Sheet 10 done

Elias Myllymäki

May 2025

Exercise 3

a)

To calculate the orthogonal projection matrix Q, we can just copy the mallivastaukset of question 2 and we define $Q = V(V^TV)^{-1}V^T$ and doing the calculation with sympy:

```
from sympy import *

def s():
 # Do the stuff here maybe????

 V = Matrix([[1,1],[2,1],[1,2],[0,1]])

 # Now we do the shit

 Q = V @ (V.transpose() @ V).inv() @ V.transpose()
 print("Q: "+str(Q))
 print("As latex: "+str(latex(Q)))

 return

if __name__=="__main__":
 s()
 exit(0)
```

So the answer is:

$$Q = \begin{bmatrix} \frac{3}{17} & \frac{5}{17} & \frac{4}{17} & \frac{1}{17} \\ \frac{5}{17} & \frac{14}{17} & \frac{1}{17} & -\frac{4}{17} \\ \frac{4}{17} & \frac{1}{17} & \frac{11}{17} & \frac{7}{17} \\ \frac{1}{17} & -\frac{4}{17} & \frac{7}{17} & \frac{6}{17} \end{bmatrix}$$

b)

To solve part b we can just join the matrices V and W to form a 4x4 matrix then take the inverse matrix of this and then finally multiply it with V to get the answer. Here is a quick python script:

```
from sympy import *

def s():
    V = Matrix([[1,1],[2,1],[1,2],[0,1]])
    W = Matrix([[2,1],[1,1],[1,1],[0,1]])

    # Concatenate to form a full-rank basis for R^4
    M = V.row_join(W)

    # Inverse of M
    M_inv = M.inv()

    # Take first 2 rows of M_inv to extract V-components
    P = V @ M_inv[:2, :]

    print("P: "+str(P))
    print("As latex: "+latex(P))

if __name__=="__main__":
    s()
    exit(0)
```

And the answer for P is:

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

c)

Let's check the solutions...

```
from sympy import *




def partb():
    V = Matrix([[1,1],[2,1],[1,2],[0,1]])
    W = Matrix([[2,1],[1,1],[1,1],[0,1]])
```

```
    # Concatenate to form a full-rank basis for R^4
    M = V.row_join(W)

    # Inverse of M
    M_inv = M.inv()

    # Take first 2 rows of M_inv to extract V-components
    P = V @ M_inv[:2, :]

    print("P: "+str(P))
    print("As latex: "+latex(P))

    return P


def s():
 # Do the stuff here maybe????
 P = partb()
 V = Matrix([[1,1],[2,1],[1,2],[0,1]])
 W = Matrix([[2,1],[1,1],[1,1],[0,1]])
 I = Matrix([[1,0,0,0],[0,1,0,0],[0,0,1,0],[0,0,0,1]])

 Q = V @ (V.transpose() @ V).inv() @ V.transpose()
 print("Q: "+str(Q))
 print("As latex: "+str(latex(Q)))

 print("Checking...")

 assert Q @ Q == Q # Should be true...
 x = Q.transpose() @ (I - Q)
 assert x == zeros(4,4) # Should be the zero matrix...

 assert P @ V == V
 # print(P @ W)
 assert P @ W == zeros(4,2)


 return

if __name__=="__main__":
 s()
 exit(0)
```

and because it passes all of the assertions so our solution should be good.

Exercise 4

a)

For P to be a projection matrix, it must satisfy $P^T = P$ and $P^2 = P$:

$P^T = (\frac{aa^T}{||a||_2^2} + \frac{bb^T}{||b||_2^2})^T$ and this is via the properties of the transpose equal to $(\frac{aa^T}{||a||_2^2})^T + (\frac{bb^T}{||b||_2^2})^T = \frac{aa^T}{||a||_2^2} + \frac{bb^T}{||b||_2^2} = P$

$P^2 = (\frac{aa^T}{||a||_2^2} + \frac{bb^T}{||b||_2^2})^2$, now because projection is idempotent, $(\frac{aa^T}{||a||_2^2})^2 = \frac{aa^T}{||a||_2^2}$ and $(\frac{bb^T}{||b||_2^2})^2 = \frac{bb^T}{||b||_2^2}$. The cross terms cancel out, because $(\frac{aa^T}{||a||_2^2})(\frac{bb^T}{||b||_2^2}) = \frac{a^Tb}{||a||_2^2||b||_2^2}ab^T = 0$

Therefore P is a projection matrix. $I - P$ is also a projection matrix.

b)

We know because of a) that P is a projection matrix. We also know that $P^T = P$. Therefore P is an orthogonal projection

c)

Just guess such a vector?

Let's make a script:

```
from sympy import *
import random

MAX_INTEGER_MAGNITUDE = 10 # Use low integers for now

def rrat():
 return random.randrange(-MAX_INTEGER_MAGNITUDE, MAX_INTEGER_MAGNITUDE)

def euclidean_norm_squared(v):
 return v.dot(v) # Already gives the squared value

def s():
 # Solve the problem...
 a = Matrix([[1],[1],[1]])
 b = Matrix([[1],[-2],[1]])
 I = Matrix([[1,0,0],[0,1,0],[0,0,1]]) # 3x3 identity
 c = Matrix([[rrat()], [rrat()], [rrat()]])
 # Calculate P
 P = (a * a.T) / (euclidean_norm_squared(a)) + (b * b.T) / (euclidean_norm_squared(b))
 while True: # Main bruteforce loop
  lhs = I - P
  c = Matrix([[rrat()], [rrat()], [rrat()]])
  rhs = (c * c.T) / (euclidean_norm_squared(c))
  if simplify(lhs - rhs) == Matrix.zeros(3, 3):
   break
 print("Solution: "+str(c))
```

```
 return

if __name__=="__main__":
 s()
 exit(0)
```

and this gives an answer of

```
Solution: Matrix([[-4], [0], [4]])
```

we observe that this is just the of cross product of a and b multiplied by some vector.