Fabian Figueroa

Professor Stephen Pena

Applied Linear Algebra

March 31, 2025

Project 1

Part 1:

For the creation of a random time series with noise, the provided code by professor was utilised.

```
t = np.arange(0, np.pi + 0.1, 0.1)
clean_signal = np.cos(np.pi * t) * t
noise_level = np.std(clean_signal) / np.random.rand()
x = clean_signal + noise_level * np.random.randn(len(t))
x_interp = interp1d(t, x, kind='linear', bounds_error=False, fill_value="extrap")
```

Utilisation of given code to generate a random time series

Part 2:

For the implementation of the integral calculation, the NumPy trapz function was leveraged. This function utilities the *Trapezoid Rule* to calculate an integral.

Trapezoidal Rule for Numerical Integration $\int_a^b f(x)\,dx \approx \frac{b-a}{2n}\sum_{k=0}^{n}\left(f(x_k)+f(x_{k+1})\right)$

```
# Compute Gram matrix A (G) (6×6) and vector b (6×1)
G = np.zeros((6, 6))
b = np.zeros(6)

for i in range(6):
    for j in range(6):
        # Inner product: A[i,j] = ∫ u_i(t) u_j(t) dt
        integrand = lambda t: basis_funcs(t, i) * basis_funcs(t, j)
        G[i, j] = np.trapz(integrand(t), t)

    # Projection: b[i] = ∫ x_interp(t) u_i(t) dt
    integrand_b = lambda t: x_interp(t) * basis_funcs(t, i)
    b[i] = np.trapz(integrand_b(t), t)

# Solve Ac = b for coefficients (required for projection)
coefficients = np.linalg.solve(G, b)

# Reconstruct approximation
approximation = np.zeros_like(t)
for i in range(6):
    approximation += coefficients[i] * basis_funcs(t, i)
```

NumPy's trapz function being utilised within the matrix calculation

Part 3:

For my basis functions, I chose to use the following:

$$u_n(t) = t^n \cdot e^{nt},\ n \in \{1,2,3,4,5,6\}$$

Basis Functions:

$u_1 = te^t$, $u_2 = t^2e^{2t}$, $u_3 = t^3e^{3t}$, $u_4 = t^4e^{4t}$, $u_5 = t^5e^{5t}$, $u_6 = t^6e^{6t}$

Written portion of basis functions chosen

```python
# Define new basis functions {t·eᵗ, t²·e²ᵗ, ..., t⁶·e⁶ᵗ}
def basis_funcs(t, index):
    n = index + 1  # n = 1, 2, ..., 6
    return (t ** n) * np.exp(n * t)
```

Code snapshot of implementation for chosen basis functions

Part 5:

To formulate the linear system of the chosen basis functions I set up a matrix that receives the inner products of $<u_i,u_j>$, i, $j \in \{1,2,3,4,5,6\}$, for each element of the 6x6 matrix.

$$\begin{bmatrix} \langle u_1, u_1 \rangle & \langle u_1, u_2 \rangle & \langle u_1, u_3 \rangle & \langle u_1, u_4 \rangle & \langle u_1, u_5 \rangle & \langle u_1, u_6 \rangle \\ \langle u_2, u_1 \rangle & \langle u_2, u_2 \rangle & \langle u_2, u_3 \rangle & \langle u_2, u_4 \rangle & \langle u_2, u_5 \rangle & \langle u_2, u_6 \rangle \\ \langle u_3, u_1 \rangle & \langle u_3, u_2 \rangle & \langle u_3, u_3 \rangle & \langle u_3, u_4 \rangle & \langle u_3, u_5 \rangle & \langle u_3, u_6 \rangle \\ \langle u_4, u_1 \rangle & \langle u_4, u_2 \rangle & \langle u_4, u_3 \rangle & \langle u_4, u_4 \rangle & \langle u_4, u_5 \rangle & \langle u_4, u_6 \rangle \\ \langle u_5, u_1 \rangle & \langle u_5, u_2 \rangle & \langle u_5, u_3 \rangle & \langle u_5, u_4 \rangle & \langle u_5, u_5 \rangle & \langle u_5, u_6 \rangle \\ \langle u_6, u_1 \rangle & \langle u_6, u_2 \rangle & \langle u_6, u_3 \rangle & \langle u_6, u_4 \rangle & \langle u_6, u_5 \rangle & \langle u_6, u_6 \rangle \end{bmatrix}$$

Latex implementation of written matrix for clarity

Example computation of an inner product:

Choosing u_2 and u_3 to compute inner product, we have:

$\langle \vec{u}_2, \vec{u}_3 \rangle = \int_a^b u_2(t)\, u_3(t)\, dt$, where u_2 and u_3 are two integrable functions on $[a, b]$ and $\int_a^b u_2(t)\, u_3(t)\, dt$ is an inner product equating to $\cancel{\int_a^b t \cdot t^2 dt = \int_a^b t^3 dt}$ $\int_a^b t^2 e^{2t} \cdot t^3 e^{3t}$

Formulating a linear system:

To formulate the linear system of the chosen basis functions you will set up a matrix that takes the inner product of $\langle u_i, u_j \rangle$ for each element:

$$\begin{bmatrix}
\langle u_1, u_1 \rangle & \langle u_1, u_2 \rangle & \langle u_1, u_3 \rangle & \langle u_1, u_4 \rangle & \langle u_1, u_5 \rangle & \langle u_1, u_6 \rangle \\
\langle u_2, u_1 \rangle & \langle u_2, u_2 \rangle & \langle u_2, u_3 \rangle & \langle u_2, u_4 \rangle & \langle u_2, u_5 \rangle & \langle u_2, u_6 \rangle \\
\langle u_3, u_1 \rangle & \langle u_3, u_2 \rangle & \langle u_3, u_3 \rangle & \langle u_3, u_4 \rangle & \langle u_3, u_5 \rangle & \langle u_3, u_6 \rangle \\
\langle u_4, u_1 \rangle & \langle u_4, u_2 \rangle & \langle u_4, u_3 \rangle & \langle u_4, u_4 \rangle & \langle u_4, u_5 \rangle & \langle u_4, u_6 \rangle \\
\langle u_5, u_1 \rangle & \langle u_5, u_2 \rangle & \langle u_5, u_3 \rangle & \langle u_5, u_4 \rangle & \langle u_5, u_5 \rangle & \langle u_5, u_6 \rangle \\
\langle u_6, u_1 \rangle & \langle u_6, u_2 \rangle & \langle u_6, u_3 \rangle & \langle u_6, u_4 \rangle & \langle u_6, u_5 \rangle & \langle u_6, u_6 \rangle
\end{bmatrix}$$

Written 6x6 matrix and example computation for the inner products of $\langle u_i, u_j \rangle$

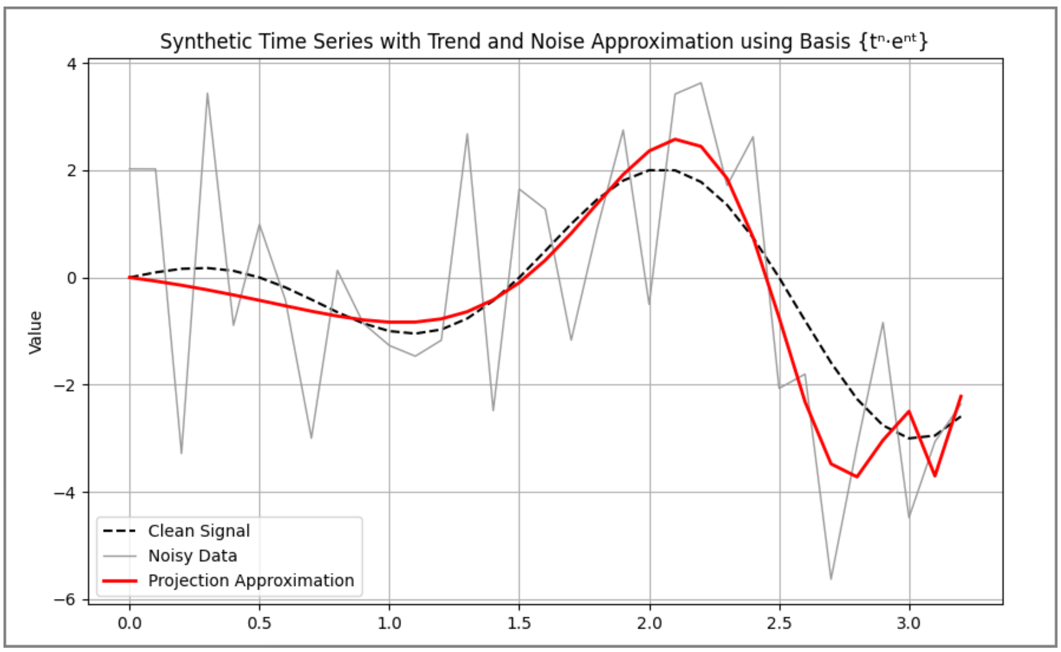
```
G = np.zeros((6, 6))
b = np.zeros(6)

for i in range(6):
    for j in range(6):
        # Inner product: G[i,j] = ∫ u_i(t) u_j(t) dt
        integrand = lambda t: basis_funcs(t, i) * basis_funcs(t, j)
        G[i, j] = np.trapz(integrand(t), t)

    # Projection: b[i] = ∫ x_interp(t) u_i(t) dt
    integrand_b = lambda t: x_interp(t) * basis_funcs(t, i)
    b[i] = np.trapz(integrand_b(t), t)

# Solve Gc = b for coefficients (required for projection)
coefficients = np.linalg.solve(G, b)
```

Code snapshot of the implementation and calculation of matrix and inner products for each element

Part 6:

For the approximation of the randomly generated time series the inner product of each element was calculated and then used to solve Ax = b for coefficients.

Programme Output



Synthetic time series approximation using 6 basis functions; $u_n(t) = t^n \cdot e^{nt}$, $n \in \{1,2,3,4,5,6\}$

```python
# Compute Gram matrix A (G) (6×6) and vector b (6×1)
G = np.zeros((6, 6))
b = np.zeros(6)

for i in range(6):
    for j in range(6):
        # Inner product: A[i,j] = ∫ u_i(t) u_j(t) dt
        integrand = lambda t: basis_funcs(t, i) * basis_funcs(t, j)
        G[i, j] = np.trapz(integrand(t), t)

    # Projection: b[i] = ∫ x_interp(t) u_i(t) dt
    integrand_b = lambda t: x_interp(t) * basis_funcs(t, i)
    b[i] = np.trapz(integrand_b(t), t)

# Solve Ac = b for coefficients (required for projection)
coefficients = np.linalg.solve(G, b)

# Reconstruct approximation
approximation = np.zeros_like(t)
for i in range(6):
    approximation += coefficients[i] * basis_funcs(t, i)
```

Code snapshot showing implementation of coefficient calculation

Whole Programme:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d

# Define basis functions {t·eᵗ, t²·e²ᵗ, ..., t⁶·e⁶ᵗ}
def basis_funcs(t, index):
    n = index + 1  # n = 1, 2, ..., 6
    return (t ** n) * np.exp(n * t)

# Generate noisy data (identical to original)
t = np.arange(0, np.pi + 0.1, 0.1)
clean_signal = np.cos(np.pi * t) * t
noise_level = np.std(clean_signal) / np.random.rand()
x = clean_signal + noise_level * np.random.randn(len(t))
x_interp = interp1d(t, x, kind='linear', bounds_error=False, fill_value="extrap")

# Compute Gram matrix A (G) (6×6) and vector b (6×1)
G = np.zeros((6, 6))
b = np.zeros(6)

for i in range(6):
    for j in range(6):
        # Inner product: A[i,j] = ∫ u_i(t) u_j(t) dt
        integrand = lambda t: basis_funcs(t, i) * basis_funcs(t, j)
        G[i, j] = np.trapz(integrand(t), t)

    # Projection: b[i] = ∫ x_interp(t) u_i(t) dt
    integrand_b = lambda t: x_interp(t) * basis_funcs(t, i)
    b[i] = np.trapz(integrand_b(t), t)

# Solve Ac = b for coefficients (required for projection)
coefficients = np.linalg.solve(G, b)

# Reconstruct approximation
approximation = np.zeros_like(t)
for i in range(6):
    approximation += coefficients[i] * basis_funcs(t, i)

# Plot results (same as original)
plt.figure(figsize=(10, 6))
plt.grid(True)
plt.plot(t, clean_signal, 'k--', linewidth=1.5, label='Clean Signal')
plt.plot(t, x, color=[0.6, 0.6, 0.6], linewidth=1, label='Noisy Data')
plt.plot(t, approximation, 'r-', linewidth=2, label='Projection Approximation')
plt.title('Synthetic Time Series with Trend and Noise Approximation using Basis
{tⁿ·eⁿᵗ}')
plt.xlabel('Time (t)')
plt.ylabel('Value')
plt.legend(loc='best')
plt.show()
```