value for  $T_{\text{GS}}$  let us calculate its eigenvalues.

$$\begin{aligned}\det(T_{\text{GS}} - I\lambda) &= 0 \\ \begin{vmatrix} -\lambda & \frac{1}{2} \\ 0 & 1/4 - \lambda \end{vmatrix} &= 0 \\ -\lambda \cdot \left(\frac{1}{4} - \lambda\right) - \frac{1}{2} \cdot 0 &= 0 \\ \lambda &= 0, \frac{1}{4}\end{aligned}$$

We know that the largest absolute eigenvalue is  $\frac{1}{4}$ .

$$\rho(E^{-1}) = \lambda_1 = \frac{1}{4}$$

Now let us calculate the asymptotic rate of convergence.

$$-\log_{10} \rho(E^{-1}) = -\log_{10} \left(\frac{1}{4}\right) \approx \mathbf{0.6021}$$

**Solution (b).**

Jacobi will take twice as many iterations as Gauss-Seidel since the asymptotic rate of Jacobi is roughly double that of Gauss-Seidel, thus we can say that Gauss-Seidel will converge at twice the rate that Jacobi will converge.

**Solution (c).**

We know that the following given an SOR iteration.

$$M = \frac{1-\omega}{\omega}D + E$$

Let us substitute  $M$  into the function  $T_{\text{SOR}}$  where  $T_{\text{SOR}} : \mathbb{R}^{n \times n} \rightarrow \mathbb{R}^{n \times n}$ .

$$\begin{aligned} T_{\text{SOR}}(\omega) &= I - M^{-1}A \\ &= I - \left( \frac{1-\omega}{\omega}D + E \right)^{-1} A \end{aligned}$$

Thus we have the following function.

$$T_{\text{SOR}} = I - \left( \frac{1-\omega}{\omega}D + E \right)^{-1} A$$

**Solution (d).**

From the textbook (*page 235*) we know the following.

$$\rho = \omega_{opt} - 1$$

Since we already have a value for the thing

**Solution (e).**

---

compare  
both of the  
values from  
d and a then  
compare



## Problem 2

### Problem 2

- (a) Download from the assignment webpage the file `J_GS.m`. In your MATLAB command line, run the command: `J_GS(100)`;  
Include in your assignment solution the convergence graph that you are seeing. This is a linear system with the two-dimensional Laplacian of size  $10,000 \times 10,000$  (we have  $n = 100, n^2 = 10,000$ ), and we are plotting the norm of the relative residual after 10,000 iterations for Jacobi and Gauss-Seidel. There is no reason to be impressed with this graph; convergence here is slow. But we are going to see the effect of using SOR.
- (b) Given the eigenvalues of the Laplacian in the slides, show that the optimal SOR parameter can be expressed as

$$\omega_{\text{opt}} = \frac{2}{1 + \sin\left(\frac{\pi}{n+1}\right)}.$$

- (c) Modify the MATLAB function so that in addition to the Jacobi and Gauss-Seidel graphs (for the same matrix) a convergence plot for the SOR method is included as well. Use the same initial guess (the zero vector) and the same stopping criterion:  $\frac{\|\vec{r}_k\|_2}{\|\vec{b}\|_2} < 10^{-6}$ .
- Tip: You should be seeing an *extremely dramatic* improvement in convergence. If you are not seeing such an improvement, then you must have done something wrong.
- (d) Explain your results.

## Solution (a).

Note: The line

```
saveas(gcf, "DevamSisodraker_2a.jpg", "jpg");
```

was appended to the end of **File: J\_GS.m** to ease the exporting process.

```
File: J_GS.m

function J_GS(n)
close all
gcf
% Apply a stationary method to a linear system involving the n^2-by-n^2
% Laplacian

A=delsq(numgrid('S',n+2));
D=diag(diag(A)); % M=D for Jacobi
E=tril(A); % M=E for Gauss-Seidel
itermax=10000; % maximum number of iterations
resvecJ=zeros(itermax,1); % allocate initial space for Jacobi residual norm vector
resvecGS=zeros(itermax,1); % allocate initial space for Gauss-Seidel residual vector

% Jacobi
xJ=zeros(n^2,1);
b=A*ones(n^2,1); % generate a solution of all 1s and a right-hand side
nb=norm(b);
r=b;
for i=1:10000
    resvecJ(i)=norm(r)/nb; % relative residual norm
    if resvecJ(i)<1e-6, break,end % terminate loop if stopping criterion is satisfied
    xJ=xJ+D\r; % next iterate
    r=b-A*xJ; % update residual
end
semilogy(resvecJ); % plot relative residual norm for Jacobi

% Gauss-Seidel
xGS=zeros(n^2,1);
r=b;
for i=1:10000
    resvecGS(i)=norm(r)/nb; % relative residual norm
    if resvecGS(i)<1e-6, break,end % terminate loop if stopping criterion is satisfied
    xGS=xGS+E\r; % next iterate
    r=b-A*xGS; % update residual
end
hold on
semilogy(resvecGS,'r'); % plot relative residual norm for Gauss-Seidel
legend('Jacobi','Gauss-Seidel')
saveas(gcf, "DevamSisodraker_2a.jpg", "jpg");
```

continued on next page...

```
File: DevamSisodraker_2a.out.txt

>> J_GS(100)

ans =

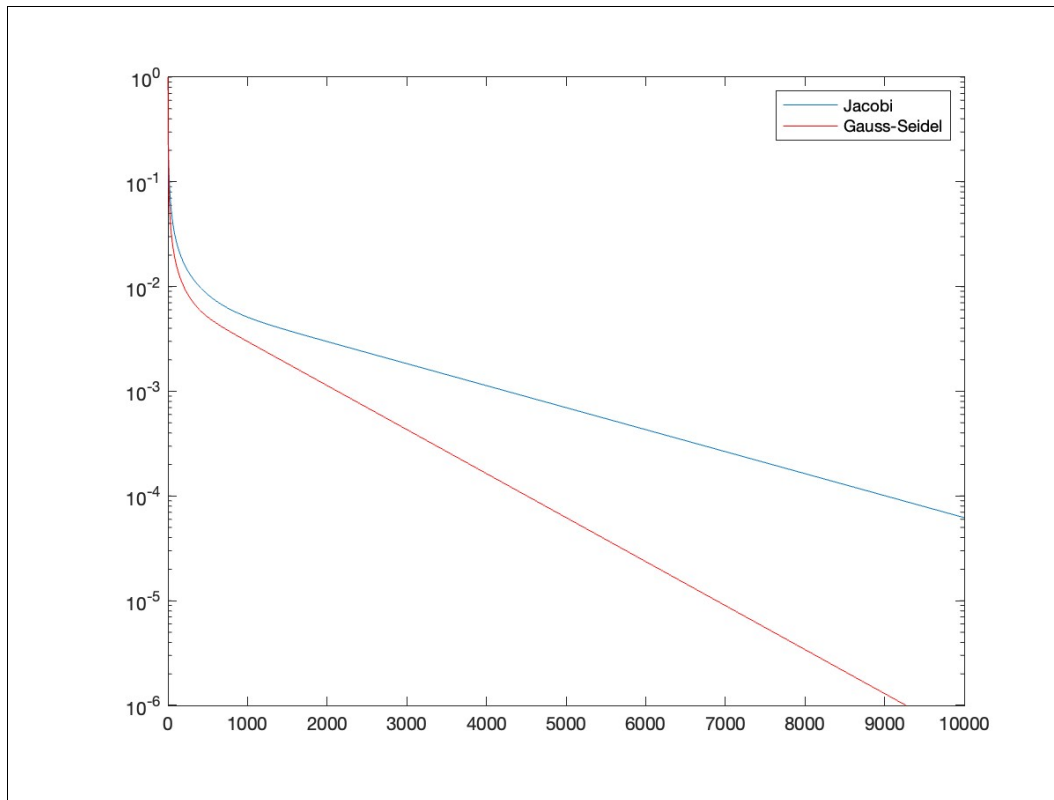
Figure (1) with properties:

    Number: 1
    Name: ''
    Color: [0.9400 0.9400 0.9400]
    Position: [584 595 560 420]
    Units: 'pixels'

Show all properties

>>
```

*continued on next page...*



**Solution (b).**

find proof in  
book

## Solution (c).

```

File: DevamSisodraker_2c.m

function J_GS(n)
close all
gcf
% Apply a stationary method to a linear system involving the n^2-by-n^2
% Laplacian

A=delsq(numgrid('S',n+2));
D=diag(diag(A)); % M=D for Jacobi
E=tril(A); % M=E for Gauss-Seidel
itermax=10000; % maximum number of iterations
resvecJ=zeros(itermax,1); % allocate initial space for Jacobi residual norm vector
resvecGS=zeros(itermax,1); % allocate initial space for Gauss-Seidel residual vector
resvecSOR=zeros(itermax,1); % allocate initial space for SOR residual vector

% Jacobi
xJ=zeros(n^2,1);
b=A*ones(n^2,1); % generate a solution of all 1s and a right-hand side
nb=norm(b);
r=b;
for i=1:10000
    resvecJ(i)=norm(r)/nb; % relative residual norm
    if resvecJ(i)<1e-6, break,end % terminate loop if stopping criterion is satisfied
    xJ=xJ+D\r; % next iterate
    r=b-A*xJ; % update residual
end
semilogy(resvecJ); % plot relative residual norm for Jacobi

% Gauss-Seidel
xGS=zeros(n^2,1);
r=b;
for i=1:10000
    resvecGS(i)=norm(r)/nb; % relative residual norm
    if resvecGS(i)<1e-6, break,end % terminate loop if stopping criterion is satisfied
    xGS=xGS+E\r; % next iterate
    r=b-A*xGS; % update residual
end
hold on
semilogy(resvecGS,'r'); % plot relative residual norm for Gauss-Seidel

% SOR
xSOR=zeros(n^2,1);
r=b;
for i=1:10000
    wOpt = 2 / (1 + sin(pi / (n + 1)));
    resvecSOR(i)=norm(r)/nb; % relative residual norm
    if resvecSOR(i)<1e-6, break,end % terminate loop if stopping criterion is satisfied
    xSOR=xSOR+((1 - wOpt)*D + wOpt*E)\r; % next iterate
    r=b-A*xSOR; % update residual
end
hold on
semilogy(resvecSOR,'g'); % plot relative residual norm for Gauss-Seidel

legend('Jacobi','Gauss-Seidel','SOR')
saveas(gcf, "DevamSisodraker_2c.jpg", "jpg");

```

continued on next page...

```
File: DevamSisodraker_2c.out.txt

>> DevamSisodraker_2c(100)

ans =

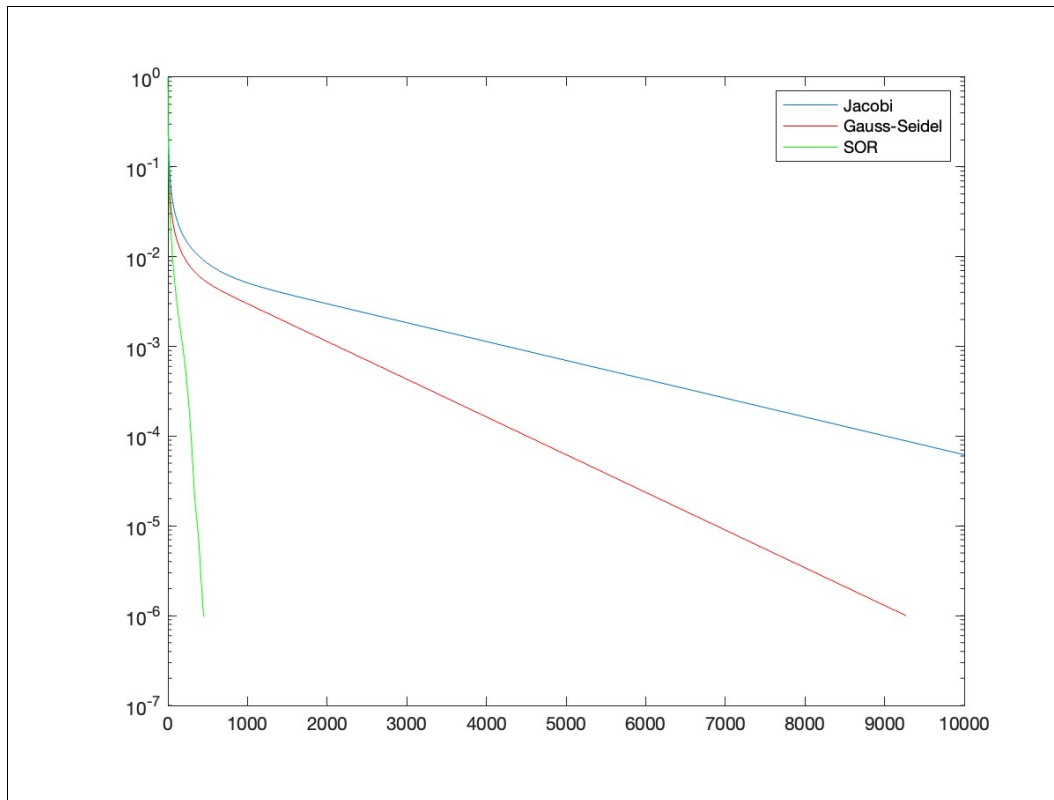
Figure (1) with properties:

    Number: 1
    Name: ''
    Color: [0.9400 0.9400 0.9400]
    Position: [584 595 560 420]
    Units: 'pixels'

Show all properties

>>
```

*continued on next page...*





**Solution (d).**

---

write some  
bs explana-  
tion after  
you see a, b,  
c

### Problem 3

**Problem 3**

- (a) Suppose  $A$  is an  $n$ -by- $n$  orthogonal matrix. Show that all its singular values are equal to 1.
- (b) Recall that an orthogonal projector is a symmetric matrix  $P$  for which  $P^2 = P$ . What are the eigenvalues of an orthogonal projector?

**Solution (a).**

bullshit

**Solution (b).**

bullshit

## Problem 4

### Problem 4

Load the following .mat matrix that appears on the assignment page:

```
load powerMatrix;
```

If you hit `whos` you should be seeing a matrix called  $A$ , of size  $100 \times 100$ . To validate any of your results below, you may run the MATLAB command `eig`, as long as you understand that in typical eigenvalue computations (in a potentially more challenging computational environment) we generally do not have the luxury of running `eig` to check ourselves.

- (a) Apply the power method. Terminate the iteration once the iterates satisfy

$$|\lambda_1^{(k)} - \lambda_1^{(k-1)}| < 10^{-4}.$$

Your program should print out the value of the final iterate and a graph of the absolute errors:  $|\lambda_1^{(k)} - \lambda_{\max}|$ . For better visualization, use `semilogy` for your graphs when necessary. As an initial guess for the eigenvector use a vector produced by the MATLAB command `randn`. (When you repeat your experiments, the number of iterations may slightly vary due to the random initial guess.)

- (b) Repeat your computations with the *inverse* power iteration, with a shift  $\alpha = 4$ , and produce the same graph as you did for the power method.
- (c) Discuss the differences between the performance of the power and the inverse power methods in terms of the cost of single iterations and the overall computational cost.
- (d) Suppose now that we know that  $A$  has an eigenvalue close to 3 and we are interested to compute it to six correct decimal digits. Suggest an efficient procedure for doing so. Implement your suggested algorithm and compute the eigenvalue.

## Solution (a).

```
File: DevamSisodraker_4a.m

gcf
hold on;

clear;
load('powerMatrix.mat')

k = 0;
vector_k = randn(100, 1);
lambda_k = transpose(vector_k)*A*vector_k;
lambda_k_1 = transpose(vector_k)*A*vector_k;
lambda_audit = [];
lambda_delta_audit = [];
eigval = max(eig(A));

while true
    % statements here
    % if ~WhileCondition, break ; end
    lambda_k_1 = lambda_k;
    vector_k = A*vector_k;
    vector_k = vector_k/norm(vector_k);
    lambda_k = transpose(vector_k)*A*vector_k;
    lambda_audit = [lambda_audit, lambda_k];
    lambda_delta_audit = [lambda_delta_audit, abs(lambda_k - eigval)];
    k = k + 1;
    if abs(lambda_k_1 - lambda_k) < 10^-4
        break;
    end
end

plot(1:1:k, lambda_audit);

plot(1:1:k, lambda_delta_audit);

plot([1, k], [eigval, eigval]);

title("4a");
legend({ ...
    '\lambda_k', ...
    '| \lambda_k - \lambda_{MAX} |', ...
    '\lambda_{MAX}', ...
});
xlabel("k");
ylabel("Value");

hold off;
saveas(gcf, "DevamSisodraker_4a.jpg", "jpg");
```

*continued on next page...*

```
File: DevamSisodraker_4a.out.txt

>> DevamSisodraker_q4a

ans =

Figure (1) with properties:

    Number: 1
    Name: ''
    Color: [0.9400 0.9400 0.9400]
    Position: [584 595 560 420]
    Units: 'pixels'

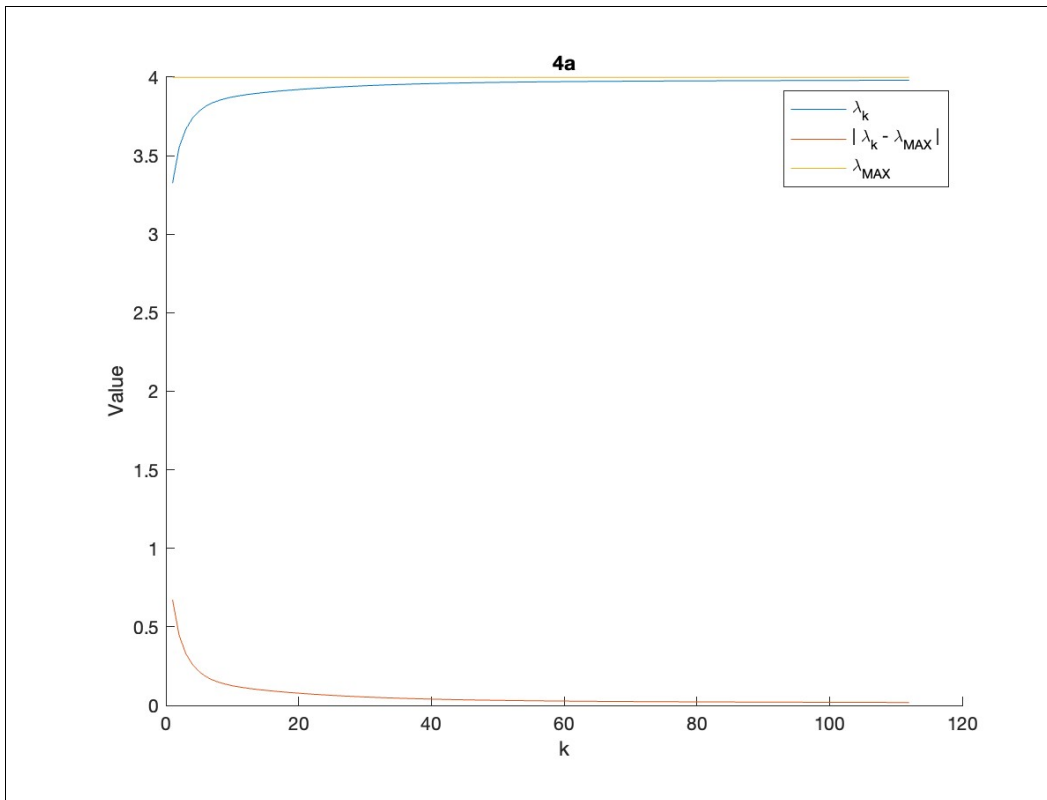
Show all properties

lambda_k =

    3.9804

>>
```

*continued on next page...*



Solution (b).

File: DevamSisodraker\_4b.m

```
gcf
hold on;

clear;
load('powerMatrix.mat');

k = 0;
vector_k = randn(100, 1);
vector_k_1 = vector_k;
lambda_k = transpose(vector_k)*A*vector_k;
lambda_k_1 = transpose(vector_k)*A*vector_k;
lambda_audit = [];
lambda_delta_audit = [];
eigval = max(eig(A));
alpha = 4;

while true
    % statements here
    % if ~WhileCondition, break ; end
    lambda_k_1 = lambda_k;
    vector_k_1 = vector_k;
```

```
vector_k = (A - alpha * eye(size(A)))/transpose(vector_k_1);
vector_k = vector_k/norm(vector_k);
lambda_k = transpose(vector_k)*A*vector_k;

lambda_audit = [lambda_audit, lambda_k];
lambda_delta_audit = [lambda_delta_audit, abs(lambda_k - eigval)];
k = k + 1;

if abs(lambda_k_1 - lambda_k) < 10^-4
    break;
end
end

plot(1:1:k, lambda_audit);
plot(1:1:k, lambda_delta_audit);
plot([1, k], [eigval, eigval]);

title("4b");
legend({ ...
    '\lambda_k', ...
    '| \lambda_k - \lambda_{MAX} |', ...
    '\lambda_{MAX}', ...
});
xlabel("k");
ylabel("Value");

hold off;
saveas(gcf, "DevamSisodraker_4b.jpg", "jpg");
lambda_k
```

*continued on next page...*



```
File: DevamSisodraker_4b.out.txt

>> DevamSisodraker_4b

ans =

Figure (1) with properties:

    Number: 1
    Name: ''
    Color: [0.9400 0.9400 0.9400]
    Position: [584 595 560 420]
    Units: 'pixels'

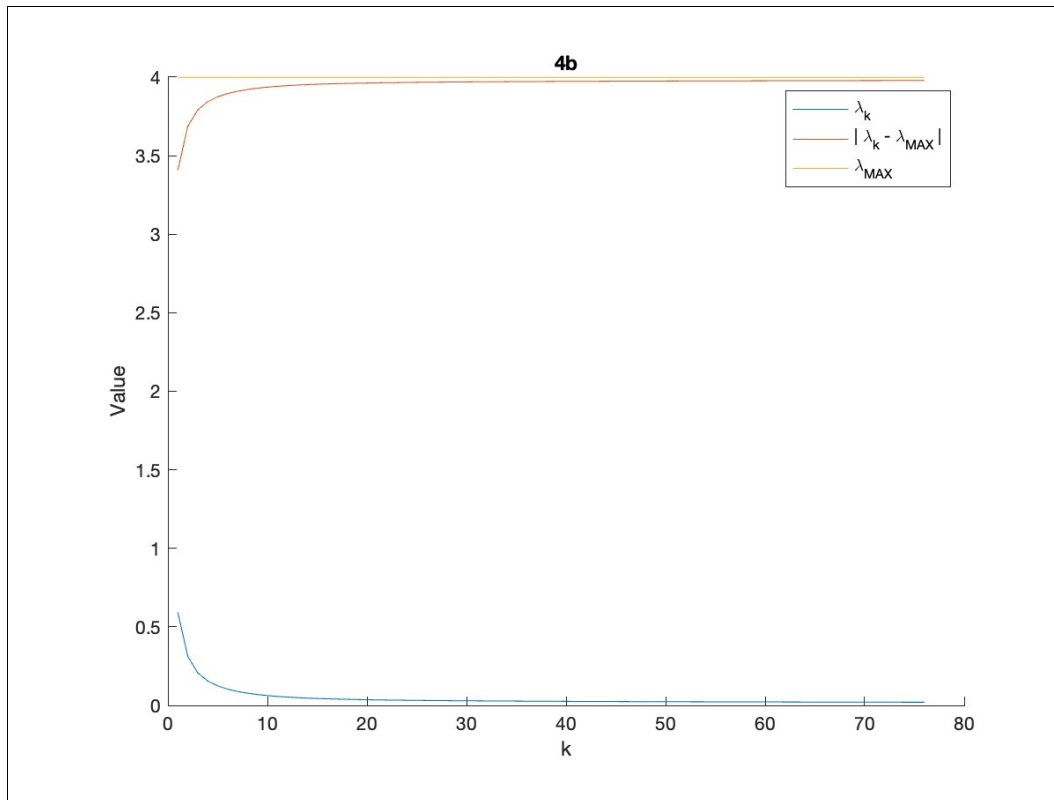
Show all properties

lambda_k =

    0.0202

>>
```

*continued on next page...*



**Solution (c).**

bullshit at a  
later time

**Solution (d).**

We will use the inverse power method with  $\alpha = 3$  to find the eigenvalue close to 3.

*continued on next page...*

```
File: DevamSisodraker_4d.m

gcf
hold on;

clear;
load('powerMatrix.mat');

k = 0;
vector_k = randn(100, 1);
vector_k_1 = vector_k;
lambda_k = transpose(vector_k)*A*vector_k;
lambda_k_1 = transpose(vector_k)*A*vector_k;
lambda_audit = [];
alpha = 3;

while true
    % statements here
    % if ~WhileCondition, break ; end
    lambda_k_1 = lambda_k;
    vector_k_1 = vector_k;

    vector_k = (A - alpha * eye(size(A)))\vector_k_1;
    vector_k = vector_k/norm(vector_k);
    lambda_k = transpose(vector_k)*A*vector_k;

    lambda_audit = [lambda_audit, lambda_k];
    k = k + 1;

    if abs(lambda_k_1 - lambda_k) < 10^-4
        break;
    end
end

plot(1:1:k, lambda_audit);

title("4d");
legend({ ...
    '\lambda_k', ...
});
xlabel("k");
ylabel("Value");

hold off;
saveas(gcf, "DevamSisodraker_4d.jpg", "jpg");
lambda_k
```

*continued on next page...*

```
File: DevamSisodraker_4d.out.txt

>> DevamSisodraker_4d

ans =

Figure (1) with properties:

    Number: 1
    Name: ''
    Color: [0.9400 0.9400 0.9400]
    Position: [584 595 560 420]
    Units: 'pixels'

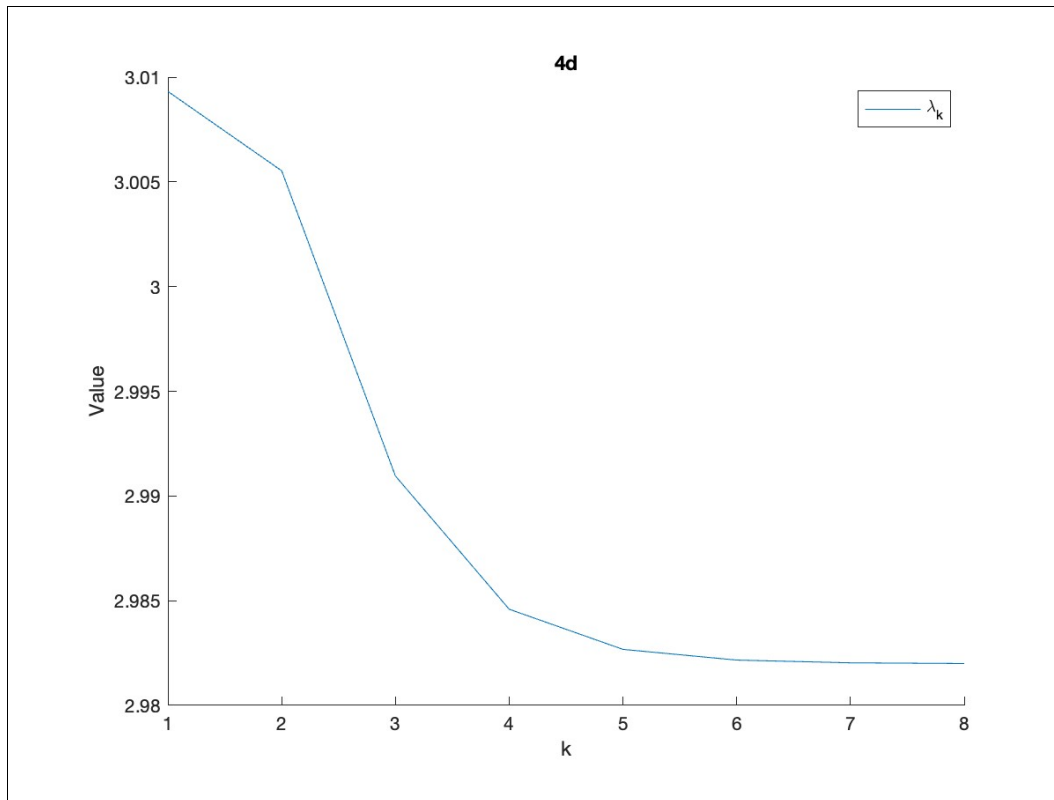
Show all properties

lambda_k =

    2.9820

>>
```

*continued on next page...*

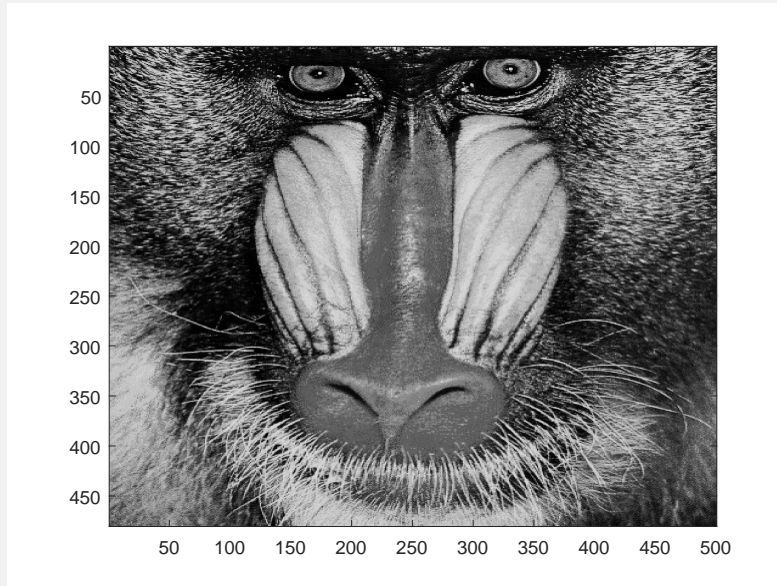


## Problem 5

### Problem 5

The following picture appears in MATLAB's repository of images, and can be retrieved by entering

```
load mandrill;  
colormap('gray');  
image(X);
```



- (a) Print out the images generated by the truncated SVD. Start with  $r = 2$  and go up by powers of 2, to  $r = 2^6 = 64$  (six plots in total). For a compact presentation of your figures, you may use the command `subplot(3,2)`. (Check out `help subplot`.)
- (b) Comment briefly on the quality of the images as a function of  $r$ . For what value of  $r$  would you say that the quality of the image is acceptable, in that we can be rather confident of what we are seeing? (We are not looking for a specific “correct answer” here - just make your subjective observation.)
- (c) For the value of  $r$  you stated in part (b), how much storage is required? Compare it to the storage required for the original image.



**Solution (a).**

File: DevamSisodraker\_5a.m

```
clear;
load mandrill;
colormap("gray");
[U,S,V] = svd(X);
image(U*S*V');

close all;
for i = 1:1:6
    r = 2^i;

    dims = size(X);
    S_trunc = diag(S);
    S_trunc((r + 1):min(dims(1), dims(2))) = 0;
    S_trunc = diag(S_trunc);
    S_trunc(dims(1), dims(2)) = 0;

   (gcf
    hold on;
    Xout = U * S_trunc * V';
    colormap("gray");
    image(flipud(Xout));
    title(strcat("r=", string(r)));
    saveas(gcf, strcat("DevamSisodraker_5a_", string(r), ".jpg"), "jpg");
    hold off;
end
```

*continued on next page...*

```
File: DevamSisodraker_5a.out.txt

>> DevamSisodraker_5a

ans =

Figure (1) with properties:

    Number: 1
    Name: ''
    Color: [0.9400 0.9400 0.9400]
    Position: [584 595 560 420]
    Units: 'pixels'

Show all properties

ans =

Figure (1) with properties:

    Number: 1
    Name: ''
    Color: [0.9400 0.9400 0.9400]
    Position: [584 587 560 420]
    Units: 'pixels'

Show all properties

ans =

Figure (1) with properties:

    Number: 1
    Name: ''
    Color: [0.9400 0.9400 0.9400]
    Position: [584 587 560 420]
    Units: 'pixels'

Show all properties

ans =

Figure (1) with properties:
```

```
    Number: 1
    Name: ''
    Color: [0.9400 0.9400 0.9400]
    Position: [584 587 560 420]
    Units: 'pixels'

Show all properties

ans =

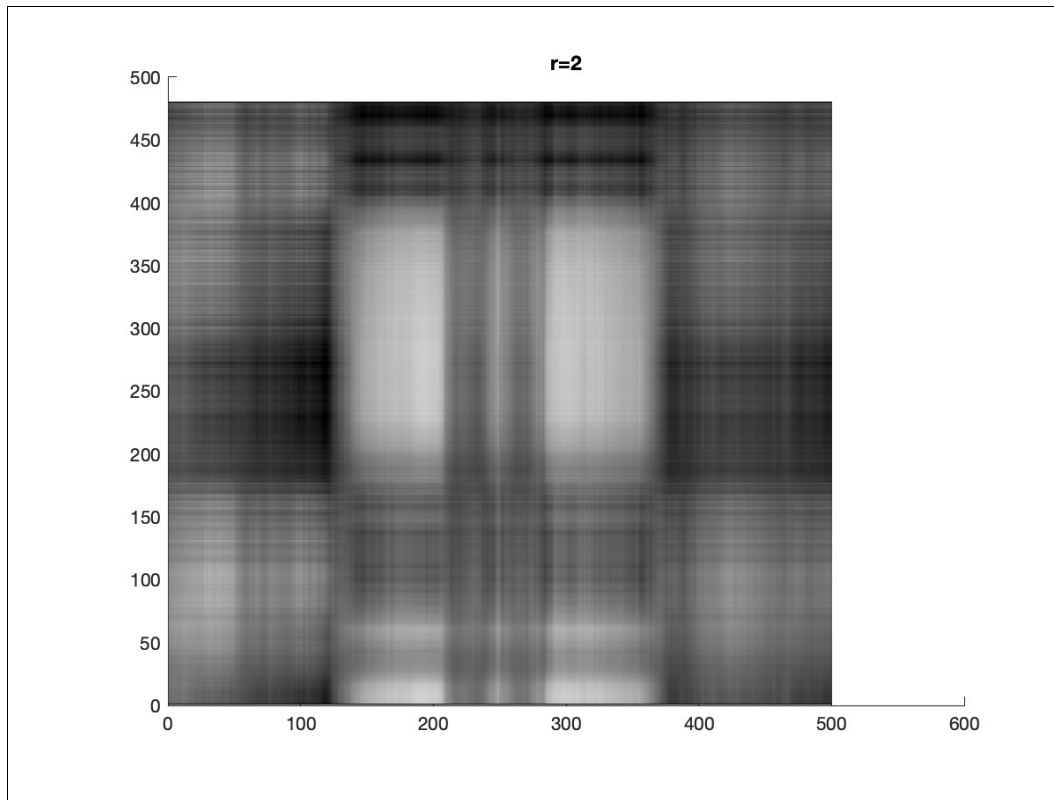
Figure (1) with properties:

    Number: 1
    Name: ''
    Color: [0.9400 0.9400 0.9400]
    Position: [584 587 560 420]
    Units: 'pixels'

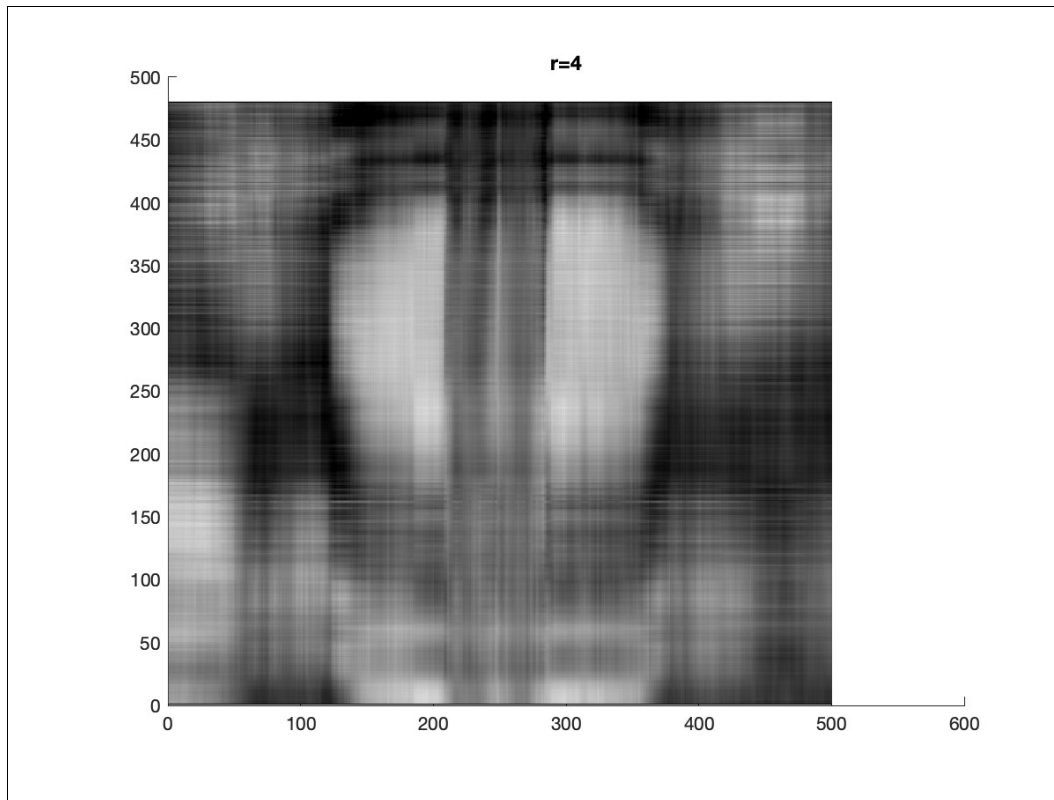
Show all properties

>> >
```

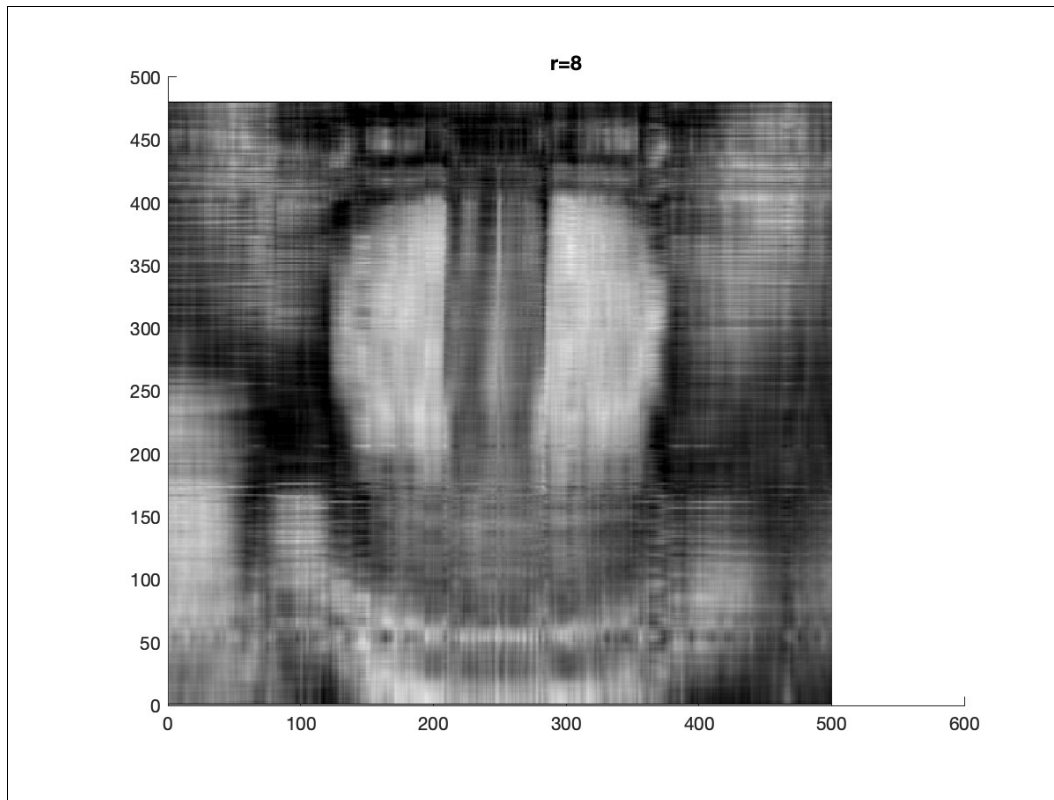
*continued on next page...*



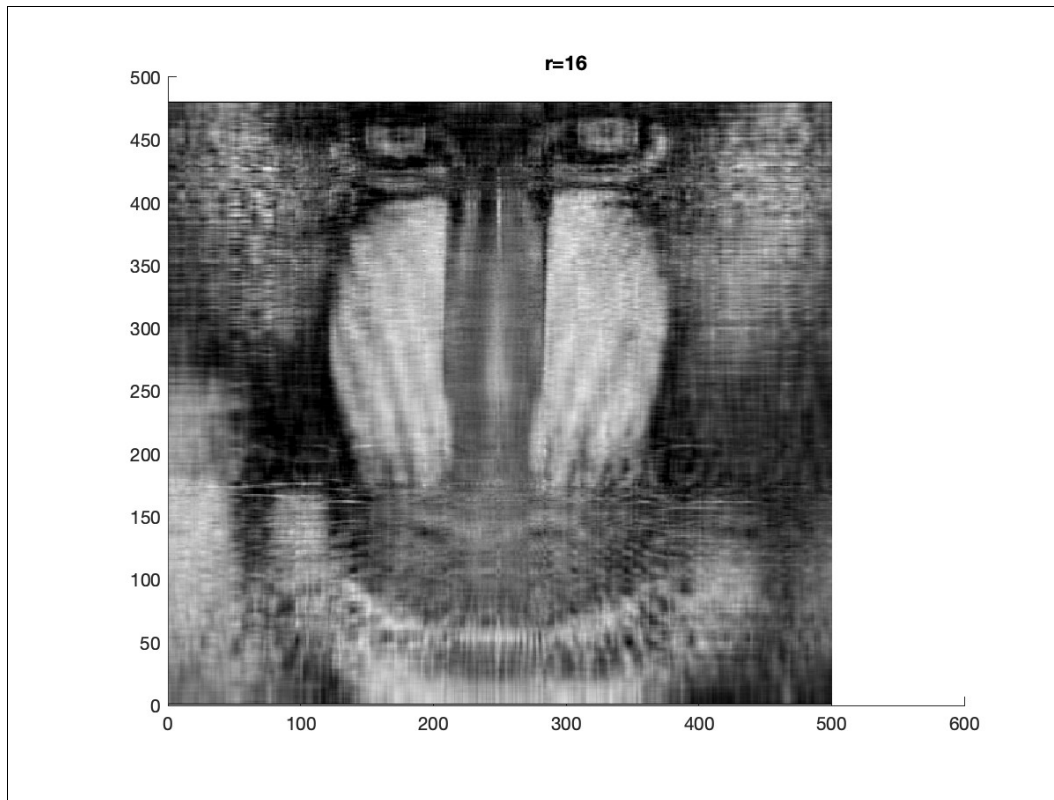
*continued on next page...*



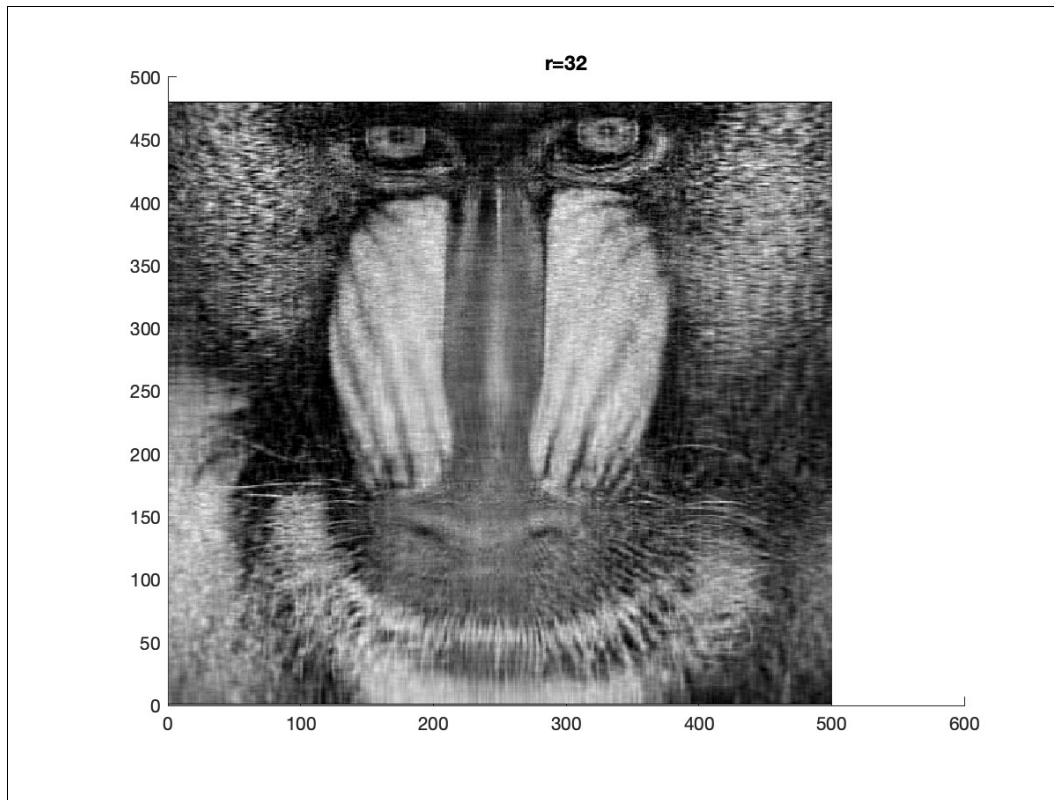
*continued on next page...*



*continued on next page...*

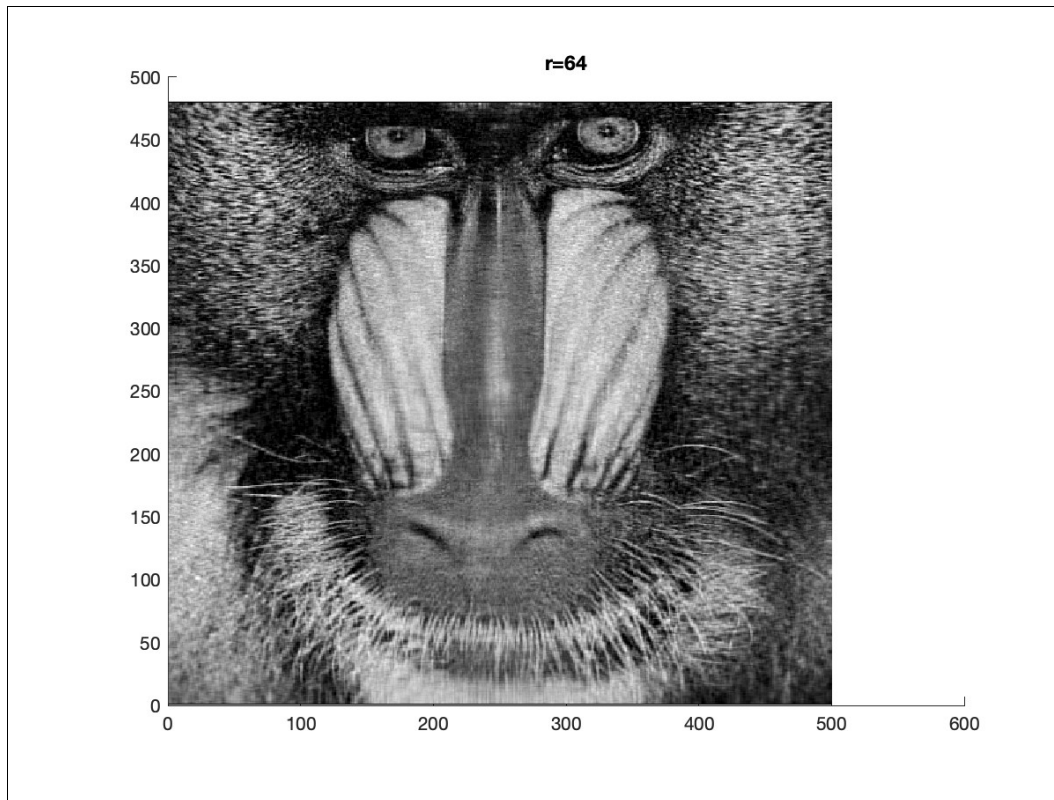


*continued on next page...*



*continued on next page...*





**Solution (b).**

bullshit

**Solution (c).**

bullshit