Mathematical Methods in Engineering and Applied Science

This webpage contains all assignments and solutions to the MMEAS course.

Problem Set 1

Problem 1

Some basic problems on matrix/vector multiplication.

(a) Calculate by hand the following matrix/vector products:
\[\begin{split} (i) \hspace{10mm} \begin{bmatrix} 2 & 1\\ \end{bmatrix} \begin{bmatrix} 1 & −2 & 1\\ 2 & 3 & 2 \end{bmatrix}; \\ \\ (ii) \hspace{10mm} \begin{bmatrix} 2 & -1 & 1\\ 3 & 0 & 4 \end{bmatrix} \begin{bmatrix} 2 & 1\\ 1 & 2\\ -1 & 0 \end{bmatrix} \end{split}\]
as a combination of columns of the left matrix aswell as a combination of rows of the right matrix.

(b) Write down a permutation matrix \(P_4\) that exchanges row 1 with row 3 and row 2with row 4. What is the connection of this matrix with the permutation matricesthat exchange only row 1 and row 3, and only row 2 and row 4?

Solution

1.

a)

as rows of right mx

i) $\begin{bmatrix} 2 & 1 \end{bmatrix}\begin{bmatrix} 1 & -2 & 1 \\ 2 & 3 & 2 \end{bmatrix} = 2\begin{bmatrix} 1 & -2 & 1 \end{bmatrix} + 1\begin{bmatrix} 2 & 3 & 2 \end{bmatrix}$

as cols of left mx

ii) $\begin{bmatrix} 2 & -1 & 1 \\ 3 & 0 & 4 \end{bmatrix}\begin{bmatrix} 2 & 1 \\ 1 & 2 \\ -1 & 0 \end{bmatrix} = \left[\begin{bmatrix} 2 \\ 3 \end{bmatrix}\cdot 2 + \begin{bmatrix} -1 \\ 0 \end{bmatrix}\cdot 1 + \begin{bmatrix} 1 \\ 4 \end{bmatrix}\cdot(-1),\ \begin{bmatrix} 2 \\ 3 \end{bmatrix}\cdot 1 + \begin{bmatrix} -1 \\ 0 \end{bmatrix}\cdot 2 + \begin{bmatrix} 1 \\ 4 \end{bmatrix}\cdot 0 \right] =$

$= \begin{bmatrix} 2 & 0 \\ 2 & 3 \end{bmatrix} = \begin{bmatrix} [2\ 1]\cdot 2 + [1\ 2]\cdot(-1) + [-1\ 0]\cdot 1 \\ [2\ 1]\cdot 3 + [1\ 2]\cdot 0 + [-1\ 0]\cdot 4 \end{bmatrix}$

as rows of right mx

b) Matrix P_4 that changes row 1 with row 3 and row 2 with row 4:

$P_4 \begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ (\dots) \end{bmatrix} = \begin{bmatrix} r_3 \\ r_4 \\ r_1 \\ r_2 \\ (\dots) \end{bmatrix} \qquad P_4 = \begin{bmatrix} 0 & 0 & 1 & 0 & & (0) \\ 0 & 0 & 0 & 1 & & \\ 1 & 0 & 0 & 0 & & \\ 0 & 1 & 0 & 0 & & \\ & (0) & & & 1 & \\ & & & & & \ddots \\ & & & & & & 1 \end{bmatrix}$

P_4' exchanges only row 1 with row 3: P_4'' exchanges row 2 and row 4:

$P_4' = \begin{bmatrix} 0 & 0 & 1 & 0 & \\ 0 & 1 & 0 & 0 & \\ 1 & 0 & 0 & 0 & \\ 0 & 0 & 0 & 1 & \\ & & & & \ddots \end{bmatrix} \qquad P_4'' = \begin{bmatrix} 1 & 0 & 0 & 0 & \\ 0 & 0 & 0 & 1 & \\ 0 & 0 & 1 & 0 & \\ 0 & 1 & 0 & 0 & \\ & & & & \ddots \end{bmatrix}$

As exchange of rows may be done consecutively ($1 \leftrightarrow 3$, then $2 \leftrightarrow 4$), $P_4 = P_4' P_4''$ or $P_4'' P_4'$.

Problem 2

Given a 3×3 matrix $A= \begin{bmatrix}a1&a2&a3\end{bmatrix}$ with columns a_i, find a matrix B that when multiplied with A, either from left or right, performs the following operations with A:

(a) exchanges row 1 and row 2;
(b) exchanges columns 1 and 2;
(c) doubles the first row;
(d) subtracts twice row 1 from row 2.
Also find the inverse of this matrix. What doesthe inverse of this B do?

Solution

(2.) $A = [a_1\ a_2\ a_3]$

a) Exchanges row 1 and row 2:

$$B = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad BA = C$$

b) Echanges ~~col~~ 2 and col 1:
Same matrix B, multiplied from right: $AB = C$

c) Doubles the first row:

$$B = \begin{bmatrix} 2 & & 0 \\ & 1 & \\ 0 & & 1 \end{bmatrix}$$

d) Subtracts twice row 1 from row 2:

$$B = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}; \quad B \begin{bmatrix} r_1 \\ r_2 \\ r_3 \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 - 2r_1 \\ r_3 \end{bmatrix} \Rightarrow \begin{bmatrix} r_1 \\ r_2 \\ r_3 \end{bmatrix} = B^{-1} \begin{bmatrix} r_1 \\ r_2 - 2r_1 \\ r_3 \end{bmatrix}$$

Problem 2

Problem 3

For matrix A = $\begin{bmatrix} 1 & 2 & 3\\ 3 & 4 & 5\\ 5 & 6 & 7 \end{bmatrix}$, determine the following:

(a) rank;
(b) eigenvalues and eigenvectors;
(c) nullspace and left nullspace;
(d) column space and row space;
(e) write A as a sum of rank-1 matrices in at least two different ways.

Solution

Problem 2
Problem 2

d) Column space and row space:

- Column space: $span(\begin{bmatrix} 1\\ 0\\ -1 \end{bmatrix}, \begin{bmatrix} 0\\ 1\\ 2 \end{bmatrix})$; equals to the row space.

e) write A as a sum of rank-1 matrices in at least two different ways:

Obviously, we can present the matrix A as a sum of single-row or single-column rank-1 matricies (with other rows or columns filled with zeros). Also, we have 3 non-colinear eigenvectors, that can be our matrix basis in the canonical decomposition (diagonalization):

$$A = UDU^{-1} = \sum\limits_{i=1}\limits^{3}{d_i\lambda_i\lambda_i^T}$$

Problem 4

The columns of matrix $C = \begin{bmatrix} 2 & 2 & 1 & 1 & 2 & 2 & 1 & 1\\ 1 & 2 & 2 & 1 & 1 & 2 & 2 & 1\\ 1 & 1 & 1 & 1 & 2 & 2 & 2 & 2 \end{bmatrix}$ represent vertices of a cube. Describe transformations of the cube that result from the action on C of the following three matrices:

$A_1 = \begin{bmatrix} 1 & 2 & 2\\ 0 & 2 & 2\\ 0 & 0 & 3 \end{bmatrix}, \hspace{3mm} A_2 = \begin{bmatrix} 1 & 2 & 2\\ 0 & 2 & 2\\ 0 & 0 & 0 \end{bmatrix}, \hspace{3mm} A3 = \begin{bmatrix} 0 & 2 & 2\\ 0 & 2 & 2\\ 0 & 0 & 0 \end{bmatrix}.$

Relate the results to the ranks of A_k and to the dimensions and bases of the four fundamental subspaces of A_k. Is there a 3×3 matrix A that can transform a cube into a tetrahedron? Explain.

Solution

Firstly, let us see what every transformation does with the cube.

```
# Creating matricies

A1 = np.matrix([
    [1, 2, 2],
    [0, 2, 2],
    [0, 0, 3]
])

A2 = np.matrix([
    [1, 2, 2],
    [0, 2, 2],
    [0, 0, 0]
])

A3 = np.matrix([
    [0, 2, 2],
    [0, 2, 2],
    [0, 0, 0]
])

C = np.matrix([
    [2, 2, 1, 1, 2, 2, 1, 1],
    [1, 2, 2, 1, 1, 2, 2, 1],
    [1, 1, 1, 1, 2, 2, 2, 2]
])

#Applying A matricies to the cube

B1 = A1*C
B2 = A2*C
B3 = A3*C

#Plotting

%config InlineBackend.figure_format = 'svg'
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(xs = C[0], ys = C[1], zs = C[2])
ax.scatter(xs = B1[0], ys = B1[1], zs = B1[2], color='red', depthshade = True)
ax.scatter(xs = B2[0], ys = B2[1], zs = B2[2], color='green', depthshade = True)
ax.scatter(xs = B3[0], ys = B3[1], zs = B3[2], color='yellow', depthshade = True)
ax.set_xticks(np.arange(0, 11, 2))
ax.set_yticks(np.arange(0, 11, 2))
ax.set_zticks(np.arange(0, 7, 1))
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')

plt.title('Cube Transformations')

plt.legend(('Initial Cube', 'A1', 'A2', 'A3'))
```

```
<matplotlib.legend.Legend at 0x1a848bc78c8>
```

C:/Users/backg/OneDrive/Workspace/Study/Skoltech/MMEAS/MMEASbook/_build/jupyter_execute/ProblemSet1_11_1.svg

The points on this 3D plot represent the verticies of the initial cube (blue) and resulting shapes. As we see,

- A1 combines some stretching and rotation;
- A2 projects the first transformation on the XY plane;
- A3 projects the first (or the second) transformation on a line.

Now, let's derive the properties of each matrix A_i.
$$ rg(A_1) = 3;\hspace{3mm}rg(A_2) = 2;\hspace{3mm}rg(A_3) = 1 $$
It follows from the Main theorem of Linear Algebra that
$$ rg(ker(A_1)) = 0;\hspace{3mm}rg(ker(A_2)) = 1;\hspace{3mm}rg(ker(A_3)) = 2 $$
That means,

- A_1 maps $\mathbb{R}^3$ to $\mathbb{R}^3$
- A_2 maps $\mathbb{R}^3$ to $\mathbb{R}^2$
- A_3 maps $\mathbb{R}^3$ to $\mathbb{R}^1$

That corresponds to our findings in the previous section.

Let us find the four fundamental subspaces for each A_i operator:

- A_1 is a full-rank matrix, that means nullspace is empty.

The same applies to A_1^T. A_1 column space is $ \begin{bmatrix} 1\\ 0\\ 0 \end{bmatrix} , \begin{bmatrix} 0\\ 2\\ 0 \end{bmatrix}, \begin{bmatrix} 0\\ 0\\ 3 \end{bmatrix} $, as well as for A_1^T (row space of A_1).

- for A_2 we solve a simple set of linear equations and receive the following result:

$$\begin{split}Null(A_2) = span(\begin{bmatrix} 0\\ 1\\ -1 \end{bmatrix}) ; \hspace{3mm} Null(A_2^T) = span(\begin{bmatrix} 0\\ 1\\ -1 \end{bmatrix}). \end{split}$$
The column space of A_2 is $ \begin{bmatrix} 1\\ 0\\ 0 \end{bmatrix}, \begin{bmatrix} 0\\ 2\\ 0 \end{bmatrix} $, the row space is the same.

- for A_3, we have two-dimensional null space: $Null(A_3) = span(\begin{bmatrix} 1\\ 1\\ -1 \end{bmatrix}, \begin{bmatrix} 1\\ -1\\ 1 \end{bmatrix})$. It coincides with the nullspace of A_3^T.

The column space of A_3 is $span(\begin{bmatrix} 1\\ 1\\ 0 \end{bmatrix})$, as well as the row space. This result is easily observed on the visualisation: the result of the 3rd transformation lies on the $y = x$ line on the XY plane.

```
def print_eig(A, number):
    val, vec = np.linalg.eig(A)
    print("Eigenvalues and eigenvectors for matrix A{}:".format(number))
    print(val)
    print(vec)
    print()

#print_eig(A1, 1)
#print_eig(A2, 2)
#print_eig(A3, 3)
```

On transforming a cube into a tetrahedron: Linear transformation implies that we can describe it only acting on basis vectors. That means, I suppose there is no such 3×3 matrix to perform this operation.

Problem 5

For matrix $A= \begin{bmatrix} 2 & 1\\ 1 & 2 \end{bmatrix} $ determine which unit vector x_M is stretched the most and which x_m the least and by how much. That is, find x such that $y=Ax$ has the largest (or smallest) possible Euclidian length. You can do this by calculus methods, e.g. using Lagrange multipliers. Relate your findings to eigenvalues and eigenvectors of A.

Solution

Firstly, we SVD the matrix A:

```
U:
 [[-0.70710678 -0.70710678]
 [-0.70710678  0.70710678]]
E: [3. 1.]
V*:
 [[-0.70710678 -0.70710678]
 [-0.70710678  0.70710678]]
```

We see that the matricies U and $V*$ may be rewritten as a combination of the following matricies:
\[\begin{split} U = V^* = \begin{bmatrix} -\frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2}\\ -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix} = \frac{\sqrt{2}}{2} \begin{bmatrix} -1 & -1\\ -1 & 1 \end{bmatrix} = \frac{\sqrt{2}}{2} \begin{bmatrix} 1 &

-1\\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} -1 & 0\\ 0 & 1 \end{bmatrix} =\\ =cR\times M, \end{split}\]
where $c = \frac{\sqrt{2}}{2}$, cR is a rotation matrix with $\varphi = \frac{1}{4}\pi$, M is a mirror operator that mirrors about X axis.

The matrix Σ has diagonal elements $\begin{bmatrix} 3 & 1 \end{bmatrix}$ and represents stretching times 3 along X axis.

```
M = np.matrix([
    [-1, 0],
    [0, 1]
])
c = np.sqrt(2)/2
R = np.matrix([
    [1, -1],
    [1, 1]
])

#c*R*M

#c*R
```

After the stretch has been completed, the matrix U does the reverse transformation: mirrors the X' axis and rotates $\frac{1}{4}\pi$ counterclockwise.

Now we can say that the most stretched vector x_M will be the $\frac{\sqrt{2}}{2} \begin{bmatrix} 1\\ 1 \end{bmatrix}$ (or it's mirrored counterpart) unit vector, which during the transformations is aligned along the direction of the stretch operation and it's Euclidian length will be 3, and the least stretched vector x_m is the $\frac{\sqrt{2}}{2} \begin{bmatrix} 1\\ -1 \end{bmatrix}$ (or it's mirrored counterpart), which lies perpendicular to the axis of the stretch operation. It's Euclidian length remains 1.

Now let's look at the Eigenvalues and Eigenvectors of the matrix (the values and the vectors can be easily calculated by hand):

```
Eigenvalues:  [3. 1.]
Eigenvectors:
[[ 0.70710678 -0.70710678]
 [ 0.70710678  0.70710678]]
```

The result coincides with the result derived from SV decomposition: the $\frac{\sqrt{2}}{2} \begin{bmatrix} 1\\ 1 \end{bmatrix}$ vector is stretched 3 times, the $\frac{\sqrt{2}}{2} \begin{bmatrix} 1\\ -1 \end{bmatrix}$ vector remains the same.

Problem 6

Find eigenvalues and eigenvectors of the following matrices:
(a) $A_1= \begin{bmatrix} 0 & 1\\ -1 & 0 \end{bmatrix}$. If x is any real vector, how is $y=A_1x$ related to x geometrically?

(b) $A_2= \begin{bmatrix} 1 & 1 & 0\\ 0 & 1 & 1\\ 0 & 0 & 1 \end{bmatrix}$. What is the rank of A_2? How many eigenvectors are there?

Solution

The calculation of eigenvalues and eigenvectors may be performed by hand easily, as well as with *NumPy* package.

We start with matrix A_1:

```
Eigenvalues and eigenvectors for matrix A1:
[0.+1.j 0.-1.j]
[[0.70710678+0.j         0.70710678-0.j        ]
 [0.        +0.70710678j 0.        -0.70710678j]]
```

Instantly from one glance at the matrix A_1, as well as by seeing the result of eigen_operations, we derive that the matrix conducts a rotation. There are no real eigenvectors, obviously, as there are no vectors x that would be collinearly translated into another vectors y.

Now, let’s analyze the matrix A_2. It is obvious, that $rg(A_2) = 1$, because we can perform simple row operations ($r_2 = r_2 - r_3; \hspace{3mm} r_1 = r_1 - (r_2 - r_3)$) to get the resulting matrix $\begin{bmatrix} 1 & 0 & 0\\ 0 & 0 & 0\\ 0 & 0 & 0 \end{bmatrix}$.

```
Eigenvalues and eigenvectors for matrix A2:
[1. 1. 1.]
[[ 1.00000000e+00 -1.00000000e+00  1.00000000e+00]
 [ 0.00000000e+00  2.22044605e-16 -2.22044605e-16]
 [ 0.00000000e+00  0.00000000e+00  4.93038066e-32]]
```

There is one eigenvalue $\lambda = 1$ of multiplicity 3, and only one eigenvector $h = \begin{bmatrix} 1\\ 0\\ 0 \end{bmatrix}$ This matrix is a linear operator that projects on YZ plane.

Problem Set 2

Problem 1

Consider linear system

$$\begin{split} \begin{align*} 2x_1 + x_2 = 1\\ x_1 + 2x_2 + x_3 = 2\\ x_2 + 2x_3 = 3 \end{align*} \end{split}$$

(a) Find the LU factorization of the coefficient matrix A. Show that $U = DL^T$ with D diagonal and thus $A=LDL^T$. Find the exact solution using the LU factorization.

(b) Solve the system using Jacobi and Gauss-Seidel iterations. How many iterations are needed to reduce the relative error of the solution to 10^{-8}?

(c) Plot in semilog scales the relative errors by both methods as a function of the number of iterations.

(d) Explain the convergence rate. Which of the methods is better and why?

Solution

Let us perform the LU factorization by hand:

$$\begin{split} \begin{align*} \begin{bmatrix} 2 & 1 & 0\\ 1 & 2 & 1\\ 0 & 1 & 2 \end{bmatrix} \Rightarrow (R_2 - \frac{1}{2}R_1) \begin{bmatrix} 2 & 1 & 0\\ 0 & \frac{3}{2} & 1\\ 0 & 1 & 2 \end{bmatrix} \Rightarrow (R_3 - \frac{2}{3}R_2) \begin{bmatrix} 2 & 1 & 0\\ 0 & \frac{3}{2} & 1\\ 0 & 0 & \frac{4}{3} \end{bmatrix} = U; \hspace{10mm} L = \begin{bmatrix} 1 & 0 & 0\\ \frac{1}{2} & 1 & 0\\ 0 & \frac{2}{3} & 1 \end{bmatrix} \end{align*} \end{split}$$

And check it via *numpy*:

```python
import numpy as np
from fractions import Fraction
from scipy import linalg as lin
import sympy as sp
from sympy import abc

A = np.matrix([
    [2, 1, 0],
    [1, 2, 1],
    [0, 1, 2]
])

def bmatrix(a):
    """Returns a LaTeX bmatrix

    :a: numpy array
    :returns: LaTeX bmatrix as a string
    """
    if len(a.shape) > 2:
        raise ValueError('bmatrix can at most display two dimensions')
    lines = str(a).replace('[', '').replace(']', '').splitlines()
    rv = [r'\begin{bmatrix}']
    rv += ['  ' + ' & '.join(l.split()) + r'\\' for l in lines]
    rv +=  [r'\end{bmatrix}']
    return '\n'.join(rv)

E, L, U = lin.lu(A, permute_l = False)
L = np.matrix(L)
U = np.matrix(U)
print("L:")
print(L)
print("\nU:")
print(U)
```

```
L:
[[1.         0.         0.        ]
 [0.5        1.         0.        ]
 [0.         0.66666667 1.        ]]

U:
[[2.         1.         0.        ]
 [0.         1.5        1.        ]
 [0.         0.         1.33333333]]
```

Let us show, that $U = DL^T$:

$$\begin{split} L^T = \begin{bmatrix} 1 & \frac{1}{2} & 0\\ 0 & 1 & \frac{2}{3}\\ 0 & 0 & 1 \end{bmatrix} ;\hspace{10mm} D = \begin{bmatrix} 2 & 0 & 0\\ 0 & \frac{3}{2} & 0\\ 0 & 0 & 1 \end{bmatrix} ;\hspace{10mm} U = DL^T \end{split}$$

That means, $A = LU = LDL^T$.

Now, let us solve the system using LU factorization:

$$\begin{split} Ax = b \hspace{3mm} \Rightarrow \hspace{3mm} LUx = b; \\ \begin{align*} \begin{cases} Ux = y;\\ Ly = b \end{cases} \end{align*} \end{split}$$

$$\begin{split} \begin{bmatrix} 1 & 0 & 0\\ \frac{1}{2} & 1 & 0\\ 0 & \frac{2}{3} & 1\\ \end{bmatrix} \begin{bmatrix} y_1\\ y_2\\ y_3 \end{bmatrix} = \begin{bmatrix} 1\\ 2\\ 3 \end{bmatrix} \Rightarrow \begin{cases} y_1 = 1\\ y_2 = \frac{3}{2}\\ y_3 = 2 \end{cases} \end{split}$$

$$\begin{split} \begin{bmatrix} 2 & 1 & 0\\ 0 & \frac{3}{2} & 1\\ 0 & 0 & \frac{4}{3} \end{bmatrix} \begin{bmatrix} x_1\\ x_2\\ x_3 \end{bmatrix} = \begin{bmatrix} 1\\ \frac{3}{2}\\ 2 \end{bmatrix} \Rightarrow \begin{cases} x_1 = \frac{1}{2}\\ x_2 = 0\\ x_3 = \frac{3}{2} \end{cases} \end{split}$$

$$\begin{split} x = \begin{bmatrix} 1/2\\ 0\\ 3/2 \end{bmatrix} \end{split}$$

Next we solve the system using Jacobi and Gauss-Seidel methods:

```python
import matplotlib.pyplot as plt
import copy
```

```python
D = np.matrix(np.diag(np.diag(A)))
U = np.matrix(np.triu(A-D))
L = np.matrix(np.tril(A-D))

x_exact = np.matrix([1/2, 0, 3/2]).T
tol = 1e-8
err = 1
x_init = np.matrix([1, 1, 1]).T #initial x
b = np.matrix([1, 2, 3]).T

def Jacobi(L, D, U, x_init, b, err, tol):

    x = copy.deepcopy(x_init)

    max_iters = 500

    err_iter = np.array([])
    err_exact_iter = np.array([])


    iters = 0
    while ((err > tol) and (iters <= max_iters)) :
        iters = iters + 1
        bb = b - (U + L)*x
        x_new = np.linalg.solve(D, bb)
        err = np.linalg.norm(x_new - x)/np.linalg.norm(x)
        err_exact = np.linalg.norm(x_new - x_exact)

        err_iter = np.append(err_iter, err)
        err_exact_iter = np.append(err_exact_iter, err_exact)

        x = x_new

    return err_iter, err_exact_iter, x_new, iters

def Gauss_Seidel(L, D, U, x_init, b, err, tol):

    x = copy.deepcopy(x_init)

    max_iters = 250

    err_iter = np.array([])
    err_exact_iter = np.array([])

    iters = 0
    while ((err > tol) and (iters <= max_iters)):
        iters = iters + 1

        bb = b - U*x
        x_new = np.linalg.solve(D+L, bb)
        err = np.linalg.norm(x_new - x)/np.linalg.norm(x)
        err_exact = np.linalg.norm(x_new - x_exact)

        err_iter = np.append(err_iter, err)
        err_exact_iter = np.append(err_exact_iter, err_exact)

        x = x_new

    return err_iter, err_exact_iter, x_new, iters
```

```python
%config InlineBackend.figure_format='svg'

fig = plt.figure()
ax = plt.gca()

errs_j, ex_errs_j, x_j, i_j = Jacobi(L, D, U, x_init, b, err, tol)
plt.plot(errs_j)

errs_g, ex_errs_g, x_g, i_g = Gauss_Seidel(L, D, U, x_init, b, err, tol)
plt.plot(errs_g)

ax.set_yscale('log')


ax.legend(('Jacobi', 'Gauss_Seidel'))
ax.set_xlabel('Iterations')
ax.set_ylabel('Relative error')
ax.set_yticks([1e-7, 1e-5, 1e-3, 1e-1])
ax.tick_params(axis='y', which='minor')
ax.grid(which='both')
```

C:/Users/backg/OneDrive/Workspace/Study/Skoltech/MMEAS/MMEASbook/_build/jupyter_execute/ProblemSet2_13_0.svg

As we see, it took 26 iterations for Gauss-Seidel method to reach the target tolerance of 10^{-8}, while Jacobi method required 54 iterations.

This can be explained by the following: in Gauss-Seidel, as soon as we acquire a new iteration of a vector x component $x_i^{(k+1)}$, we instantly utilize this updated value in the computation of the following components: $x_i^{(k+1)} = f(x_1^{(k+1)}, ..., x_{i-1}^{(k+1)}, x_{i+1}^{(k)}, ..., x_n^{(k)})$. In Jacobi, we calculate the new vector x^{k+1} relying solely on the result of the previous iteration: $x_i^{k+1} = f(x_j^{(k)}),\hspace{3mm}j \neq i$.

Problem 2

Factor these two matrices A into $S\Lambda S^{-1}$:

$$\begin{split} A1 = \begin{bmatrix} 1 & 2\\ 0 & 3 \end{bmatrix} ,\hspace{3mm} A2 = \begin{bmatrix} 1 & 2\\ 0 & 3 \end{bmatrix} \end{split}$$

Using that factorization, find for both: (a) A^3; (b) A^{-1}.

Solution

Firstly, we find the eigenvalues and eigenvectors:

$$\begin{split} \begin{bmatrix} 1-\lambda & 2\\ 0 & 3-\lambda \end{bmatrix} = 0 \end{split}$$

By performing simple calculations by hand, we obtain:

$$\begin{split} h_1 = \begin{bmatrix} 1\\ 0 \end{bmatrix} ,\hspace{3mm} \lambda_1 = 1 \\ h_2 = \frac{\sqrt{2}}{2} \begin{bmatrix} 1\\ 1 \end{bmatrix} ,\hspace{3mm} \lambda_2 = 3 \end{split}$$

> **Note**
>
> We normalized the eigenvectors

Now let's perform a check with *numpy*:

```
A1 = np.matrix([
    [1, 2],
    [0, 3]
])

val1, vec1 = np.linalg.eig(A1)
print("E_values A1:")
print(val1)
print("E_vectors A1:")
print(vec1)
```

```
E_values A1:
[1. 3.]
E_vectors A1:
[[1.         0.70710678]
 [0.         0.70710678]]
```

As we have our vectors and values, we can construct Λ and S, S^{-1} matrices:

```
Lambda = np.matrix(np.diag(val1))
S = vec1
Si = np.linalg.inv(S)

print("S:")
print(S)
print("Lambda:")
print(Lambda)
print("S_inverse:")
print(Si)
```

```
S:
[[1.         0.70710678]
 [0.         0.70710678]]
Lambda:
[[1. 0.]
 [0. 3.]]
S_inverse:
[[ 1.         -1.        ]
 [ 0.          1.41421356]]
```

$$\begin{split}S = \begin{bmatrix} 1 & \sqrt{2}/2\\ 0 & \sqrt{2}/{2} \end{bmatrix} ;\hspace{3mm} \Lambda = \begin{bmatrix} 1 & 0\\ 0 & 3 \end{bmatrix} ;\hspace{3mm} S^{-1} = \begin{bmatrix} 1 & -1\\ 0 & 2/\sqrt{2} \end{bmatrix} \end{split}$$

From now, we can fing the A^3 powered matrix by simply multiplying the decomposition:

$$\begin{split} A^3 = S\Lambda S^{-1} S \Lambda S^{-1} S \Lambda S^{-1} = \\ \hspace{1mm} \\ = S \Lambda^3 S^{-1}. \end{split}$$

And for Λ it is easy to power because it is a diagonal matrix.

$$\begin{split} \Lambda ^3 = \begin{bmatrix} 1 & 0\\ 0 & 9 \end{bmatrix} , \\ \hspace{1mm} \\ S\Lambda ^3 S^{-1} = \begin{bmatrix} 1 & 8\\ 0 & 9 \end{bmatrix} \end{split}$$

For inverse matrix:

$$ S\Lambda S^{-1} A_1^{-1} = E, \Rightarrow A_1^{-1} = S^{-1} \Lambda ^{-1} S $$

We already have S and S^{-1}, and for diagonal Λ the inverse matrix contains the inverse diagonal elements of Λ:

$$\begin{split} \Lambda ^{-1}= \begin{bmatrix} 1 & 0\\ 0 & 1/3 \end{bmatrix} \end{split}$$

So we easily find A_1^{-1}:

$$\begin{split} A_1^{-1} = \begin{bmatrix} 1 & \sqrt{2}{3}\\ 0 & 1/3 \end{bmatrix} \end{split}$$

Now let's look at the second matrix A_2. Instantly we notice it is a rank-1 matrix, thus, A_2^{-1} matrix doesn't exist.

$$\begin{split} h_1 = \begin{bmatrix} 1\\ -1 \end{bmatrix} ,\hspace{3mm} \lambda_1 = 0 \\ \hspace{1mm} \\ h_2 = \begin{bmatrix} 1\\ 3 \end{bmatrix} ,\hspace{3mm} \lambda_2 = 4 \end{split}$$

It's eigenvectors are non-collinear and form a basis in 2-dimensional space. Thus we can perform the factorization.

$$\begin{split} S = \begin{bmatrix} 1 & 1\\ -1 & 3 \end{bmatrix} ;\hspace{3mm} \Lambda = \begin{bmatrix} 0 & 0\\ 0 & 4 \end{bmatrix} ;\hspace{3mm} S^{-1} = \begin{bmatrix} 3/4 & -1/4\\ 1/4 & 1/4 \end{bmatrix} \end{split}$$

For A_2^3:

$$\begin{split} A_2^3 = \begin{bmatrix} 64 & 64\\ 192 & 192 \end{bmatrix} . \end{split}$$

Problem 3

Given a system $Ax = b$ with

$$\begin{split} A = \begin{bmatrix} 1 & -1 & -3\\ 2 & 3 & 4\\ -2 & 1 & 4\\ \end{bmatrix} ,\hspace{3mm} b = \begin{bmatrix} 3\\ a\\ -1\\ \end{bmatrix} , \end{split}$$

for which a there is a solution? Find the general solution of the system for that a.

Solution

Let's check the matrix' rank:

```
a = abc.symbols('a')

A = sp.Matrix([
    [1, -1, -3],
    [2, 3, 4],
    [-2, 1, 4]
])

b = sp.Matrix([
    3, a, -1]
)

print('Rank: {}'.format(A.rank()))
A.rref()[0]
```

```
Rank: 2
```

$$\begin{split}\displaystyle \left[\begin{matrix}1 & 0 & -1\\0 & 1 & 2\\0 & 0 & 0\end{matrix}\right]\end{split}$$

This matrix is a rank-2 matrix. Let's find it's left nullspace, write down the solvability condition and find the appropriate a. Starting with the left null-space:

```
y = A.T.nullspace()[0]
y
```

$$\begin{split}\displaystyle \left[\begin{matrix}\frac{8}{5}\\\frac{1}{5}\\1\end{matrix}\right]\end{split}$$

We want to fulfill the following condition:

$$ y^Tb = 0 $$

```
ans = sp.solve(y.T*b, a)
ans
```

```
{a: -19}
```

As we see, the system is solvable with $a = -19$. Let us perform some check:

```
bs = b.subs(a, -19)
y.T*bs
```

$$\displaystyle \left[\begin{matrix}0\end{matrix}\right]$$

Now as we have our vector b with wich the system is solvable, we may find the general solution for the system:

```
a, b, c = abc.symbols('a b c')
system = A, bs
sol = sp.linsolve((A, bs), a, b, c); sol
```

$$\displaystyle \left\{\left(c - 2, \ - 2 c - 5, \ c\right)\right\}$$

With $c \in \mathbb{R}$ we get our solution:

$$\begin{split} x = \begin{bmatrix} -2\\ -5\\ 0 \end{bmatrix} + \begin{bmatrix} 1\\ -2\\ 1 \end{bmatrix} \cdot c. \end{split}$$

Problem 4