Assignment 5

CPSC 302 - 2022W1

Devam Sisodraker
69899771

Problem 1

Problem 1
Consider the 2×2 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 2×2 linear system using a stationary scheme is an utterly ridiculous idea, but the following is helpful in understanding more about convergence of stationary methods.

(a) 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.

(b) 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?

(c) Write down the SOR iteration matrix as a function of the relaxation parameter, ω.

(d) Find the optimal SOR parameter, ω_{opt}, and the spectral radius of the corresponding iteration matrix.

(e) Approximately how much faster does SOR with ω_{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_{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_{Jacobi} to obtain the spectral radius.

calculate the eigenvalues of T_{Jacobi} here

todo

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

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

Now that we have a value for $\rho\left(D^{-1}\right)$, 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.*

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

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.

finish this part here, after you finish the Jacobi part

thing

0.6021 for rate of convergence

Solution (b).
Gauss-Seidel will converge at twice the rate that Jacobi will converge. Jacobi will take twice as many iterations as Gauss-Seidel.

Solution (c).

We know that the SOR iteration matrix T_{SOR} is the following by definition.

$$I - \omega((1-\omega)D + \omega E)^{-1}A$$

Thus we can compose it as the following function.

$$W(\omega) = I - \frac{1}{\omega}((1-\omega)D + \omega E)A$$

write this up properly

Solution (d).

use page 235 ch7 from textbook to solve this

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).

insert image a

Solution (b).

find proof in book

Solution (c).

insert image c

fix graph and compare

Solution (d).

write some bs explanation 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 100×100. 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 t 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

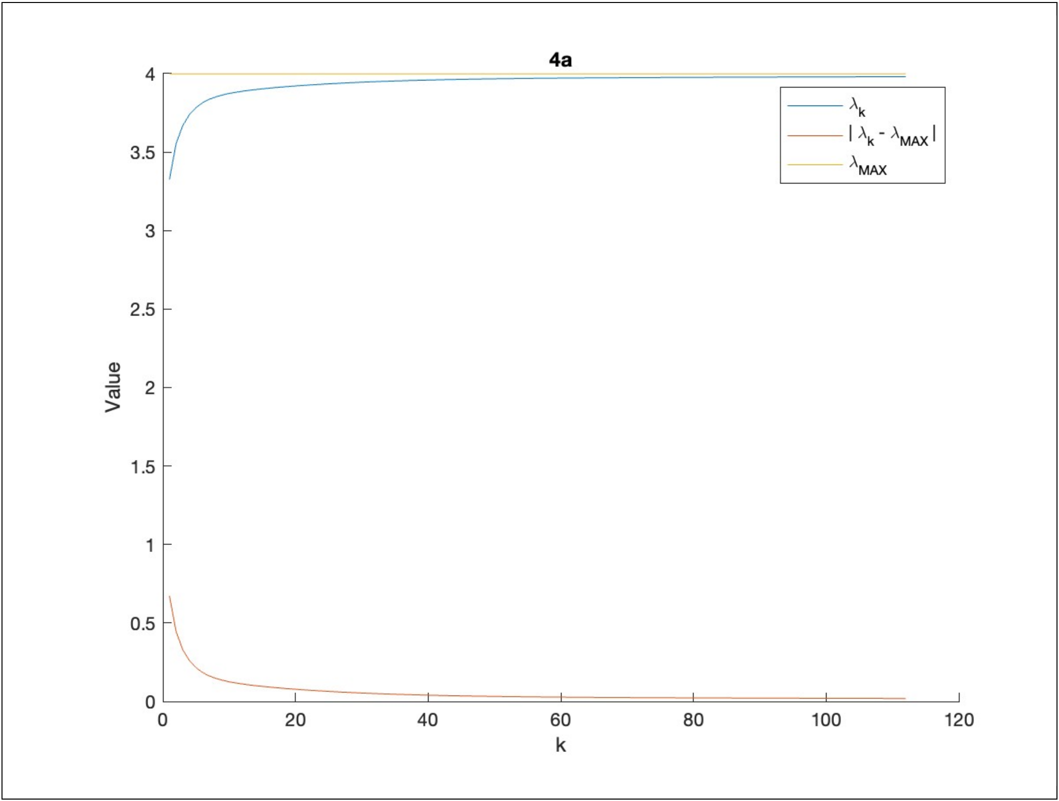
```
gcf
hold on;

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

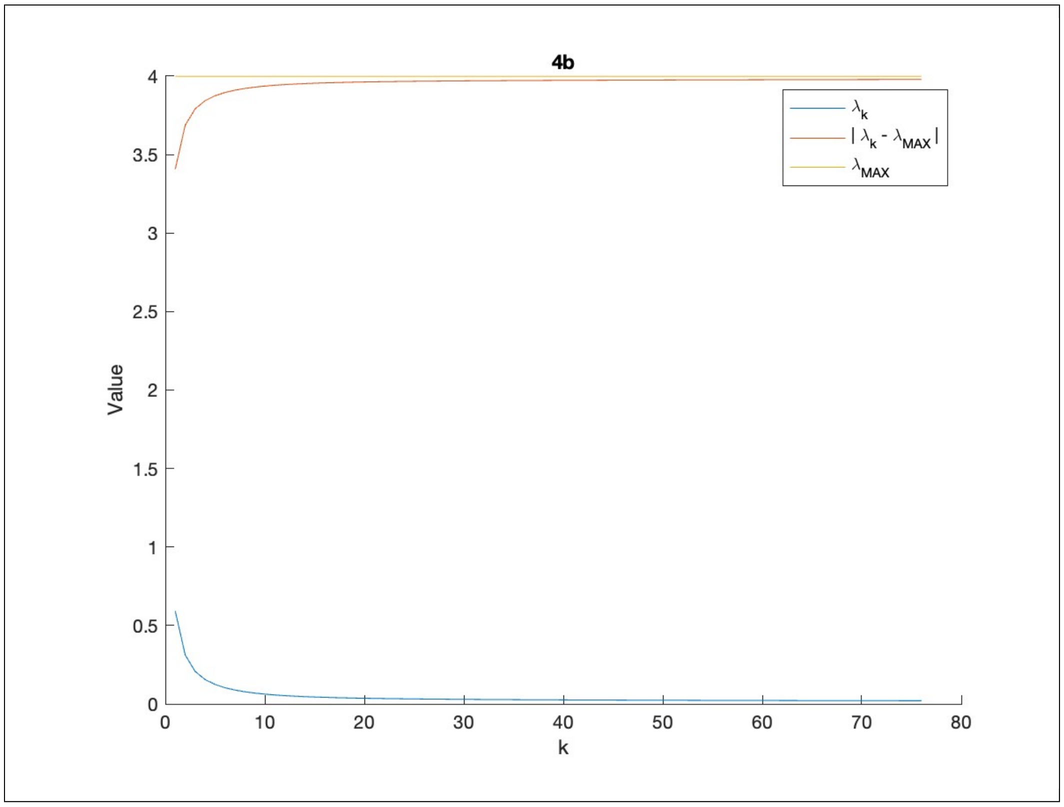
```
gcf
hold on;

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...


4b
Value
k
λ_k
$|\lambda_k - \lambda_{MAX}|$
λ_{MAX}
0
0.5
1
1.5
2
2.5
3
3.5
4
0
10
20
30
40
50
60
70
80

Solution (c).

bullshit at a later

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

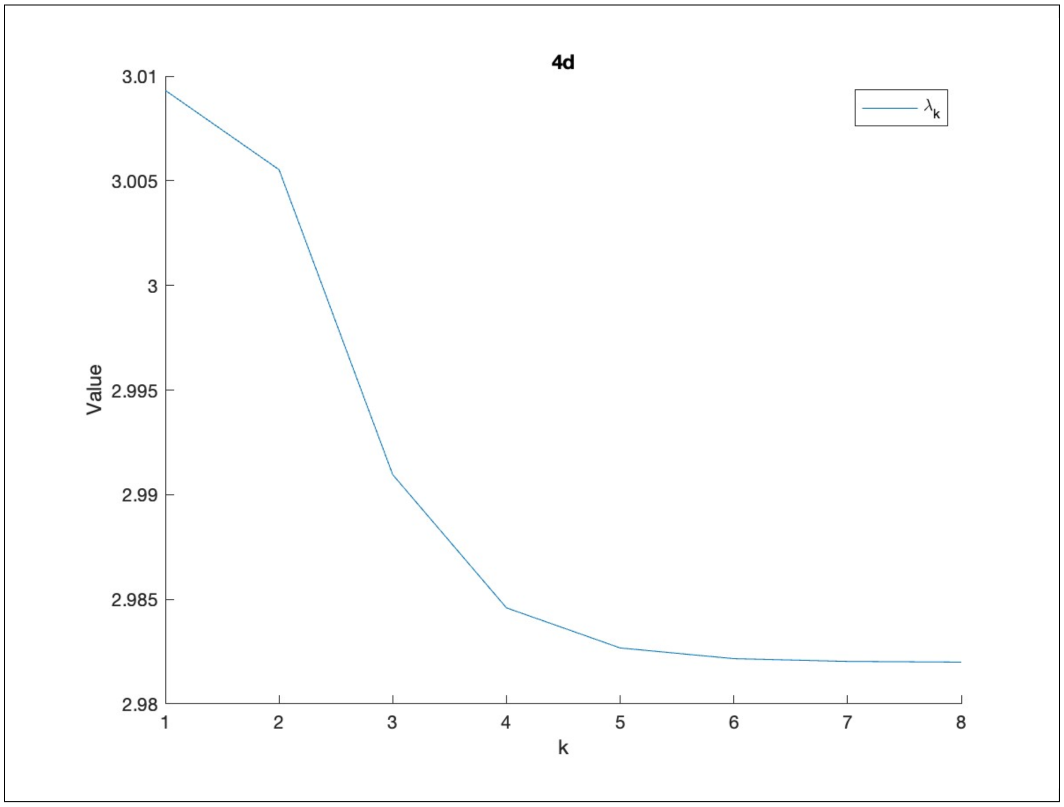
```
gcf
hold on;

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

>>
```

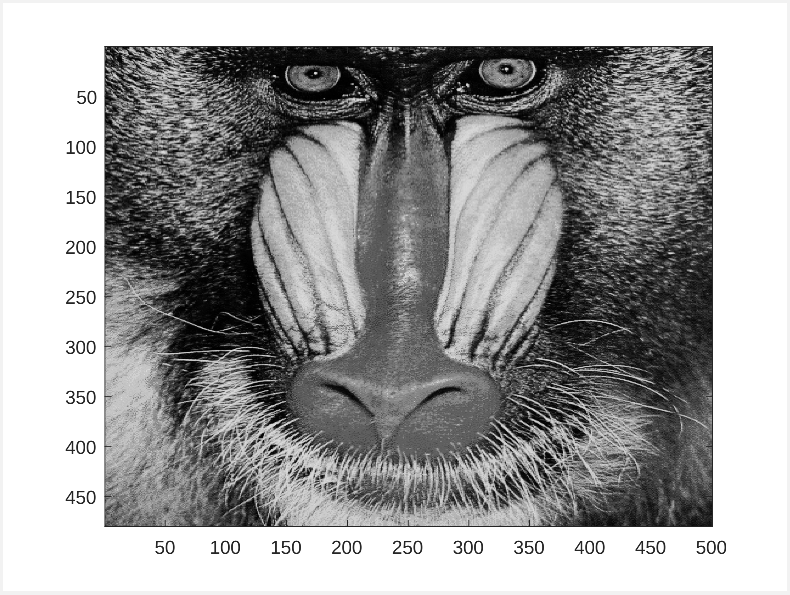
continued on next page...


4d
λ_k
Value
k
3.01
3.005
3
2.995
2.99
2.985
2.98
1
2
3
4
5
6
7
8

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).

im assuming that r that is the nubmer of eigenvalues

Solution (b).

bullshit

Solution (c).

bullshit