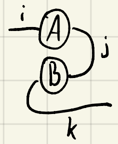
Group 2:
- Armin Ettenhofer
- Atul Agarwal
- Johannes Spies

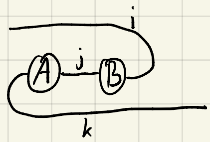
10/10

3.1.

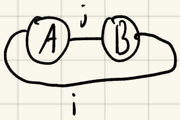
a) AB^T: $(AB^T)_{ik} = \sum_j a_{ij} b_{kj} \Rightarrow$



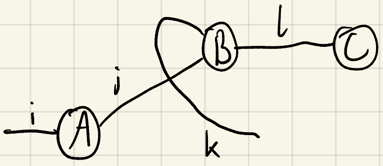
$B^T A^T$: $(B^T A^T)_{ik} = \sum_j b_{ji} a_{kj} \Rightarrow$



$tr[AB] = \sum_i (AB)_{ii} \Rightarrow$



b)



c) Contraction result:

$$\underbrace{C_{i_1 \cdots i_{(dA-c)} j_1 \cdots j_{(dB-c)}}}_{\Rightarrow (d_A - c) + (d_B - c) \text{ dimensions}} = \underbrace{\sum_{k_1 \cdots k_c} a_{i_1 \cdots k_1 \cdots} \cdot b_{k_1 \cdots j_1 \cdots}}_{c \text{ nested sums}}$$

$\Rightarrow n^{d_A + d_B - 2c}$ elements

$\Downarrow$

$O(n^{d_A + d_B - 2c})$

c nested sums $\Downarrow$ $O(n^c)$

$$O(n^{d_A + d_B - 2c} \cdot n^c) = \underline{\underline{O(n^{d_A + d_B - c})}}$$

5/5

3.2

a) $(-\beta X)^\dagger = \begin{pmatrix} 0 & -\beta \\ -\beta & 0 \end{pmatrix} = -\beta X \Rightarrow$ normal

<u>spectral decomposition of $-\beta X$:</u>

$$\det(\lambda I - (-\beta X)) = \det\begin{pmatrix} \lambda & \beta \\ \beta & \lambda \end{pmatrix} = \lambda^2 - \beta^2 = (\lambda-\beta)\cdot(\lambda+\beta)$$

$\Rightarrow \lambda_1 = \beta \quad \lambda_2 = -\beta$

$\lambda = \beta$:

$$\beta I - (-\beta X) = \begin{pmatrix} \beta & \beta \\ \beta & \beta \end{pmatrix} \Rightarrow \beta x + \beta y = 0 \Rightarrow v_1 = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

$\lambda = -\beta$:

$$-\beta I - (-\beta X) = \begin{pmatrix} -\beta & \beta \\ \beta & -\beta \end{pmatrix} \Rightarrow -\beta x + \beta y = 0 \Rightarrow v_2 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$$U = \begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ -1/\sqrt{2} & 1/\sqrt{2} \end{pmatrix} = \frac{1}{\sqrt{2}} \cdot \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}$$ (with normalised eigenvectors)

$$U^{-1} = U^\dagger = \frac{1}{\sqrt{2}} \cdot \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix} \qquad \Rightarrow -\beta X = \frac{1}{\sqrt{2}} \cdot \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix} \cdot \begin{pmatrix} \beta & 0 \\ 0 & -\beta \end{pmatrix} \cdot \frac{1}{\sqrt{2}} \cdot \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}$$

(because the eigenvectors are normalised)

$$\Rightarrow e^{-\beta X} = U \begin{pmatrix} e^\beta & 0 \\ 0 & e^{-\beta} \end{pmatrix} U^\dagger = \frac{1}{\sqrt{2}} \cdot \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix} \cdot \begin{pmatrix} e^\beta & 0 \\ 0 & e^{-\beta} \end{pmatrix} \cdot \frac{1}{\sqrt{2}} \cdot \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}$$

$$= \frac{1}{2} \cdot \begin{pmatrix} e^\beta & e^{-\beta} \\ -e^\beta & e^{-\beta} \end{pmatrix} \cdot \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix} = \frac{1}{2} \cdot \begin{pmatrix} e^\beta + e^{-\beta} & -e^\beta + e^{-\beta} \\ -e^\beta + e^{-\beta} & e^\beta + e^{-\beta} \end{pmatrix} \overset{\text{(Hint)}}{=} \begin{pmatrix} \cosh(\beta) & -\sinh(\beta) \\ -\sinh(\beta) & \cosh(\beta) \end{pmatrix}$$

c)

$$f(A) = \frac{1}{2\pi i} \oint_\gamma f(z)(zI - A)^{-1} \mathrm{d}z,$$

$z = re^{2\pi i t} \qquad \frac{dz}{dt} = 2\pi i \, re^{2\pi i t}$

$$f(A) = \int_0^1 f(re^{2\pi i t}) \frac{1}{(re^{2\pi i t})^2 - 1} \cdot (re^{2\pi i t} I - A) \cdot re^{2\pi i t} \, dt$$

$\Rightarrow$ insert into Python

exercise3

May 18, 2022

[1]:
```
import numpy as np
from scipy.linalg import expm
import scipy.integrate as integrate
from numpy import exp
```

[2]:
```
#####3.2b)
# Pauli-X matrix
X = np.array([[0., 1.], [1., 0.]])

# evaluate matrix exponential numerically
beta = 0.4
numeric = expm(-beta * X)

# Solution using spectral decomposition
with_decomposition = np.array([[np.cosh(beta), -np.sinh(beta)], [-np.
 ↪sinh(beta), np.cosh(beta)]])

print(numeric)
print("Matches with expected: " + str(np.allclose(numeric, with_decomposition)))
```

```
[[ 1.08107237 -0.41075233]
 [-0.41075233  1.08107237]]
Matches with expected: True
```

[3]:
```
#####3.2c)

# Code for single value
beta = 0.4

def f(z):
    return exp(-beta * z)

r = 2.  # circle radius

# Adapted code for a matrix
def z(t):
    # For better readability
```

```
    return r * exp(2j * np.pi * t)

integrand_mat = lambda t: f(z(t)) / (z(t) ** 2 - 1) * z(t) * (z(t) * np.eye(2)␣
 ↪+ X)

integrate_matrix = (integrate.quad_vec(lambda t: np.real(integrand_mat(t)), 0,␣
 ↪1)[0] +
 1j * integrate.quad_vec(lambda t: np.imag(integrand_mat(t)), 0, 1)[0])
print(integrate_matrix)
print("Matches with expected: " + str(np.allclose(numeric, integrate_matrix)))
```

```
[[ 1.08107237-1.38777878e-17j -0.41075233+0.00000000e+00j]
 [-0.41075233+0.00000000e+00j  1.08107237-1.38777878e-17j]]
Matches with expected: True
```

5/5