Problem Set 2

Contents

Problem 1

Consider linear system

$$\begin{aligned} 2x_1 + x_2 &= 1 \\ x_1 + 2x_2 + x_3 &= 2 \\ x_2 + 2x_3 &= 3 \end{aligned}$$

(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:

$$\begin{bmatrix} 2 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 2 \end{bmatrix} \Rightarrow (R_2 - \frac{1}{2}R_1) \begin{bmatrix} 2 & 1 & 0 \\ 0 & \frac{3}{2} & 1 \\ 0 & 1 & 2 \end{bmatrix} \Rightarrow (R_3 - \frac{2}{3}R_2) \begin{bmatrix} 2 & 1 & 0 \\ 0 & \frac{3}{2} & 1 \\ 0 & 0 & \frac{4}{3} \end{bmatrix} = U; \qquad L = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{2} & 1 & 0 \\ 0 & \frac{2}{3} & 1 \end{bmatrix}$$

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 $U = DL^T$:

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

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

Now, let us solve the system using LU factorization:

$$Ax = b \quad \Rightarrow \quad LUx = b;$$
$$\begin{cases} Ux = y; \\ Ly = b \end{cases}$$

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

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

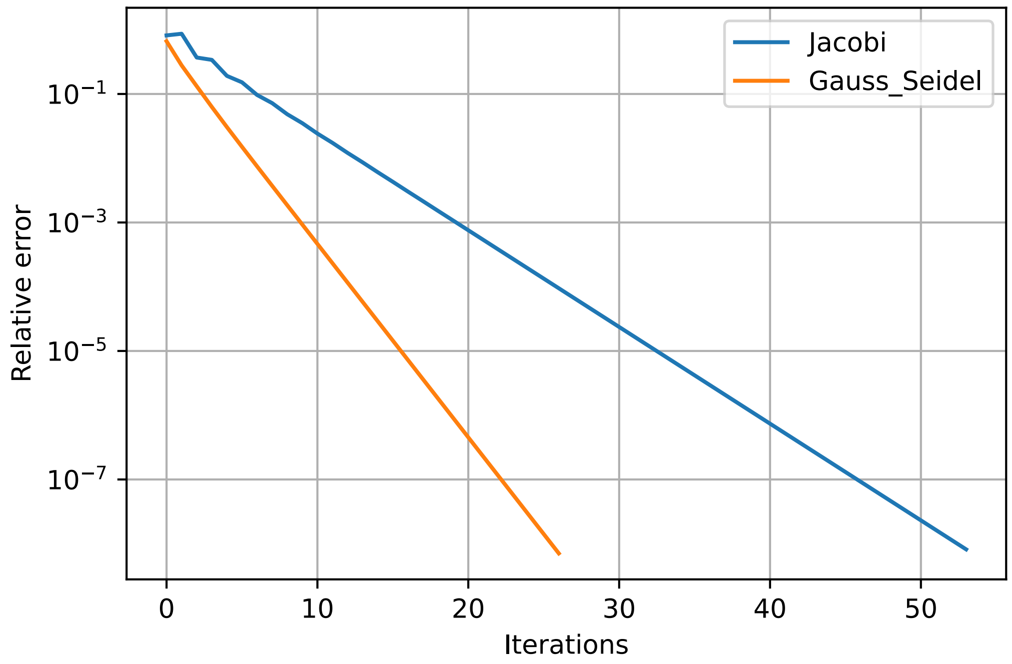
$$x = \begin{bmatrix} 1/2 \\ 0 \\ 3/2 \end{bmatrix}$$

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')
```



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: $x_i^{(k+1)} = f(x_1^{(k+1)}, \dots, x_{i-1}^{(k+1)}, x_{i+1}^{(k)}, \dots, x_n^{(k)})$. In Jacobi, we calculate the new vector x^{k+1} relying solely on the result of the previous iteration: $x_i^{k+1} = f(x_j^{(k)}), \quad j \neq i$.

Problem 2

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

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

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

Solution

Firstly, we find the eigenvalues and eigenvectors:

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

By performing simple calculations by hand, we obtain:

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

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]]
```

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

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

$$A^3 = S\Lambda S^{-1} S\Lambda S^{-1} S\Lambda S^{-1} =$$

$$= S\Lambda^3 S^{-1}.$$

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

$$\Lambda^3 = \begin{bmatrix} 1 & 0 \\ 0 & 9 \end{bmatrix},$$

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

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:

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

So we easily find A_1^{-1}:

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

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

$$h_1 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \quad \lambda_1 = 0$$

$$h_2 = \begin{bmatrix} 1 \\ 3 \end{bmatrix}, \quad \lambda_2 = 4$$

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

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

For A_2^3:

$$A_2^3 = \begin{bmatrix} 16 & 16 \\ 48 & 48 \end{bmatrix}.$$

Problem 3

Given a system $Ax = b$ with

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

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
```

$$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 2 \\ 0 & 0 & 0 \end{bmatrix}$$

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
```

$$\begin{bmatrix} \frac{8}{5} \\ \frac{1}{5} \\ 1 \end{bmatrix}$$

We want to fulfill the following condition:

$$y^T b = 0$$

```
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
```

$$[0]$$

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
```

$$\{(c-2,\ -2c-5,\ c)\}$$

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

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

Problem 4

Consider the system of linear differential equations:

$$\frac{du}{dt} = Au, \quad A = \begin{bmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix}.$$

(a) Using the spectral factorization of A, determine the general solution of the system.
(b) Find the behavior of the solution at large t if the initial condition is $u(0) = \begin{bmatrix} 1 & -1 & 1 \end{bmatrix}^T$

Solution

Check the rank of the matrix:

```
A = sp.Matrix([
    [1, -1, 0],
    [-1, 2, -1],
    [0, -1, 1]
])

A.rank()
```

```
2
```

Spectral decomposition:

```
S, L = A.diagonalize()
```

```
print("S:")
S
```

```
S:
```

$$\begin{bmatrix} 1 & -1 & 1 \\ 1 & 0 & -2 \\ 1 & 1 & 1 \end{bmatrix}$$

```
print("Lambda:")
L
```

```
Lambda:
```

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 3 \end{bmatrix}$$

```
print("S^-1:")
S**-1
```

```
S^-1:
```

$$\begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ -\frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{6} & -\frac{1}{3} & \frac{1}{6} \end{bmatrix}$$

The general solution may be written as follows:

$$\frac{du}{dt} = S\Lambda S^{-1}u$$
$$S^{-1}\frac{du}{dt} = \Lambda S^{-1}u$$
$$\frac{dv}{dt} = \Lambda v, \quad v = S^{-1}u \Rightarrow$$
$$\Rightarrow v = e^{\Lambda t}v_0; \quad u = Se^{\Lambda t}S^{-1}u_0$$

```
x, y, z, t = abc.symbols('x, y, z, t')
x0, y0, z0 = abc.symbols('x0, y0, z0')
u = sp.Matrix([x, y, z])

v = S**-1*u

elambdat = (L*t).exp()

u0 = sp.Matrix([x0, y0, z0])

u_ans = S*elambdat*S**-1*u0; u_ans.simplify()
u_ans
```

$$\begin{bmatrix} \frac{x_0(e^{3t}+3e^t+2)}{6} + \frac{y_0(1-e^{3t})}{3} + \frac{z_0(e^{3t}-3e^t+2)}{6} \\ \frac{x_0(1-e^{3t})}{3} + \frac{y_0(2e^{3t}+1)}{3} + \frac{z_0(1-e^{3t})}{3} \\ \frac{x_0(e^{3t}-3e^t+2)}{6} + \frac{y_0(1-e^{3t})}{3} + \frac{z_0(e^{3t}+3e^t+2)}{6} \end{bmatrix}$$

With $u_0 = [1 \quad -1 \quad 1]$:

```
u_ans_num = u_ans.subs({x0: 1, y0: -1, z0: 1})
u_ans_num
```

$$\begin{bmatrix} \frac{2e^{3t}}{3} + \frac{1}{3} \\ \frac{1}{3} - \frac{4e^{3t}}{3} \\ \frac{2e^{3t}}{3} + \frac{1}{3} \end{bmatrix}$$

We can decompose the solution into two vectors:

```
vec_c = sp.Matrix([1/3, 1/3, 1/3])
u_e = u_ans_num - vec_c
u_e
```

$$\begin{bmatrix} \frac{2e^{3t}}{3} \\ -\frac{4e^{3t}}{3} \\ \frac{2e^{3t}}{3} \end{bmatrix}$$

$$u = \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix} + \frac{2}{3}\begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix} e^{3t}$$

With large t, the solution will approach the $[1 \quad -2 \quad 1]^T$ direction.

Problem 5

For matrix $A = \begin{bmatrix} 2021 & 20 & 0 \\ 20 & 2021 & 21 \\ 0 & 21 & 2021 \end{bmatrix}$ and vector $b = \begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix}$, what is the most likely direction of vector $x = A^{2021}b$?

Solution

Diagonalization:

```
A = sp.Matrix([
    [2021, 20, 0],
    [20, 2021, 21],
    [0, 21, 2021]
])

b = sp.Matrix([2, 1, 0])

S, L = A.diagonalize()
```

$$A^{2021} = S\Lambda^{2021}S^{-1}$$

```
print("Lambda:")
L
```

```
Lambda:
```

$$\begin{bmatrix} 1992 & 0 & 0 \\ 0 & 2021 & 0 \\ 0 & 0 & 2050 \end{bmatrix}$$

```
print("S:")
S
```

```
S:
```

$$\begin{bmatrix} 20 & -21 & 20 \\ -29 & 0 & 29 \\ 21 & 20 & 21 \end{bmatrix}$$

Note

we will extract the common denominator from the Λ, as we are only interested in the direction (this step is not necessary but is helpful)

```
denom = 2050
L_denom = L/denom
L_denom
```

$$\begin{bmatrix} \frac{996}{1025} & 0 & 0 \\ 0 & \frac{2021}{2050} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

As we see, upon exponentiation of the matrix, the 3^{rd} diagonal element of the matrix Λ will stay equal to 1, while the 1^{st} and the 2^{nd} tend to zero. So we may neglect the λ_1 and λ_2 in comparison with λ_3.

We will substitute the matrix with the following pseudo-matrix:

$$\Lambda_{pseudo} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

because the denominator is a scalar that can be removed to the left of the whole equation chain:

$$x = D^{2021}S\Lambda_D^{2021}S^{-1}b \approx D^{2021}S\Lambda_{pseudo}^{2021}S^{-1}b$$

And to know the direction, we may get rid of the denominator completely, also noting that $\Lambda_{pseudo}^{2021} = \Lambda_{pseudo}$:

$$x_{dir} = S\Lambda_{pseudo}S^{-1}b$$

The only thing left is to explicitly calculate the x_{dir} vector, conjure a bit with the denominators (remember, we don't care about the length of the vector) and find the direction:

```
L_pseudo = sp.diag([0, 0, 1], unpack=True)
x = S*L_pseudo*S**-1*b
x*1682/1380
```

$$\begin{bmatrix} 1 \\ \frac{29}{20} \\ \frac{21}{20} \end{bmatrix}$$

$$x_{dir} = \begin{bmatrix} 1 \\ \frac{29}{20} \\ \frac{21}{20} \end{bmatrix} \approx \begin{bmatrix} 1 \\ 1.5 \\ 1 \end{bmatrix}.$$

Problem 6

Four unit masses are joined by springs of unit spring constant on a ring of unit raduis as shown in the figure.

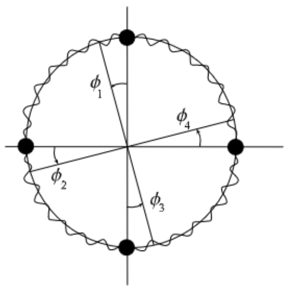


Fig. 1 System illustration

Convince yourself that Newton's second law for the masses results in

$$\begin{aligned}
\ddot{\phi}_1 &= -2\phi_1 + \phi_2 + \phi_4 \\
\ddot{\phi}_2 &= -2\phi_2 + \phi_3 + \phi_1 \\
\ddot{\phi}_3 &= -2\phi_3 + \phi_4 + \phi_2 \\
\ddot{\phi}_4 &= -2\phi_4 + \phi_1 + \phi_3
\end{aligned}$$

or $\ddot{u} = -K_4 u$, where $u = \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \phi_4 \end{bmatrix}$ and K_4 is a matrix of coefficients.

(a) How can you see that K_4 is singular? What is the physical meaning of this fact?

(b) Find the eigenvalues and eigenvectors of K_4 and using the spectral factorization, $K_4 = S\Lambda S^{-1}$, diagonalize and solve the system.

(c) Describe the normal modes of the oscillators in terms of eigenvalues and eigenvectors of K_4. What are the largest and smallest frequencies and corresponding eigenvectors? Can you explain the physics of the largest frequency mode?

(d) What is the solution that starts at $u(0) = \begin{bmatrix} 1 & 0 & -1 & 0 \end{bmatrix}^T$ and $\dot{u}(0) = 0$?

Solution

```
K = sp.Matrix([
    [-2, 1, 0, 1],
    [1, -2, 1, 0],
    [0, 1, -2, 1],
    [1, 0, 1, -2]
])
K = -K
print("Rank: {}".format(K.rank()))
```

```
Rank: 3
```

The singularity of the matrix implies that we have a redundant coordinate: the system is described with 4 variables, while 3 independend variables would be enough.

```python
from sympy import Function

phi1, phi2, phi3, phi4 = abc.symbols('phi1, phi2, phi3, phi4', cls=Function)
t = abc.symbols('t')

phi1 = phi1(t)
phi2 = phi2(t)
phi3 = phi3(t)
phi4 = phi4(t)

phi1d, phi2d, phi3d, phi4d = (sp.diff(i, t) for i in (phi1, phi2, phi3, phi4))
phi1dd, phi2dd, phi3dd, phi4dd = (sp.diff(i, t) for i in (phi1d, phi2d, phi3d, phi4d))

u = sp.Matrix([phi1, phi2, phi3, phi4])
udd = sp.Matrix([phi1dd, phi2dd, phi3dd, phi4dd])


eig = K.eigenvects()
l1 = eig[0][0]
l2 = eig[1][0]
l3 = eig[2][0]
h1 = eig[0][2][0]
h21 = eig[1][2][0]
h22 = eig[1][2][1]
h3 = eig[2][2][0]
```

$$\lambda_1 = 4, \quad h_1 = \begin{bmatrix} -1 \\ 1 \\ -1 \\ 1 \end{bmatrix};$$
$$\lambda_2 = 2, \quad h_{2_1} = \begin{bmatrix} -1 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \quad h_{2_2} = \begin{bmatrix} 0 \\ -1 \\ 0 \\ 1 \end{bmatrix};$$
$$\lambda_3 = 0, \quad h_3 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}.$$

```python
Ss, L = K.diagonalize()
L
```

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 4 \end{bmatrix}$$

We found the Λ component of the diagonalized matrix K, now we can proceed with solving:

$$\ddot{u} = S\Lambda S^{-1}u,$$
$$\ddot{v} = \Lambda v, \quad v = S^{-1}u;$$
$$\ddot{v}_i = c_1 \cos\left(\sqrt{\lambda_i}t\right) + c_2 \sin\left(\sqrt{\lambda_i}t\right)$$

Note

All the eigenvalues are ≥ 0. That means, our solutions are representable as a sum of $\sin$ and $\cos$ functions.

Finally, our general solution looks like this:

$$v_1(t) = c_1 \cos(2t) + c_2 \sin(2t);$$
$$v_2(t) = c_3 \cos\left(\sqrt{2}t\right) + c_4 \sin\left(\sqrt{2}t\right);$$
$$v_3(t) = c_5 \cos\left(\sqrt{2}t\right) + c_6 \sin\left(\sqrt{2}t\right);$$
$$v_4(t) = (c_7 + c_8 t)$$

and after returning into original basis ($u = Sv$):

$$
\begin{aligned}
u = (c_7 + c_8 t)h_3 \\
+ \\
(c_3 \cos(\sqrt{2}t) + c_4 \sin(\sqrt{2}t))h_{2_1} \\
+ \\
(c_5 \cos(\sqrt{2}t) + c_6 \sin(\sqrt{2}t))h_{2_2} \\
+ \\
(c_1 \cos(2t) + c_2 \sin(2t))h_1
\end{aligned}
$$

Let's confirm this with *sympy*:

```
from sympy import *

v1, v2, v3, v4 = abc.symbols('v1, v2, v3, v4', cls=Function)
c1, c2, c3, c4, c5, c6, c7, c8 = abc.symbols('c1, c2, c3, c4, c5, c6, c7, c8')
t = abc.symbols('t')
v1 = v1(t)
v2 = v2(t)
v3 = v3(t)
v4 = v4(t)

v1 = c7 + c8*t
v2 = c3*cos(sqrt(2)*t) + c4*sin(sqrt(2)*t)
v3 = c5*cos(sqrt(2)*t) + c6*sin(sqrt(2)*t)
v4 = c1*cos(2*t) + c2*sin(2*t)

v = sp.Matrix([v1, v2, v3, v4])

Ss*v
```

$$
\begin{bmatrix}
-c_1 \cos(2t) - c_2 \sin(2t) - c_3 \cos\left(\sqrt{2}t\right) - c_4 \sin\left(\sqrt{2}t\right) + c_7 + c_8 t \\
c_1 \cos(2t) + c_2 \sin(2t) - c_5 \cos\left(\sqrt{2}t\right) - c_6 \sin\left(\sqrt{2}t\right) + c_7 + c_8 t \\
-c_1 \cos(2t) - c_2 \sin(2t) + c_3 \cos\left(\sqrt{2}t\right) + c_4 \sin\left(\sqrt{2}t\right) + c_7 + c_8 t \\
c_1 \cos(2t) + c_2 \sin(2t) + c_5 \cos\left(\sqrt{2}t\right) + c_6 \sin\left(\sqrt{2}t\right) + c_7 + c_8 t
\end{bmatrix}
$$

$$
u = \begin{bmatrix} \phi_1(t) \\ \phi_2(t) \\ \phi_3(t) \\ \phi_4(t) \end{bmatrix} =
$$

$$
= \begin{bmatrix}
-c_1 cos(2t) & - & c_2 sin(2t) & - & c_3 cos(\sqrt{2}t) & - & c_4 sin(\sqrt{2}t) & + & c_7 & + & c_8 t \\
c1 cos(2t) & + & c_2 sin(2t) & - & c_5 cos(\sqrt{2}t) & - & c_6 sin(\sqrt{2}t) & + & c_7 & + & c_8 t \\
-c_1 cos(2t) & - & c_2 sin(2t) & + & c_3 cos(\sqrt{2}t) & + & c_4 sin(\sqrt{2}t) & + & c_7 & + & c_8 t \\
c1 cos(2t) & + & c_2 sin(2t) & + & c_5 cos(\sqrt{2}t) & + & c_6 sin(\sqrt{2}t) & + & c_7 & + & c_8 t
\end{bmatrix}
$$

We can analyze the system by looking at the eigenvalues and eigenvectors. The eigenvalue $\lambda_1 = 0$ and the corresponding $h_1 = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}^T$ represent the rotational motion of the system as a whole. It's frequency is 0, and the time function $f_1(t) = c_7 + c_8 t$ accounts for the initial position c_7 and the initial velocity c_8.

The rest of the eigen-entities correspond to a certain type of oscillation, see the related picture.

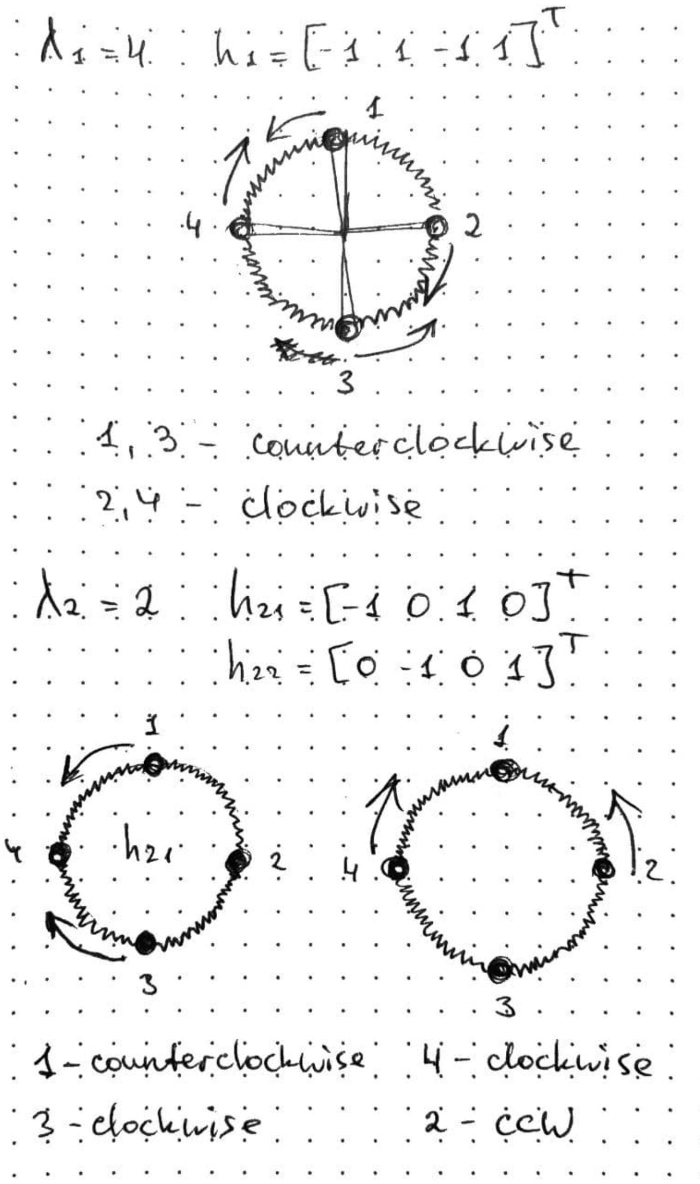


Fig. 2 Oscillation types

The eigenvalues correspond to the amplitude of the oscillations. The oscillations corresponding to the $\lambda_1 = 4$ will be the most frequent, because the eigenvalue is effectively the squared frequency in the harmonic time function.

Now let us analyze the initial conditions:

$u(0) = [1 \quad 0 \quad -1 \quad 0]^T$ and $\dot{u}(0) = 0$.

Let us start top-down.

- Firstly, we substitude t with 0 at the general solution and solve for $u(0)$:

```
u0 = Matrix([1, 0, -1, 0])
u0d = 0
c_vec = Matrix([c1, c2, c3, c4, c5, c6, c7, c8])

S0 = Ss*v.subs(t, 0)
S0
```

$$\begin{bmatrix} -c_1 - c_3 + c_7 \\ c_1 - c_5 + c_7 \\ -c_1 + c_3 + c_7 \\ c_1 + c_5 + c_7 \end{bmatrix}$$

```
sol = solve(S0 - u0, c_vec)
sol
```

```
{c1: 0, c3: -1, c5: 0, c7: 0}
```

We get the following constants:

```
c_vec_s = copy.deepcopy(c_vec)
c_vec_s = c_vec_s.subs(sol)
c_vec_s
```

$$\begin{bmatrix} 0 \\ c_2 \\ -1 \\ c_4 \\ 0 \\ c_6 \\ 0 \\ c_8 \end{bmatrix}$$

And we receive the following semi-solution:

```
u_s = Ss*v
u_s.subs(sol)
```

$$\begin{bmatrix} -c_2 \sin(2t) - c_4 \sin\left(\sqrt{2}t\right) + c_8 t + \cos\left(\sqrt{2}t\right) \\ c_2 \sin(2t) - c_6 \sin\left(\sqrt{2}t\right) + c_8 t \\ -c_2 \sin(2t) + c_4 \sin\left(\sqrt{2}t\right) + c_8 t - \cos\left(\sqrt{2}t\right) \\ c_2 \sin(2t) + c_6 \sin\left(\sqrt{2}t\right) + c_8 t \end{bmatrix}$$

- Now, to account for $\dot{u}(0) = 0$, we need to differentiate our solution u and apply the condition:

```
u_s_1 = diff(u_s, t).subs(t, 0)
u_s_1
```

$$\begin{bmatrix} -2c_2 - \sqrt{2}c_4 + c_8 \\ 2c_2 - \sqrt{2}c_6 + c_8 \\ -2c_2 + \sqrt{2}c_4 + c_8 \\ 2c_2 + \sqrt{2}c_6 + c_8 \end{bmatrix}$$

```
sol2 = solve(u_s_1, c2, c4, c6, c8)
sol2
```

```
{c2: 0, c4: 0, c6: 0, c8: 0}
```

```
c_vec_s2 = copy.deepcopy(c_vec_s)
c_vec_s2 = c_vec_s2.subs(sol2)
c_vec_s2.simplify()
c_vec_s2
```

$$\begin{bmatrix} 0 \\ 0 \\ -1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

We get $c_3 = -1$ and all other constants equal to 0.

```
uuu = u_s.subs(sol).subs(sol2)
uuu
#print(bmatrix(np.matrix(uuu)))
```

$$\begin{bmatrix} \cos\left(\sqrt{2}t\right) \\ 0 \\ -\cos\left(\sqrt{2}t\right) \\ 0 \end{bmatrix}$$

This is our particular solution for given initial conditions:

$$u_p(t) = \begin{bmatrix} \cos(\sqrt{2}t) \\ 0 \\ -\cos(\sqrt{2}t) \\ 0 \end{bmatrix}$$

But, we may come to this solution by some reasoning. If we look at the initial condition $u(0)$, we may notice this is one of the eigenvectors taken with a factor -1. So, the resulting motion will be a normal mode of the corresponding oscillation. That we can see in our exact solution, with $c_3 = -1$.
