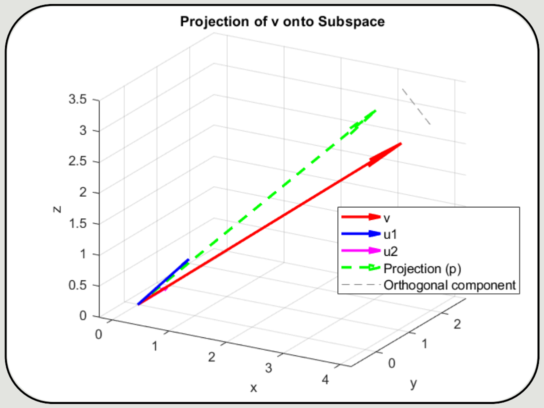
Ku3a Tuts Presents Orthogonal Projection

Introduction

Orthogonal projection maps a vector onto a subspace along a perpendicular direction, decomposing it into two components: the projection within the subspace (closest approximation) and the orthogonal component (perpendicular residual).

Visualization



The projection p is the "shadow" of vector v on the subspace W, while the difference vector (v - p) is perpendicular to W.
v: red vector representing the original vector.
u1: blue vector spanning the subspace W.
u2: pink vector orthogonal to u1 spanning subspace W.
p: green vector is the orthogonal projection of v onto W.
Orthogonal Component: The dashed black line represents v − p, which is orthogonal to W.

Equations

Orthogonal Projection onto a Vector

$$\frac{v.u}{u.u}u$$

Orthogonal Projection onto a Subspace

$$A(A^TA)^{-1}A^T$$

Examples

Orthogonal Projection onto a Vector

Find the projection of v=[3,4] onto u=[1,1].

1. Formula

$$p = \frac{v\cdot u}{u\cdot u}u$$

2. Dot Products

$$v\cdot u = (3)(1)+(4)(1)=7$$
$$u\cdot u = (1)(1)+(1)(1)=2$$

3. Scalar Coefficient

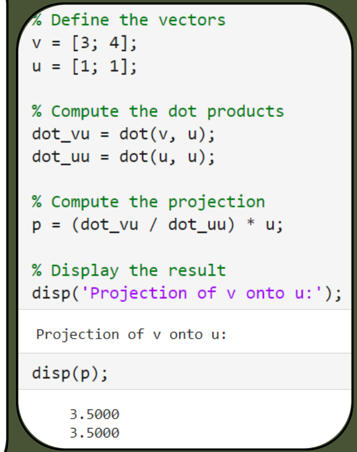
$$\frac{v\cdot u}{u\cdot u} = \frac{7}{2}$$

4. Projection Vector

$$p = \frac{7}{2}\cdot[1,1] = \left[\frac{7}{2},\frac{7}{2}\right]$$

5. Final Answer

$$p = \left[\frac{7}{2},\frac{7}{2}\right]$$



```
% Define the vectors
v = [3; 4];
u = [1; 1];

% Compute the dot products
dot_vu = dot(v, u);
dot_uu = dot(u, u);

% Compute the projection
p = (dot_vu / dot_uu) * u;

% Display the result
disp('Projection of v onto u:');
```

```
Projection of v onto u:
```

```
disp(p);
```

```
    3.5000
    3.5000
```

Orthogonal Projection onto a Subspace

Find the projection of v=[4,2,3] onto the subspace spanned by u1=[1,0,1] and u2=[0,1,0].

1. Matrix A

$$A = \begin{bmatrix}1&0\\0&1\\1&0\end{bmatrix}$$

2. Compute A^TA

$$A^TA = \begin{bmatrix}1&0&1\\0&1&0\end{bmatrix}\begin{bmatrix}1&0\\0&1\\1&0\end{bmatrix} = \begin{bmatrix}2&0\\0&1\end{bmatrix}$$

3. Inverse of A^TA

$$(A^TA)^{-1} = \begin{bmatrix}\frac{1}{2}&0\\0&1\end{bmatrix}$$

. Compute A^Tv

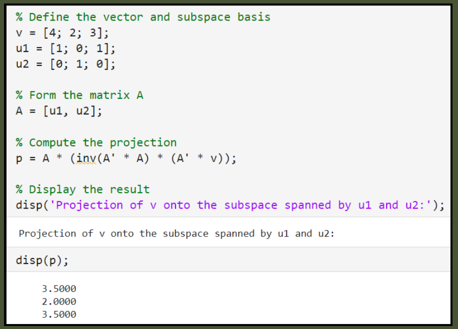
$$A^Tv = \begin{bmatrix}1&0&1\\0&1&0\end{bmatrix}\begin{bmatrix}4\\2\\3\end{bmatrix} = \begin{bmatrix}7\\2\end{bmatrix}$$

5. Combine Results

$$p = A\begin{bmatrix}\frac{1}{2}&0\\0&1\end{bmatrix}\begin{bmatrix}7\\2\end{bmatrix} = A\begin{bmatrix}3.5\\2\end{bmatrix} = \begin{bmatrix}1&0\\0&1\\1&0\end{bmatrix}\begin{bmatrix}3.5\\2\end{bmatrix} = \begin{bmatrix}3.5\\2\\3.5\end{bmatrix}$$

6. Final Answer

The projection is $p = \begin{bmatrix}3.5\\2\\3.5\end{bmatrix}$



```
% Define the vector and subspace basis
v = [4; 2; 3];
u1 = [1; 0; 1];
u2 = [0; 1; 0];

% Form the matrix A
A = [u1, u2];

% Compute the projection
p = A * (inv(A' * A) * (A' * v));

% Display the result
disp('Projection of v onto the subspace spanned by u1 and u2:');
```

```
Projection of v onto the subspace spanned by u1 and u2:
```

```
disp(p);
```

```
    3.5000
    2.0000
    3.5000
```

Common Mistakes

- *Misinterpreting Orthogonality:*

Orthogonality implies zero dot product, not necessarily perpendicular angles in visual representations.

- *Matrix Dimensions in Subspace Projection:*

Forgetting that A must have linearly independent columns for (ATA) to be invertible.

- *Projection Doesn't Change Subspace:*

The projection lies entirely within the subspace, but it does not alter the subspace or its basis.

Applications

- *Signal Processing:*

Noise reduction by separating signal components from noise using subspace projection.

- *Machine Learning:*

PCA reduces data dimensions by projecting onto a lower-dimensional subspace.

- *Computer Graphics:*

Rendering shadows and reflections via projection techniques.

- *Robotics:*

Motion control resolves movement into desired and restricted components using orthogonal projection.

- *Economics & Finance:*

Portfolio optimization separates market-driven returns from idiosyncratic risks using projection.

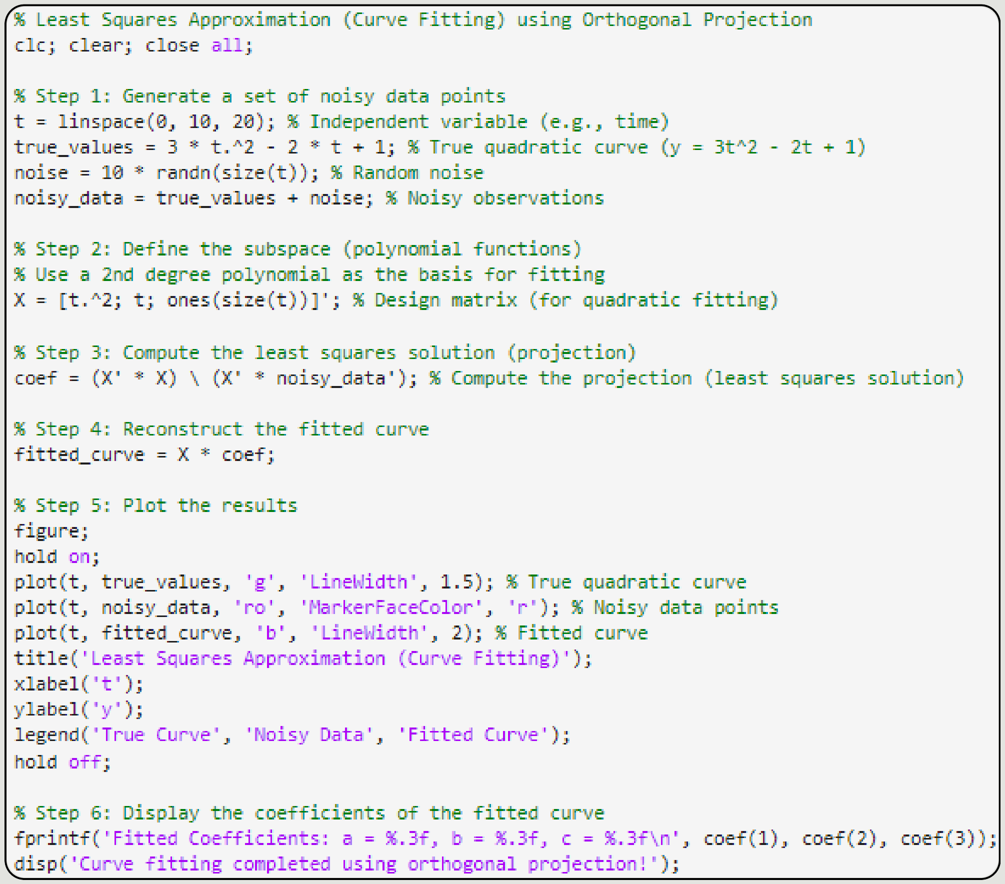
Application Example 1: Least Squares Approximation

Description

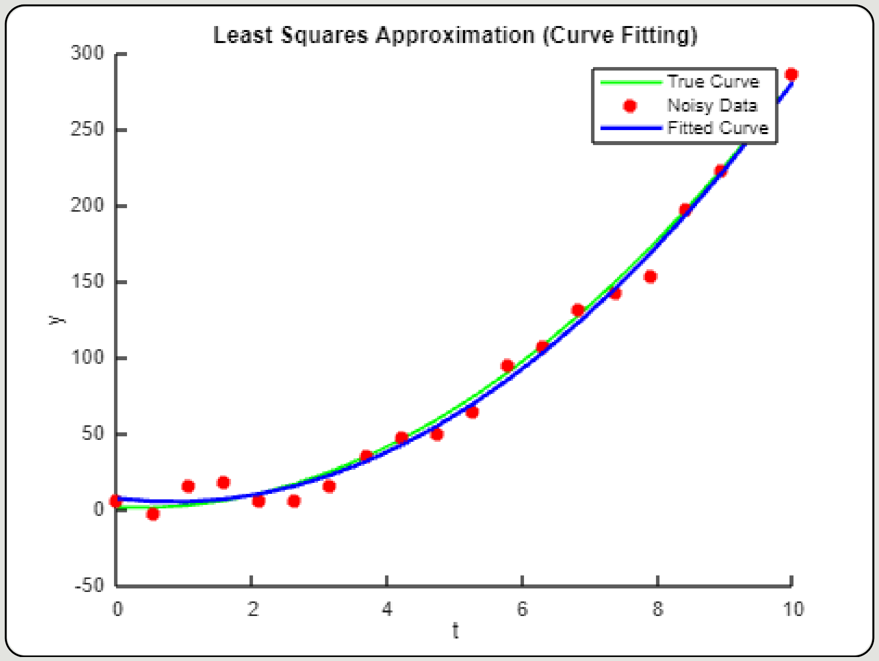
The goal is to find a function (such as a line, parabola) that best fits the data. Least squares approximation is a common technique used to find this "best-fit" function. It aims to minimize the sum of the squared differences (errors) between the observed data points and the values predicted by the function.
In the context of orthogonal projection, the data points can be considered as vectors in a high-dimensional space. The goal is to project these data points onto a subspace that is spanned by a set of basis functions that describe the family of possible curves (e.g., linear, quadratic, etc.).

How Orthogonal Projection is Used

- Data Set: The observed data points are often noisy versions of some underlying curve.
- Subspace Definition: A subspace of possible functions (e.g., polynomial functions) is chosen. For instance, if we are fitting a quadratic curve, the subspace might be spanned by the basis functions {t2,t,1}, where t is the independent variable.
- Projection: The orthogonal projection essentially finds the coefficients of the curve (e.g., the coefficients of a quadratic equation) that minimizes the squared error between the observed data points and the points predicted by the curve.
- Fitted Curve: The result of the projection is the "best-fit" curve, which is the curve in the chosen subspace that has the smallest residual error. The projection provides the coefficients for this curve, which can then be used to reconstruct the fitted model.



```
% Least Squares Approximation (Curve Fitting) using Orthogonal Projection
clc; clear; close all;

% Step 1: Generate a set of noisy data points
t = linspace(0, 10, 20); % Independent variable (e.g., time)
true_values = 3 * t.^2 - 2 * t + 1; % True quadratic curve (y = 3t^2 - 2t + 1)
noise = 10 * randn(size(t)); % Random noise
noisy_data = true_values + noise; % Noisy observations

% Step 2: Define the subspace (polynomial functions)
% Use a 2nd degree polynomial as the basis for fitting
X = [t.^2; t; ones(size(t))]'; % Design matrix (for quadratic fitting)

% Step 3: Compute the least squares solution (projection)
coef = (X' * X) \ (X' * noisy_data'); % Compute the projection (least squares solution)

% Step 4: Reconstruct the fitted curve
fitted_curve = X * coef;

% Step 5: Plot the results
figure;
hold on;
plot(t, true_values, 'g', 'LineWidth', 1.5); % True quadratic curve
plot(t, noisy_data, 'ro', 'MarkerFaceColor', 'r'); % Noisy data points
plot(t, fitted_curve, 'b', 'LineWidth', 2); % Fitted curve
title('Least Squares Approximation (Curve Fitting)');
xlabel('t');
ylabel('y');
legend('True Curve', 'Noisy Data', 'Fitted Curve');
hold off;

% Step 6: Display the coefficients of the fitted curve
fprintf('Fitted Coefficients: a = %.3f, b = %.3f, c = %.3f\n', coef(1), coef(2), coef(3));
disp('Curve fitting completed using orthogonal projection!');
```



Application Example 2: Noise Reduction

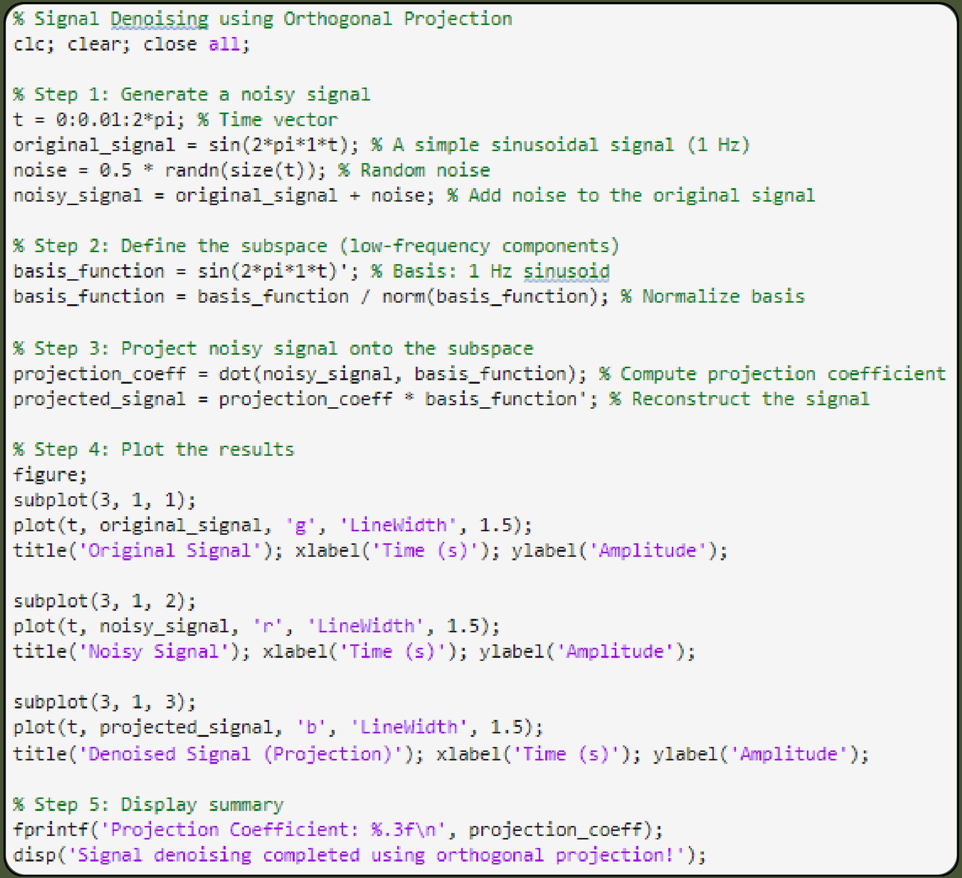
Description

Orthogonal projection is projects a signal onto a subspace. In this application, we take a noisy signal and project it onto the subspace spanned by a set of basis functions (e.g., low-frequency sinusoids). This process isolates the essential components of the signal (the low-frequency components) and removes high-frequency noise.

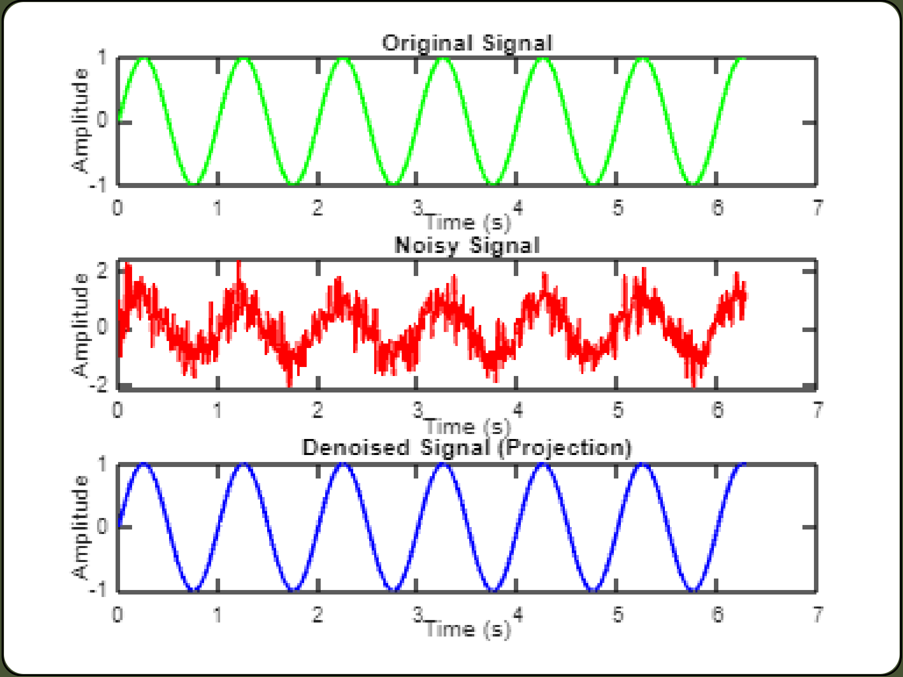
How Orthogonal Projection is Used

- Subspace Definition: The subspace is defined using basis functions that represent low-frequency components of the signal.
- Projection: The noisy signal is orthogonally projected onto this subspace to isolate the meaningful components and remove noise.
- Denoised Signal: The projection result reconstructs the signal, now free of noise, while preserving the original structure.
- Larger-Scale Use Cases
- Audio Denoising: Removing background noise in audio files while retaining speech or music components.
- Communication Systems: Enhancing received signals corrupted by channel noise.
- Medical Applications: Filtering out artifacts from biosignals like ECG to better analyze heart rhythms.

Zarin Sikder & Kuhu Jayaswal



```
% Signal Denoising using Orthogonal Projection
clc; clear; close all;

% Step 1: Generate a noisy signal
t = 0:0.01:2*pi; % Time vector
original_signal = sin(2*pi*1*t); % A simple sinusoidal signal (1 Hz)
noise = 0.5 * randn(size(t)); % Random noise
noisy_signal = original_signal + noise; % Add noise to the original signal

% Step 2: Define the subspace (low-frequency components)
basis_function = sin(2*pi*1*t)'; % Basis: 1 Hz sinusoid
basis_function = basis_function / norm(basis_function); % Normalize basis

% Step 3: Project noisy signal onto the subspace
projection_coeff = dot(noisy_signal, basis_function); % Compute projection coefficient
projected_signal = projection_coeff * basis_function'; % Reconstruct the signal

% Step 4: Plot the results
figure;
subplot(3, 1, 1);
plot(t, original_signal, 'g', 'LineWidth', 1.5);
title('Original Signal'); xlabel('Time (s)'); ylabel('Amplitude');

subplot(3, 1, 2);
plot(t, noisy_signal, 'r', 'LineWidth', 1.5);
title('Noisy Signal'); xlabel('Time (s)'); ylabel('Amplitude');

subplot(3, 1, 3);
plot(t, projected_signal, 'b', 'LineWidth', 1.5);
title('Denoised Signal (Projection)'); xlabel('Time (s)'); ylabel('Amplitude');

% Step 5: Display summary
fprintf('Projection Coefficient: %.3f\n', projection_coeff);
disp('Signal denoising completed using orthogonal projection!');
```


Original Signal
Amplitude
Time (s)
Noisy Signal
Amplitude
Time (s)
Denoised Signal (Projection)
Amplitude
Time (s)