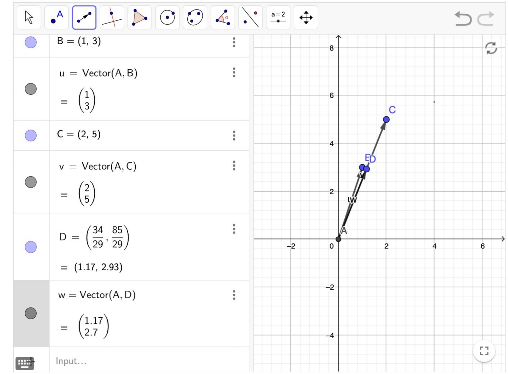
1) $b = \begin{bmatrix} 2 \\ 5 \end{bmatrix}$

$$\|b\| = \sqrt{2^2 + 5^2} = \sqrt{29}$$

$$\hat{b} = \frac{1}{\sqrt{29}} \begin{bmatrix} 2 \\ 5 \end{bmatrix} = \begin{bmatrix} 2/\sqrt{29} \\ 5/\sqrt{29} \end{bmatrix}$$



$$a = \begin{bmatrix} 1 \\ 3 \end{bmatrix}$$

$$\alpha = \langle a, \hat{b} \rangle = \begin{bmatrix} 1 \\ 3 \end{bmatrix} \begin{bmatrix} 2/\sqrt{29} \\ 5/\sqrt{29} \end{bmatrix} = \frac{17}{\sqrt{29}}$$

$$b_p = \alpha \hat{b} = \frac{17}{\sqrt{29}} \begin{bmatrix} 2/\sqrt{29} \\ 5/\sqrt{29} \end{bmatrix} = \begin{bmatrix} 34/29 \\ 85/29 \end{bmatrix}$$

2) $\text{proj}_a b = \dfrac{\langle b, a \rangle}{\langle a, a \rangle} a$

$\langle b, a \rangle = (2 \cdot 1) + (5 \cdot 3) + (1 \cdot 3) + (1 \cdot 2) = 22$

$\langle a, a \rangle = 1^2 + 3^2 + 3^2 + 2^2 = 23$

$$\text{proj}_a b = \frac{22}{23} \begin{bmatrix} 1 \\ 3 \\ 3 \\ 2 \end{bmatrix} = \begin{bmatrix} 22/33 \\ 66/23 \\ 66/23 \\ 44/23 \end{bmatrix}$$

3)

[1]:
```python
import numpy as np

v1 = np.array([0.0283, 0.7200, -0.6933])
v2 = np.array([0.7988, -0.4332, -0.4172])

# Calculate dot product
dot_product = np.dot(v1, v2)

# Calculate magnitudes
magnitude_v1 = np.linalg.norm(v1)
magnitude_v2 = np.linalg.norm(v2)

# Check for orthogonality (dot product is approximately 0)
is_orthogonal = np.isclose(dot_product, 0, atol=1e-9)

# Check for orthonormality (dot product is approximately 0 and magnitudes are approximately 1)
is_orthonormal = is_orthogonal and np.isclose(magnitude_v1, 1, atol=1e-9) and np.isclose(magnitude_v2, 1, atol=1e-9)

(is_orthogonal, is_orthonormal)
```

[1]:
```
(False, False)
```

4)

[4]:
```python
# Function to calculate error between y and its approximation using v1 and v2
def calculate_error(y):
    y_approx = np.dot(y, v1)*v1 + np.dot(y, v2)*v2
    return np.linalg.norm(y - y_approx)

# Generating 5 random points and calculating their errors
np.random.seed(42)  # for reproducibility
for i in range(5):
    random_point = np.random.rand(3)
    error_random_point = calculate_error(random_point)
    print(f"Error for random point {i+1}:", error_random_point)
```

```
Error for random point 1: 1.1704485660308621
Error for random point 2: 0.5359095405459156
Error for random point 3: 0.8575964736752298
Error for random point 4: 1.0064004526249353
Error for random point 5: 0.7220783392534825
```

[2]:
```python
import numpy as np

# Given vectors
v1 = np.array([0.0283, 0.7200, -0.6933])
v2 = np.array([0.7988, -0.4332, -0.4172])
v3 = np.array([0.6008, 0.5421, 0.5875])
y = np.array([1, 1, 1])

# Forming the matrix A and vector B to solve the system Ax = B
A = np.column_stack((v1, v2, v3))
B = y

# Solving for [a, b, c]
coefficients = np.linalg.solve(A, B)
print("Coefficients [a, b, c]:", coefficients)
```

```
Coefficients [a, b, c]: [ 0.0550031  -0.05156644  1.73041725]
```

[3]:
```python
# Given new y
y_new = np.array([1.42805, 0.82890, -0.52308])

# Since v1 and v2 form an orthonormal basis for their span,
# the closest approximation to y using v1 and v2 is given by the projection of y onto the span of v1 and v2
y_approx = np.dot(y_new, v1)*v1 + np.dot(y_new, v2)*v2

# Calculating the error as the Euclidean distance between y and its approximation
error = np.linalg.norm(y_new - y_approx)
print("Approximation:", y_approx)
print("Error:", error)
```

```
Approximation: [ 0.82699723  0.28676248 -1.11036027]
Error: 1.0000378232478215
```

5) $A = \begin{bmatrix} 8 & 3 \\ -4 & 1 \end{bmatrix}$

$\det \begin{bmatrix} 8-\lambda & 3 \\ -4 & 1-\lambda \end{bmatrix} = 0$

$(8-\lambda)(1-\lambda) - (3 \cdot -4) = 0$

$\lambda^2 - 9\lambda + 20 = 0$

$\lambda = \frac{9 \pm \sqrt{1}}{2}$

$\lambda = 5$

$\lambda = 4$

$\lambda = 5$:

$(A - 5I)v = 0$

$\begin{bmatrix} 8-5 & 3 \\ -4 & 1-5 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = 0$

$\begin{bmatrix} 3 & 3 \\ -4 & -4 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = 0$

$3x + 3y = 0$

$y = -x$

$v_1 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$

$\lambda = 4$!

$(A - 4I)v = 0$

$$\begin{bmatrix} 8-4 & 3 \\ -4 & 1-4 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = 0$$

$$\begin{bmatrix} 4 & 3 \\ -4 & -3 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = 0$$

$4x + 3y = 0$

$y = -\frac{4}{3}x$

$$v_2 = \begin{bmatrix} 3 \\ -4 \end{bmatrix}$$

$$V = \begin{bmatrix} 1 & 3 \\ -1 & -4 \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} 5 & 0 \\ 0 & 4 \end{bmatrix}$$

$$V^{-1} = \frac{1}{-3-12} \begin{bmatrix} -4 & -3 \\ 1 & 1 \end{bmatrix}$$

$$V^{-1} = -\frac{1}{15} \begin{bmatrix} -4 & -3 \\ 1 & 1 \end{bmatrix}$$

6)

[6]:

```python
import pandas as pd
import numpy as np
import os

# Step 1: Load the data into Python
A = pd.read_csv('largeDat.csv').to_numpy()

# Step 2: Compute Q and its Eigendecomposition
Q = A.T @ A
eigenvalues, eigenvectors = np.linalg.eig(Q)

# Step 3: Identify the Minimum Orthonormal Basis
indices = np.argsort(eigenvalues)[-2:]  # Get indices of the two largest eigenvalues
P = eigenvectors[:, indices]
np.savetxt("basis.csv", P, delimiter=",")

# Step 4: Represent A as Coordinates of the Minimum Basis
U = A @ P
np.savetxt("compressed.csv", U, delimiter=",")

# Step 5: Verify the Compression (optional)
A_approx = U @ P.T
reconstruction_error = np.linalg.norm(A - A_approx)
print("Reconstruction Error:", reconstruction_error)

# Step 6: Compare File Sizes
original_size = os.path.getsize('largeDat.csv')
compressed_size = os.path.getsize('compressed.csv')

print("Original Size:", original_size, "bytes")
print("Compressed Size:", compressed_size, "bytes")
```

```
Reconstruction Error: 0.002583154658073094
Original Size: 75021 bytes
Compressed Size: 50983 bytes
```