Alif Zaicho Nur Ahmad / 安曼迪 (2025/03/18)
243519015

1 Special Orthogonal Polynomials

Orthogonal polynomials are a class of polynomials that arise in various areas particularly in approximation theory, numerical analysis, and interpolation. They are defined by their orthogonality with respect to a weight function over a specific interval. Special orthogonal polynomials, such as Legendre, Chebyshev (Type 1 and Type 2), Laguerre, and Hermite polynomials, are widely used for approximating functions and solving interpolation problems.

1.1 General polynomial degree n

A general polynomial of degree n is given by:

$$S_n^*(x) = a_0^* + a_1^* x + a_2^* x^2 + \cdots + a_n^* x^n$$

```python
def general(degree, x):
    return x**degree
```

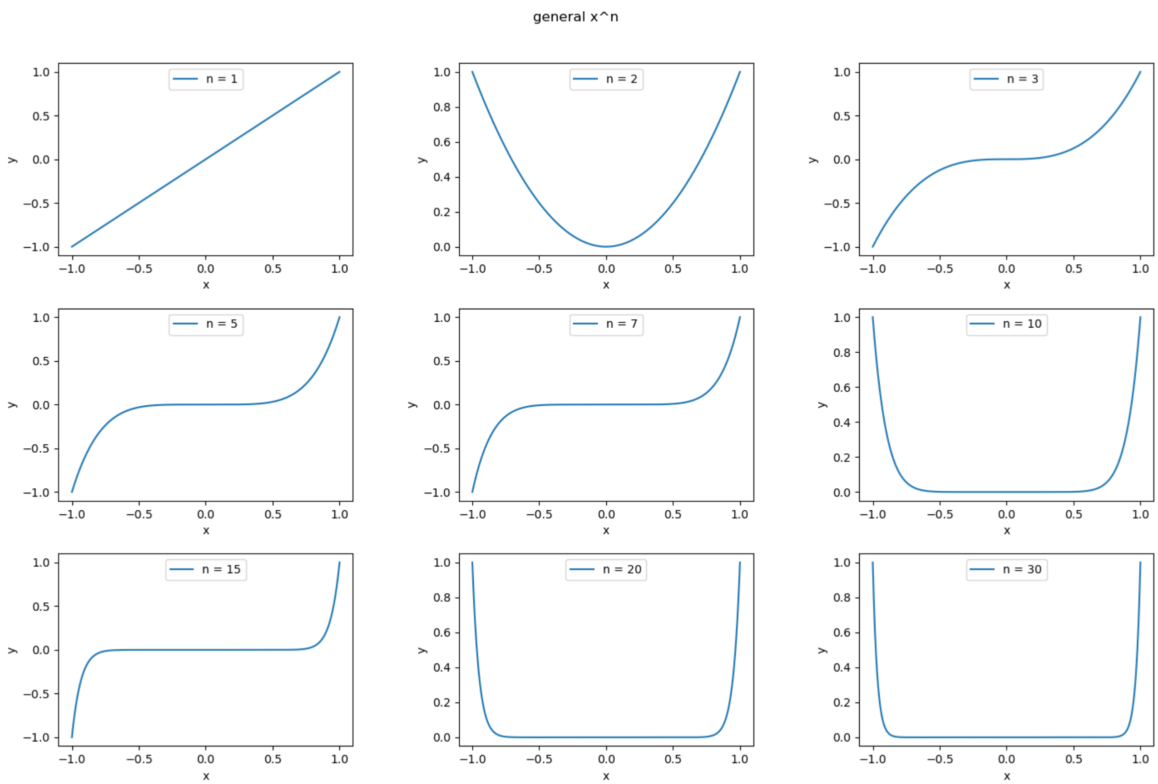

Figure 1: General polynomial $S_n^*(x)$

The inner product of two functions $f(x)$ and $g(x)$ with respect to the weight function $\rho(x) = 1$ on $[-1, 1]$ is defined as:

$$\langle f, g \rangle = \int_{-1}^{1} f(x)g(x)\, dx.$$

For monomials x^i and x^j, the inner product is:

$$\langle x^i, x^j \rangle = \int_{-1}^{1} x^i x^j\, dx.$$

The Hilbert matrix H is constructed using the inner products of the monomials:

$$H_{ij} = \langle x^i, x^j \rangle = \int_{-1}^{1} x^{i+j}\, dx.$$

For $i, j = 0, 1, 2, \ldots, n$, the entries of H are:

$$H_{ij} = \begin{cases} \frac{2}{i+j+1} & \text{if } i+j \text{ is even,} \\ 0 & \text{if } i+j \text{ is odd.} \end{cases}$$

The vector $\mathbf{d}$ is computed using the inner products of $f(x) = e^x$ with the monomials:

$$d_i = \langle f, x^i \rangle = \int_{-1}^{1} e^x x^i\, dx.$$

```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy.integrate import quad

degree_ = 3
lower_bound = 0
upper_bound = 1

def fungsi_awal(x):
    return np.exp(x)

def nama_fungsi_awal():
    return "e^x"

def inner_product_left(i, j, p, poly_type):
    integrand = lambda x: p(i, x) * p(j, x) * weight_function(
        poly_type, x)
    result, _ = quad(integrand, lower_bound, upper_bound)
    return result

def inner_product_right(i, p, f, poly_type):
    integrand = lambda x: f(x) * p(i, x) * weight_function(poly_type,
        x)
    result, _ = quad(integrand, lower_bound, upper_bound)
    return result

def weight_function(poly_type, x):
    if poly_type == "chebyshev_1":
        return 1 / np.sqrt(1 - x**2 + 1e-12)
    elif poly_type == "chebyshev_2":
        return np.sqrt(1 - x**2 + 1e-12)
    elif poly_type == "laguerre":
        return np.exp(-x)
    elif poly_type == "hermite":
        return np.exp(-x**2)
    else:
        return 1

def approximated_function(a, x, p, degree):
    n = 0
    for i in range(degree + 1):
        n = n + a[i] * p(i, x)
    return n

def calling_function(degree, origin_function, function_approximation,
    poly_type):
```

```python
    A = np.zeros((degree + 1, degree + 1))
    Y = np.zeros(degree + 1)

    for i in range(degree + 1):
        for j in range(degree + 1):
            A[i, j] = inner_product_left(i, j, function_approximation, poly_type)
        Y[i] = inner_product_right(i, function_approximation, origin_function, poly_type)

    a = np.linalg.solve(A, Y)
    x = np.linspace(lower_bound, upper_bound, 100)
    y_appx_solver = approximated_function(a, x, function_approximation, degree)
    hilbert_matrix_df = pd.DataFrame(A)
    solution_df = pd.DataFrame(a, columns=['Coefficient'])
    solution_df.index = [f'a{i}' for i in range(len(solution_df))]
    print(f"Hilbert Matrix (A) of {function_approximation.__name__}\n{hilbert_matrix_df}\n")
    print(f"Solution of {function_approximation.__name__} degree {degree}\n{solution_df}\n")
    plt.plot(x, y_appx_solver, label=function_approximation.__name__ + f" degree {degree}")

calling_function(degree_, fungsi_awal, x_power_i, "general")
```

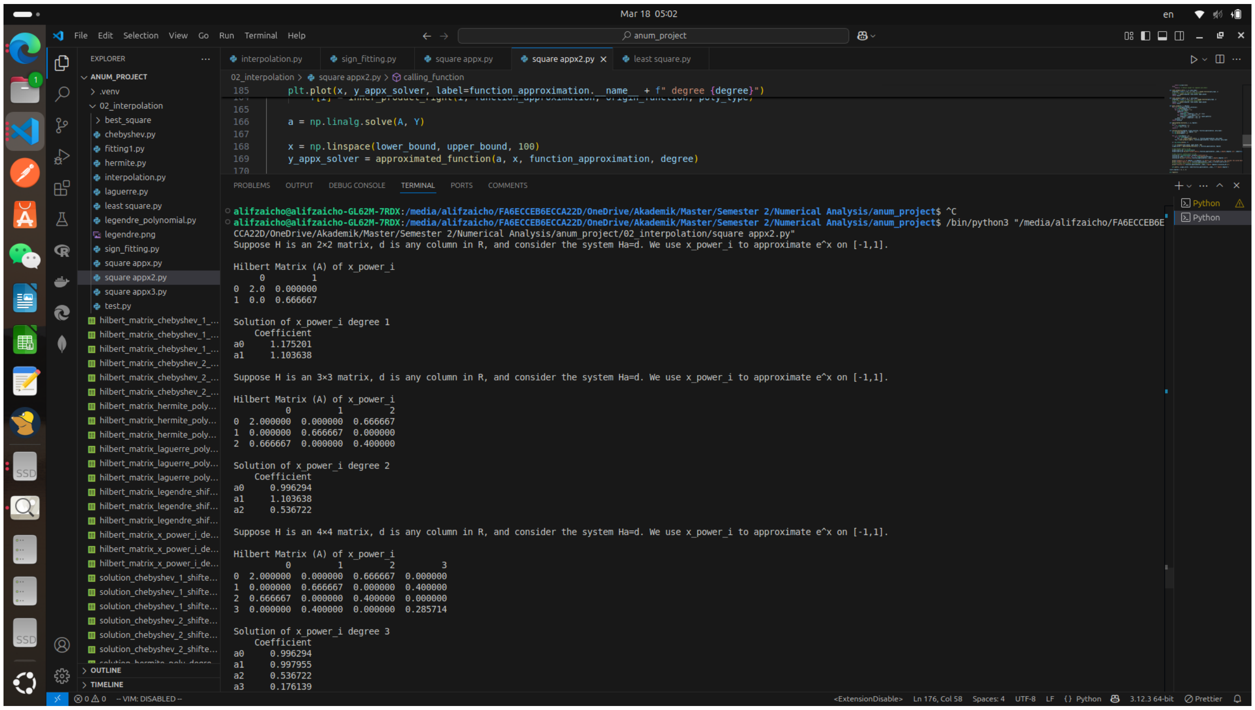

Figure 2: Hilbert Matrix and Solution from general polynomials

2x2 Hilbert Matrix (H) of `x_power_i`

$$H = \begin{bmatrix} 2.0 & 0.000000 \\ 0.0 & 0.666667 \end{bmatrix}$$

Solution of `x_power_i` Degree 1

$$\mathbf{a} = \begin{bmatrix} a_0^* \\ a_1^* \end{bmatrix} = \begin{bmatrix} 1.175201 \\ 1.103638 \end{bmatrix}$$

3x3 Hilbert Matrix (H) of `x_power_i`

$$H = \begin{bmatrix} 2.000000 & 0.000000 & 0.666667 \\ 0.000000 & 0.666667 & 0.000000 \\ 0.666667 & 0.000000 & 0.400000 \end{bmatrix}$$

Solution of `x_power_i` Degree 2

$$\mathbf{a} = \begin{bmatrix} a_0^* \\ a_1^* \\ a_2^* \end{bmatrix} = \begin{bmatrix} 0.996294 \\ 1.103638 \\ 0.536722 \end{bmatrix}$$

4x4 Hilbert Matrix (H) of `x_power_i`

$$H = \begin{bmatrix} 2.000000 & 0.000000 & 0.666667 & 0.000000 \\ 0.000000 & 0.666667 & 0.000000 & 0.400000 \\ 0.666667 & 0.000000 & 0.400000 & 0.000000 \\ 0.000000 & 0.400000 & 0.000000 & 0.285714 \end{bmatrix}$$

Solution of `x_power_i` Degree 3

$$\mathbf{a} = \begin{bmatrix} a_0^* \\ a_1^* \\ a_2^* \\ a_3^* \end{bmatrix} = \begin{bmatrix} 0.996294 \\ 0.997952 \\ 0.536722 \\ 0.176139 \end{bmatrix}$$

```
insert_degrees = [1, 2, 3]

plt.figure(1)
for i in insert_degrees:
    calling_function(i, fungsi_awal, x_power_i, "general")
dotted_line = np.linspace(lower_bound, upper_bound, 10)
plt.scatter(dotted_line, fungsi_awal(dotted_line), label=f'Original
    Function {nama_fungsi_awal()}')
plt.title(f'Approximation of {nama_fungsi_awal()} using {x_power_i.
    __name__} polynomial')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```

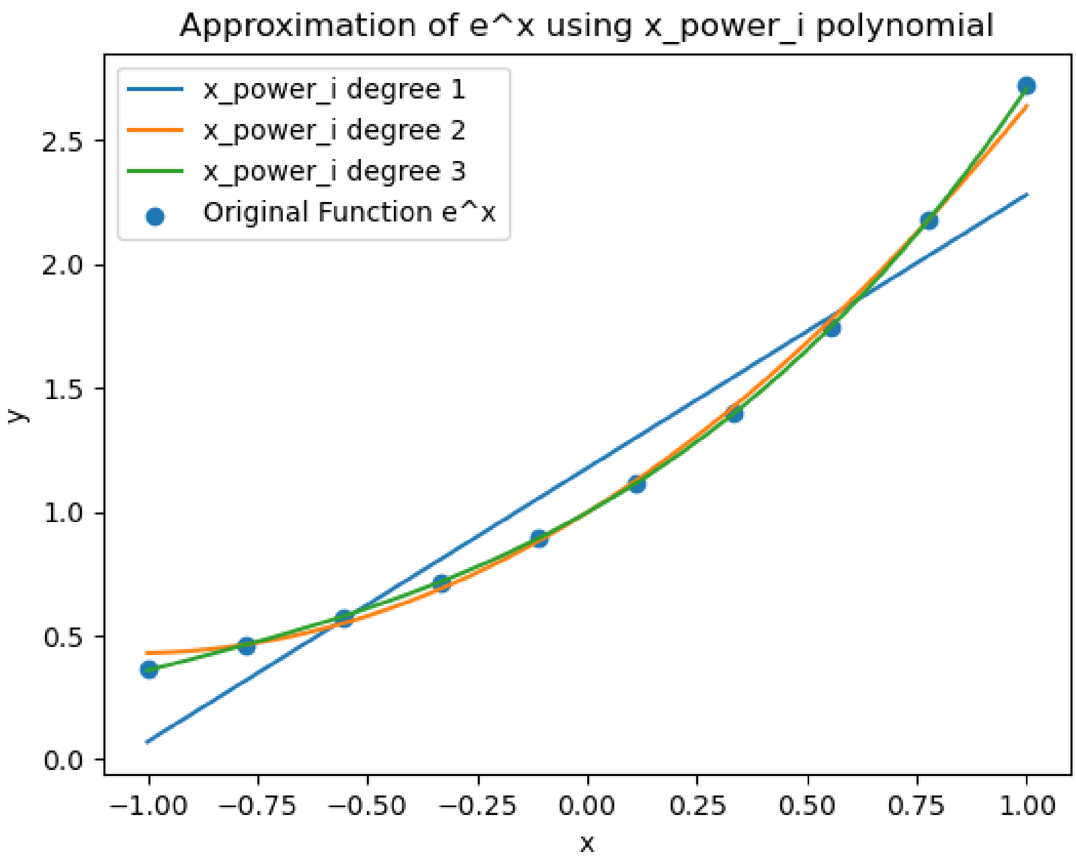

Figure 3: Approximation of e^x using $S_n^*(x)$

1.2 Legendre Polynomials

Legendre polynomials $P_n(x)$ are orthogonal with respect to the weight function $w(x) = 1$ on the interval $[-1, 1]$.

$$\int_{-1}^{1} P_m(x)P_n(x)\,dx = 0 \quad \text{for} \quad m \neq n$$

The polynomials are normalized such that $P_n(1) = 1$.
Recurrence Relation:

$$(n+1)P_{n+1}(x) = (2n+1)xP_n(x) - nP_{n-1}(x)$$

```
def legendre_poly(degree, x):
    if degree == 0:
        return np.ones_like(x)
    elif degree == 1:
        return x
    else:
```

```python
        Pn_previous = np.ones_like(x)
        Pn_current = x
        for i in range(1, degree):
            Pn_new = ((2 * i + 1) * x * Pn_current - i * Pn_previous) / (i + 1)
            Pn_previous = Pn_current
            Pn_current = Pn_new
        return Pn_current

def legendre_shifted(degree, x):
    t = (2 * x - (lower_bound + upper_bound)) / (upper_bound - lower_bound)
    return legendre_poly(degree, t)
```

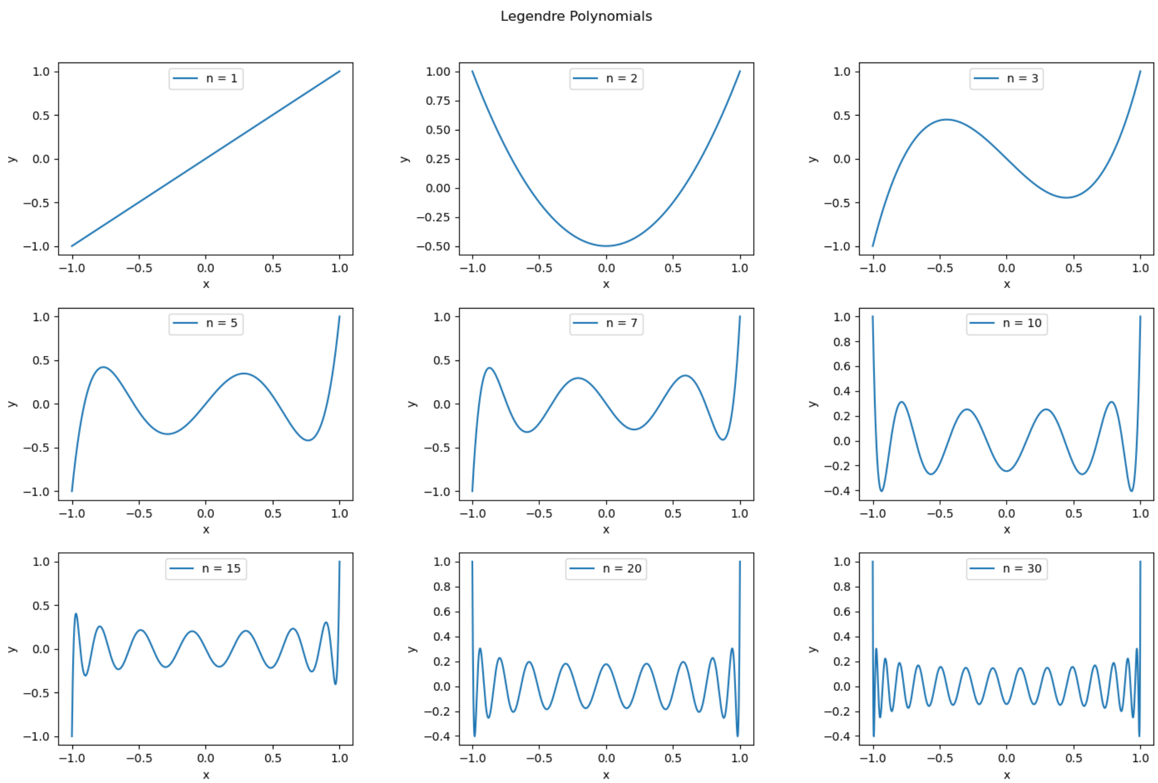

Figure 4: Legendre Polynomials

Suppose the approximation of $f(x) = e^x$ using Legendre polynomials of degree not greater than 3 is given by:

$$f(x) \approx a_0 P_0(x) + a_1 P_1(x) + a_2 P_2(x) + a_3 P_3(x),$$

where $P_0(x), P_1(x), P_2(x), P_3(x)$ are the first four Legendre polynomials:

$$\begin{aligned}
P_0(x) &= 1, \\
P_1(x) &= x, \\
P_2(x) &= \frac{1}{2}(3x^2 - 1), \\
P_3(x) &= \frac{1}{2}(5x^3 - 3x).
\end{aligned}$$

The inner product of two functions $f(x)$ and $g(x)$ with respect to the weight function $\rho(x) = 1$ on $[-1, 1]$ is defined as:

$$\langle f, g \rangle = \int_{-1}^{1} f(x)g(x)\, dx.$$

For Legendre polynomials, the inner product of $P_m(x)$ and $P_n(x)$ is:

$$\langle P_m, P_n \rangle = \int_{-1}^{1} P_m(x)P_n(x)\, dx,$$

The Hilbert matrix H is constructed using the inner products of the Legendre polynomials:

$$H_{ij} = \langle P_i, P_j \rangle = \int_{-1}^{1} P_i(x)P_j(x)\, dx.$$

The vector $\mathbf{d}$ is computed using the inner products of $f(x) = e^x$ with the Legendre polynomials:

$$d_i = \langle f, P_i \rangle = \int_{-1}^{1} e^x P_i(x)\, dx.$$

```
for i in insert_degrees:
    calling_function(i, fungsi_awal, legendre_shifted, "legendre")
```

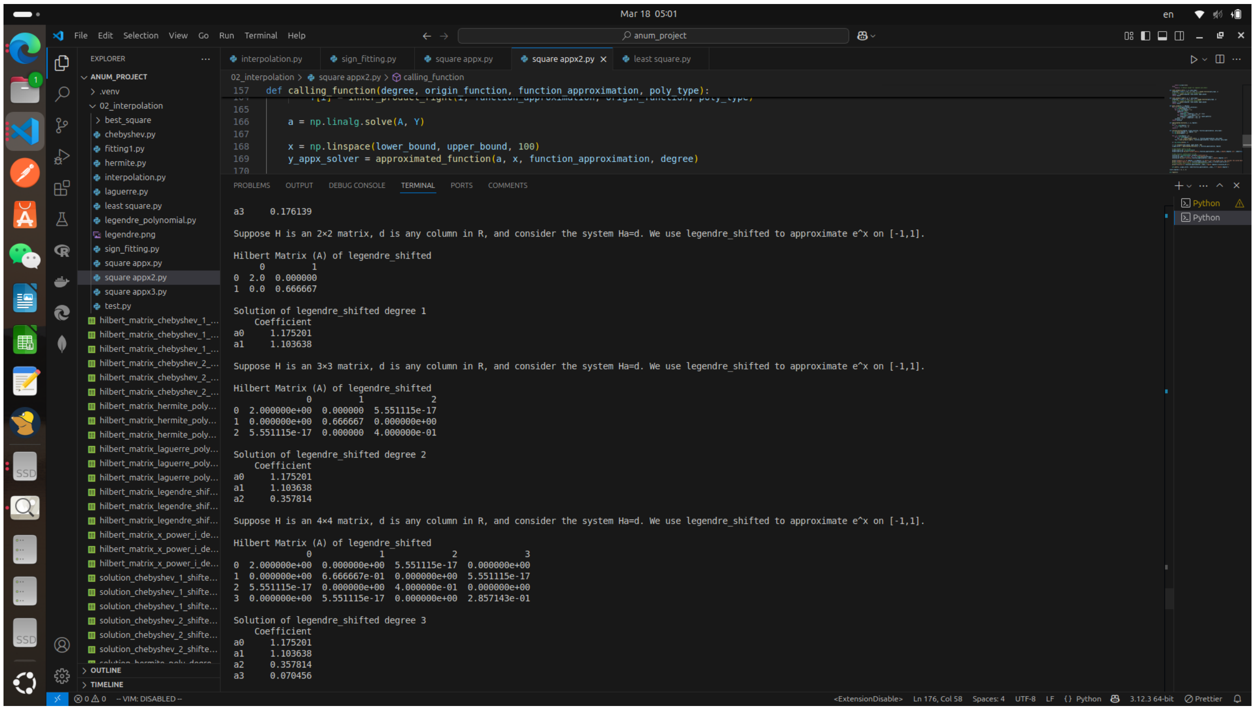

Figure 5: Hilbert Matrix and Solution from legendre polynomials

2x2 Hilbert Matrix (H) of `legendre_shifted`

$$H = \begin{bmatrix} 2.0 & 0.000000 \\ 0.0 & 0.666667 \end{bmatrix}$$

Solution of `legendre_shifted` Degree 1

$$\mathbf{a} = \begin{bmatrix} a_0^* \\ a_1^* \end{bmatrix} = \begin{bmatrix} 1.175201 \\ 1.103638 \end{bmatrix}$$

3x3 Hilbert Matrix (H) of `legendre_shifted`

$$H = \begin{bmatrix} 2.000000 & 0.000000 & 5.551115e-17 \\ 0.000000 & 0.666667 & 0.000000 \\ 5.551115e-17 & 0.000000 & 0.400000 \end{bmatrix}$$

Solution of `legendre_shifted` Degree 2

$$\mathbf{a} = \begin{bmatrix} a_0^* \\ a_1^* \\ a_2^* \end{bmatrix} = \begin{bmatrix} 1.175201 \\ 1.103638 \\ 0.357814 \end{bmatrix}$$

4x4 Hilbert Matrix (H) of `legendre_shifted`

$$H = \begin{bmatrix} 2.000000 & 0.000000 & 5.551115e-17 & 0.000000 \\ 0.000000 & 0.666667 & 0.000000 & 5.551115e-17 \\ 5.551115e-17 & 0.000000 & 0.400000 & 0.000000 \\ 0.000000 & 5.551115e-17 & 0.000000 & 0.285714 \end{bmatrix}$$

Solution of `legendre_shifted` Degree 3

$$\mathbf{a} = \begin{bmatrix} a_0^* \\ a_1^* \\ a_2^* \\ a_3^* \end{bmatrix} = \begin{bmatrix} 1.175201 \\ 1.103638 \\ 0.357814 \\ 0.070456 \end{bmatrix}$$

```
plt.figure(2)
for i in insert_degrees:
    calling_function(i, fungsi_awal, legendre_shifted, "legendre")
dotted_line = np.linspace(lower_bound, upper_bound, 10)
plt.scatter(dotted_line, fungsi_awal(dotted_line), label=f'Original
    Function {nama_fungsi_awal()}')
plt.title(f'Approximation of {nama_fungsi_awal()} using {
    legendre_shifted.__name__} polynomial')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```

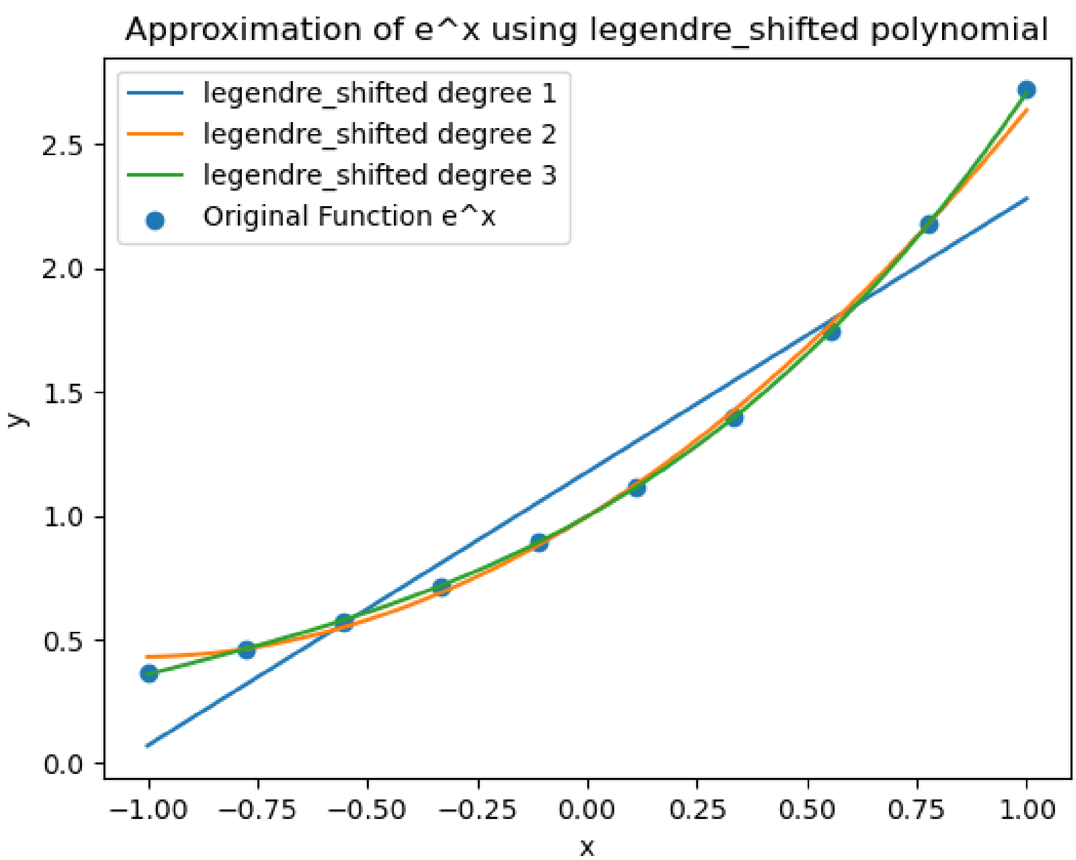

Figure 6: Approximation of e^x using legendre polynomials

1.3 Chebyshev Polynomials

1.3.1 Chebyshev 1

Orthogonality:

$$\int_{-1}^{1} \frac{T_m(x)T_n(x)}{\sqrt{1-x^2}}\,dx = 0 \quad \text{for} \quad m \neq n$$

Recurrence Relation:

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$$

```

def chebyshev_1_poly(degree, x):
    if degree == 0:
        return np.ones_like(x)
    elif degree == 1:
        return x
    else:
        T_previous = np.ones_like(x)
        T_current = x
        for i in range(1, degree):
            T_new = 2*x*T_current-T_previous
            T_previous = T_current
            T_current = T_new
        return T_current

def chebyshev_1_shifted(degree, x):
    t = (2*x-(lower_bound+upper_bound))/(upper_bound-lower_bound)
    return chebyshev_1_poly(degree, t)

fig, axs = plt.subplots(n_rows, n_cols, figsize=(15, 15))
fig.suptitle('Chebyshev_1 Polynomials')
for i, ax in enumerate(axs.flat):
    if i < len(degrees_polynomial):
        y = chebyshev_1_poly(degrees_polynomial[i],x)
        ax.plot(x, y, label=f'n = {degrees_polynomial[i]}')
        # ax.set_title(f'Degree {degrees_polynomial[i]}')
        ax.set(xlabel='x', ylabel='y')
        ax.legend(loc="upper center")
    else:
        ax.axis('off')
plt.tight_layout(pad=4.0)
```

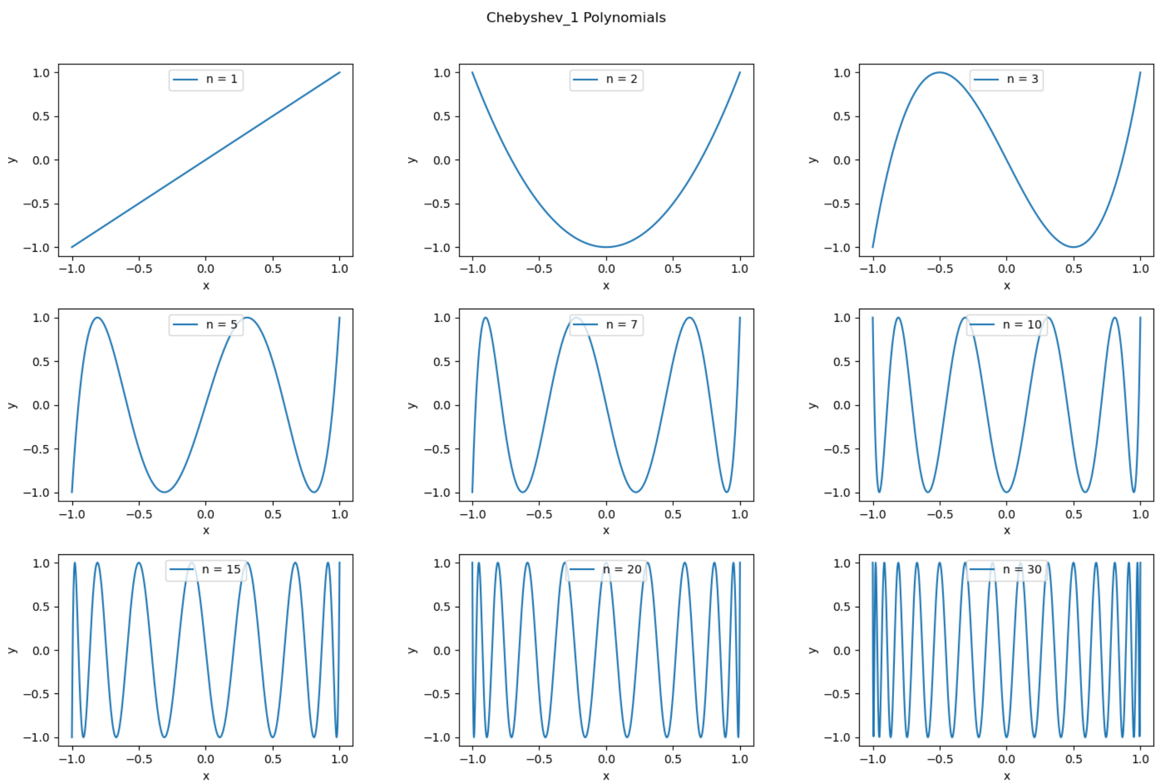

Figure 7: Chebyshev polynomials type 1

Suppose the approximation of $f(x) = e^x$ using Chebyshev type 1 polynomials of degree not greater than 3 is given by:

$$f(x) \approx a_0T_0(x) + a_1T_1(x) + a_2T_2(x) + a_3T_3(x),$$

where $T_0(x), T_1(x), T_2(x), T_3(x)$ are the first four Chebyshev type 1 polynomials:

$$\begin{aligned} T_0(x) &= 1, \\ T_1(x) &= x, \\ T_2(x) &= 2x^2 - 1, \\ T_3(x) &= 4x^3 - 3x. \end{aligned}$$

The inner product of two functions $f(x)$ and $g(x)$ with respect to the weight function $\rho(x) = \frac{1}{\sqrt{1-x^2}}$ on $[-1, 1]$ is defined as:

$$\langle f, g\rangle = \int_{-1}^{1} f(x)g(x)\frac{1}{\sqrt{1-x^2}}\,dx.$$

For Chebyshev polynomials of the first kind, the inner product of $T_m(x)$ and $T_n(x)$ is:

$$\langle T_m, T_n\rangle = \int_{-1}^{1} T_m(x)T_n(x)\frac{1}{\sqrt{1-x^2}}\,dx.$$

The Hilbert matrix H is constructed using the inner products of the Chebyshev polynomials:

$$H_{ij} = \langle T_i, T_j \rangle = \int_{-1}^{1} T_i(x)T_j(x)\frac{1}{\sqrt{1-x^2}}\,dx.$$

The vector $\mathbf{d}$ is computed using the inner products of $f(x) = e^x$ with the Chebyshev polynomials:

$$d_i = \langle f, T_i \rangle = \int_{-1}^{1} e^x T_i(x)\frac{1}{\sqrt{1-x^2}}\,dx.$$

```python
for i in insert_degrees:
    calling_function(i, fungsi_awal, chebyshev_1_shifted, "
        chebyshev_1")
```

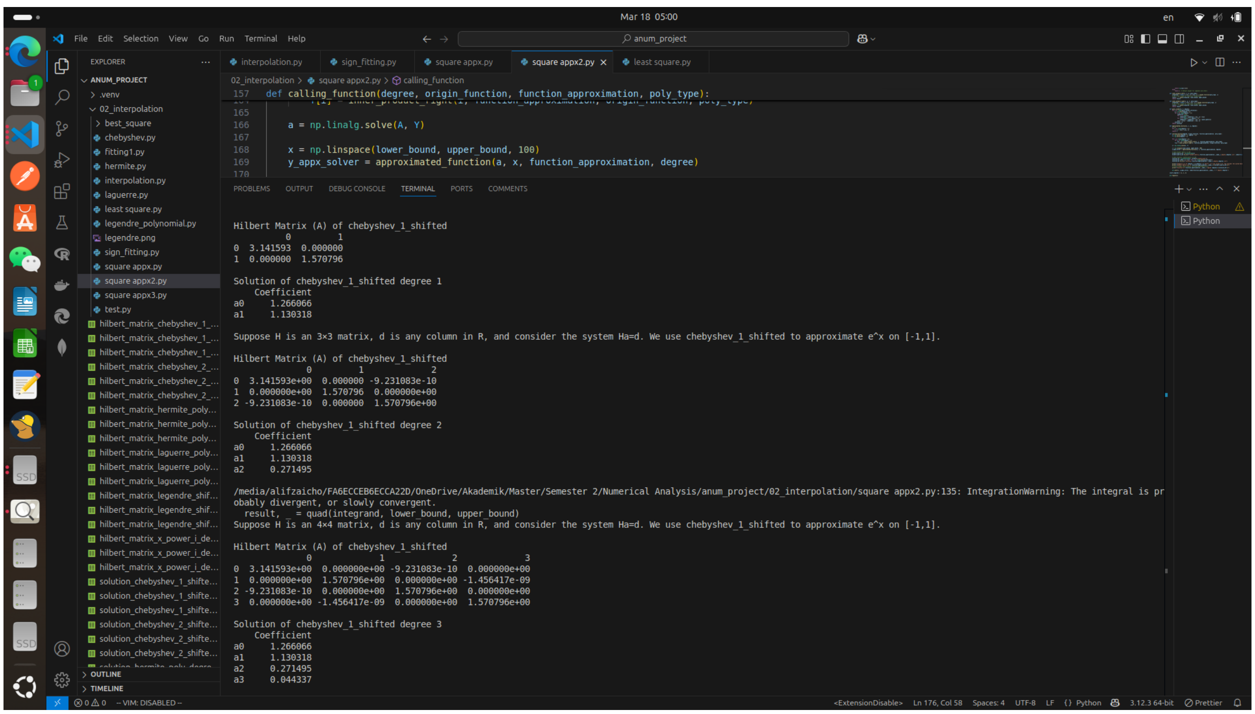

Figure 8: Hilbert Matrix and Solution from chebyshev type 1 polynomials

2x2 Hilbert Matrix (H) of `chebyshev_1_shifted`

$$H = \begin{bmatrix} 3.141593 & 0.000000 \\ 0.000000 & 1.570796 \end{bmatrix}$$

Solution of `chebyshev_1_shifted` Degree 1

$$\mathbf{a} = \begin{bmatrix} a_0^* \\ a_1^* \end{bmatrix} = \begin{bmatrix} 1.266066 \\ 1.139318 \end{bmatrix}$$

3x3 Hilbert Matrix (H) of `chebyshev_1_shifted`

$$H = \begin{bmatrix} 3.141593 & 0.000000 & -9.231803e-10 \\ 0.000000 & 1.570796 & 0.000000 \\ -9.231803e-10 & 0.000000 & 1.570796 \end{bmatrix}$$

Solution of `chebyshev_1_shifted` Degree 2

$$\mathbf{a} = \begin{bmatrix} a_0^* \\ a_1^* \\ a_2^* \end{bmatrix} = \begin{bmatrix} 1.266066 \\ 1.139318 \\ 0.274495 \end{bmatrix}$$

4x4 Hilbert Matrix (H) of `chebyshev_1_shifted`

$$H = \begin{bmatrix} 3.141593 & 0.000000 & -9.231803e-10 & 0.000000 \\ 0.000000 & 1.570796 & 0.000000 & -1.456417e-09 \\ -9.231803e-10 & 0.000000 & 1.570796 & 0.000000 \\ 0.000000 & -1.456417e-09 & 0.000000 & 1.570796 \end{bmatrix}$$

Solution of `chebyshev_1_shifted` Degree 3

$$\mathbf{a} = \begin{bmatrix} a_0^* \\ a_1^* \\ a_2^* \\ a_3^* \end{bmatrix} = \begin{bmatrix} 1.266066 \\ 1.139318 \\ 0.274495 \\ 0.644337 \end{bmatrix}$$

```python
plt.figure(3)
for i in insert_degrees:
    calling_function(i, fungsi_awal, chebyshev_1_shifted, "chebyshev_1")
dotted_line = np.linspace(lower_bound, upper_bound, 10)
plt.scatter(dotted_line, fungsi_awal(dotted_line), label=f'Original Function {nama_fungsi_awal()}')
plt.title(f'Approximation of {nama_fungsi_awal()} using {chebyshev_1_shifted.__name__} polynomial')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```

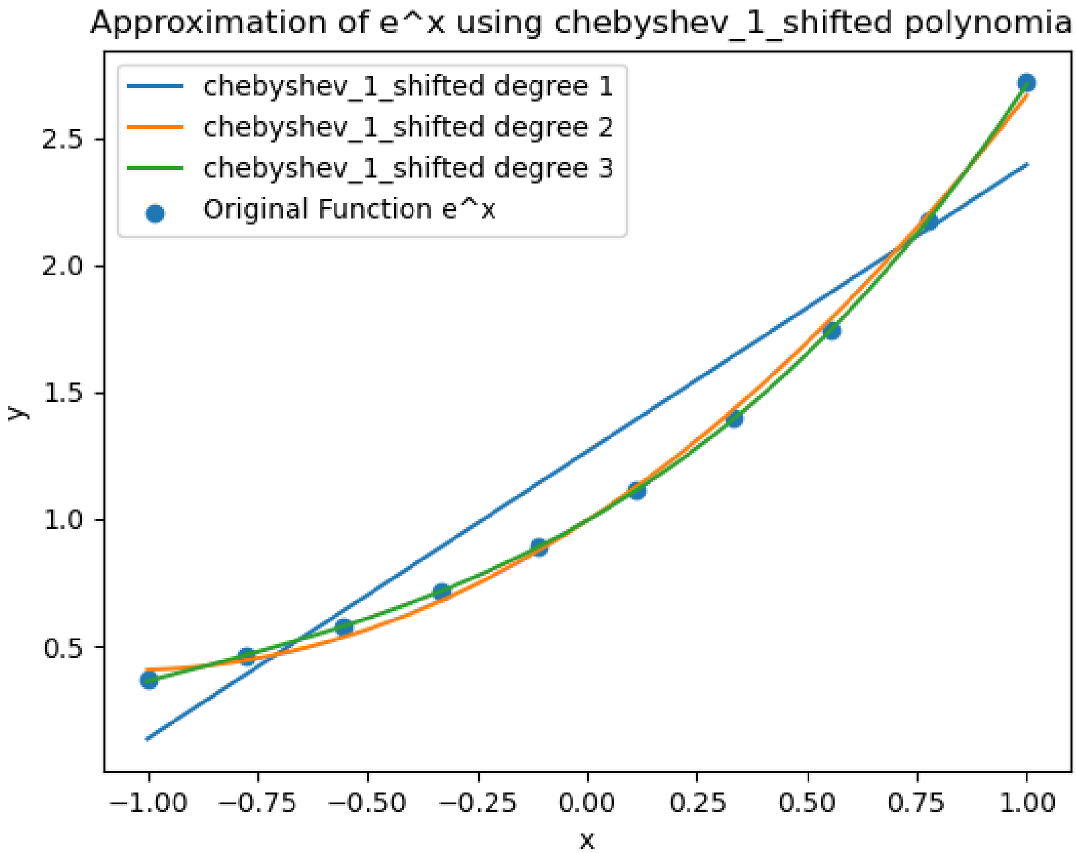

Figure 9: Approximation of e^x using chebyshev type 1 polynomials

1.3.2 Chebyshev 2

Orthogonality:

$$\int_{-1}^{1} U_m(x)U_n(x)\sqrt{1-x^2}\,dx = 0 \quad \text{for} \quad m \neq n$$

Recurrence Relation:

$$U_{n+1}(x) = 2xU_n(x) - U_{n-1}(x)$$

```
def chebyshev_2_poly(degree, x):
    if degree == 0:
        return np.ones_like(x)
    elif degree == 1:
        return 2 * x
    else:
        U_previous = np.ones_like(x)
        U_current = 2 * x
        for i in range(1, degree):
            U_new = 2 * x * U_current - U_previous
```

```
            U_previous = U_current
            U_current = U_new
        return U_current

def chebyshev_2_shifted(degree, x):
    t = (2 * x - (lower_bound + upper_bound)) / (upper_bound - lower_bound)
    return chebyshev_2_poly(degree, t)
```

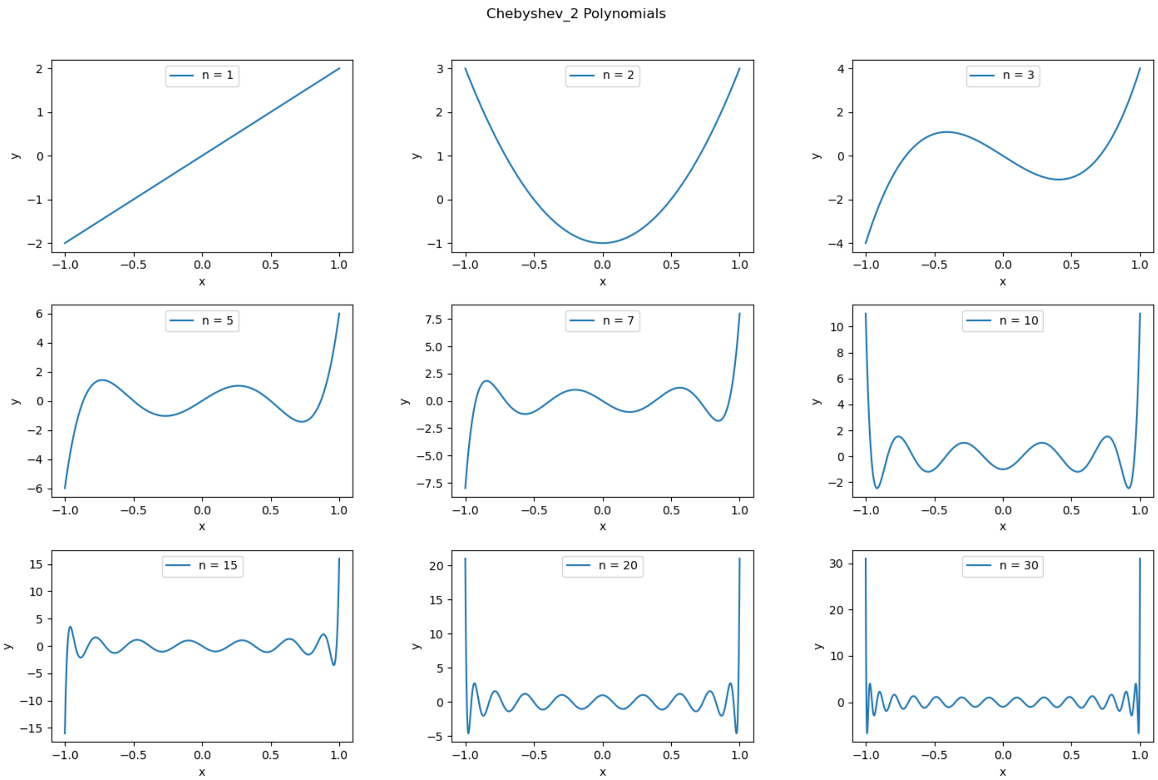

Figure 10: Chebyshev polynomials type 2

Suppose the approximation of $f(x) = e^x$ using Chebyshev type 2 polynomials of degree not greater than 3 is given by:

$$f(x) \approx a_0U_0(x) + a_1U_1(x) + a_2U_2(x) + a_3U_3(x),$$

where $U_0(x), U_1(x), U_2(x), U_3(x)$ are the first four Chebyshev type 2 polynomials:

$$\begin{aligned}
U_0(x) &= 1, \\
U_1(x) &= 2x, \\
U_2(x) &= 4x^2 - 1, \\
U_3(x) &= 8x^3 - 4x.
\end{aligned}$$

The inner product of two functions $f(x)$ and $g(x)$ with respect to the weight function $\rho(x) = \sqrt{1-x^2}$ on $[-1,1]$ is defined as:

$$\langle f, g \rangle = \int_{-1}^{1} f(x)g(x)\sqrt{1-x^2}\, dx.$$

For Chebyshev polynomials of the second kind, the inner product of $U_m(x)$ and $U_n(x)$ is:

$$\langle U_m, U_n \rangle = \int_{-1}^{1} U_m(x)U_n(x)\sqrt{1-x^2}\, dx.$$

The Hilbert matrix H is constructed using the inner products of the Chebyshev polynomials:

$$H_{ij} = \langle U_i, U_j \rangle = \int_{-1}^{1} U_i(x)U_j(x)\sqrt{1-x^2}\, dx.$$

The vector $\mathbf{d}$ is computed using the inner products of $f(x) = e^x$ with the Chebyshev polynomials:

$$d_i = \langle f, U_i \rangle = \int_{-1}^{1} e^x U_i(x)\sqrt{1-x^2}\, dx.$$

```
for i in insert_degrees:
    calling_function(i, fungsi_awal, chebyshev_2_shifted, "
        chebyshev_2")
```

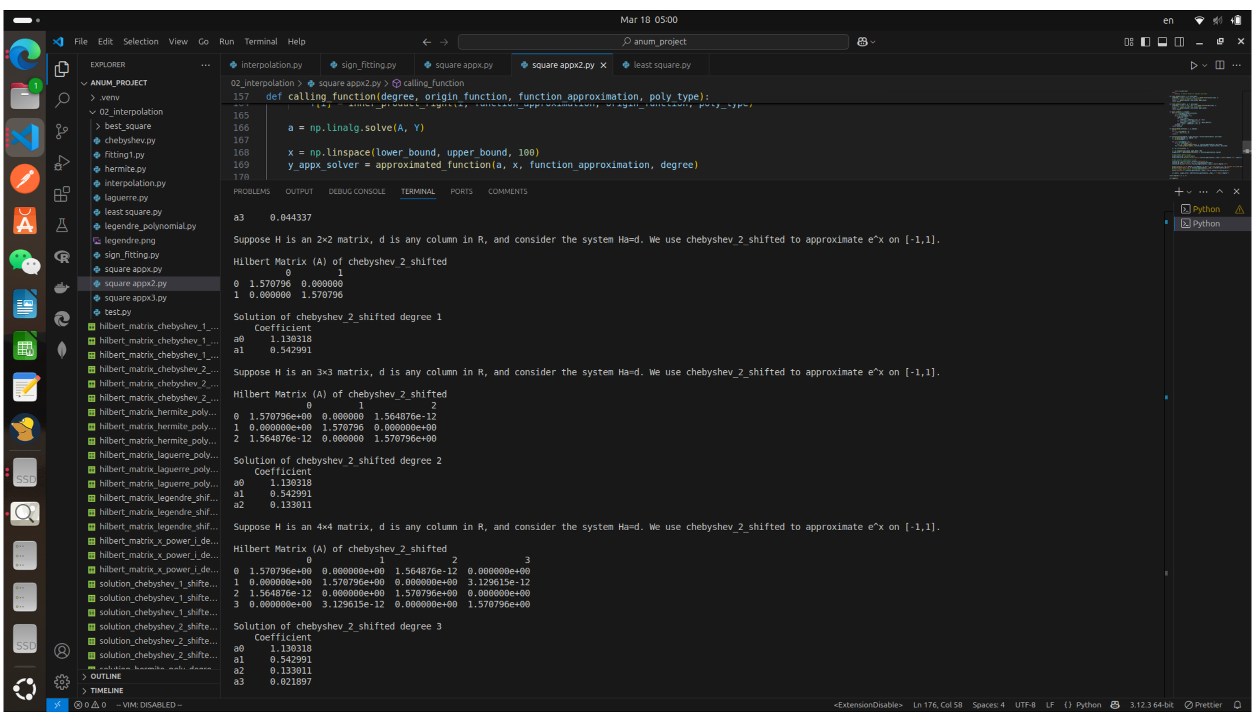

Figure 11: Hilbert Matrix and Solution from chebyshev type 2 polynomials

2x2 Hilbert Matrix (H) of `chebyshev_2_shifted`

$$H = \begin{bmatrix} 1.570796 & 0.000000 \\ 0.000000 & 1.570796 \end{bmatrix}$$

Solution of `chebyshev_2_shifted` Degree 1

$$\mathbf{a} = \begin{bmatrix} a_0^* \\ a_1^* \end{bmatrix} = \begin{bmatrix} 1.130318 \\ 0.542991 \end{bmatrix}$$

3x3 Hilbert Matrix (H) of `chebyshev_2_shifted`

$$H = \begin{bmatrix} 1.570796 & 0.000000 & 1.564876e-12 \\ 0.000000 & 1.570796 & 0.000000 \\ 1.564876e-12 & 0.000000 & 1.570796 \end{bmatrix}$$

Solution of `chebyshev_2_shifted` Degree 2

$$\mathbf{a} = \begin{bmatrix} a_0^* \\ a_1^* \\ a_2^* \end{bmatrix} = \begin{bmatrix} 1.130318 \\ 0.542991 \\ 0.133011 \end{bmatrix}$$

4x4 Hilbert Matrix (H) of `chebyshev_2_shifted`

$$H = \begin{bmatrix} 1.570796 & 0.000000 & 1.564876e-12 & 0.000000 \\ 0.000000 & 1.570796 & 0.000000 & 3.129615e-12 \\ 1.564876e-12 & 0.000000 & 1.570796 & 0.000000 \\ 0.000000 & 3.129615e-12 & 0.000000 & 1.570796 \end{bmatrix}$$

Solution of `chebyshev_2_shifted` Degree 3

$$\mathbf{a} = \begin{bmatrix} a_0^* \\ a_1^* \\ a_2^* \\ a_3^* \end{bmatrix} = \begin{bmatrix} 1.130318 \\ 0.542991 \\ 0.133011 \\ 0.021897 \end{bmatrix}$$

```
plt.figure(4)
for i in insert_degrees:
    calling_function(i, fungsi_awal, chebyshev_2_shifted, "chebyshev_2")
dotted_line = np.linspace(lower_bound, upper_bound, 10)
plt.scatter(dotted_line, fungsi_awal(dotted_line), label=f'Original Function {nama_fungsi_awal()}')
plt.title(f'Approximation of {nama_fungsi_awal()} using {chebyshev_2_shifted.__name__} polynomial')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```

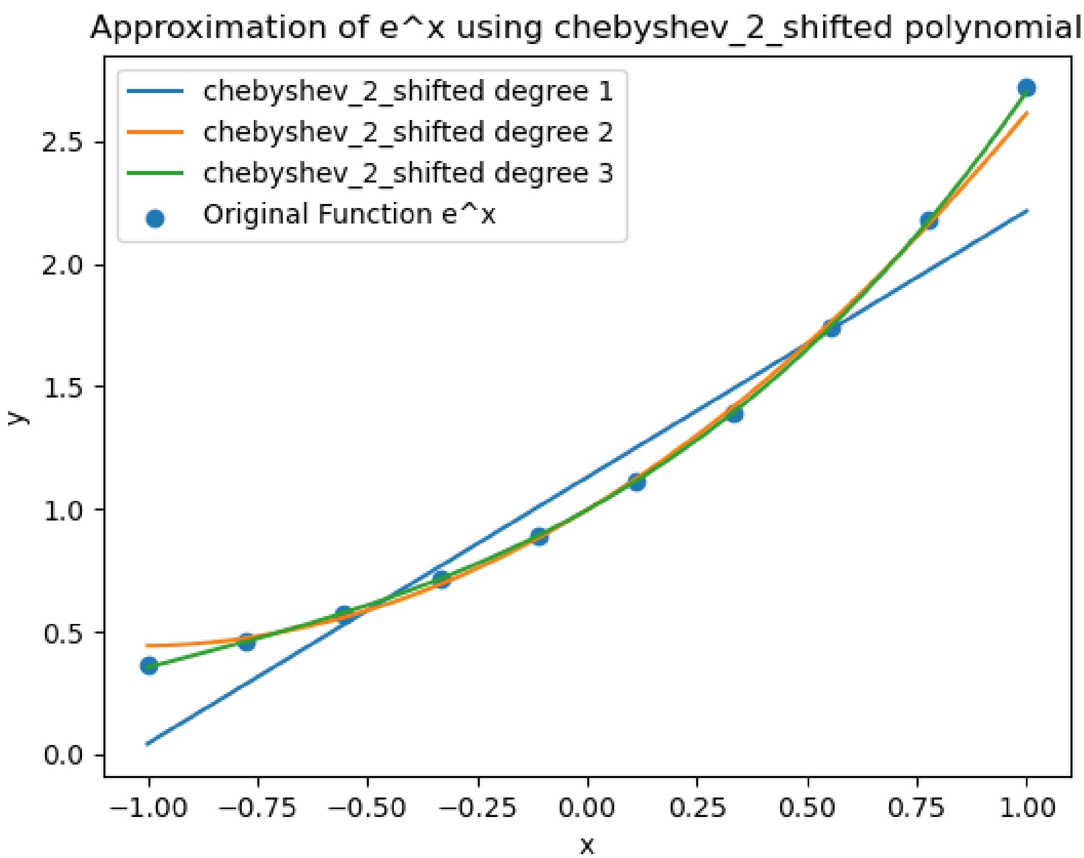

Figure 12: Approximation of e^x using chebyshev type 2 polynomials

1.4 Laguerre Polynomials

Definition:

$$\int_0^\infty L_m(x)L_n(x)e^{-x}\,dx = 0 \quad \text{for} \quad m \neq n$$

Recurrence Relation:

$$(n+1)L_{n+1}(x) = (2n+1-x)L_n(x) - nL_{n-1}(x)$$

```
def laguerre_poly(degree, x):
    if degree == 0:
        return np.ones_like(x)
    elif degree == 1:
        return 1 - x
    else:
        L_previous = np.ones_like(x)
        L_current = 1 - x
        for i in range(1, degree):
```

```
            L_new = ((2 * i + 1 - x) * L_current - i * L_previous) / (i + 1)
            L_previous = L_current
            L_current = L_new
        return L_current
```

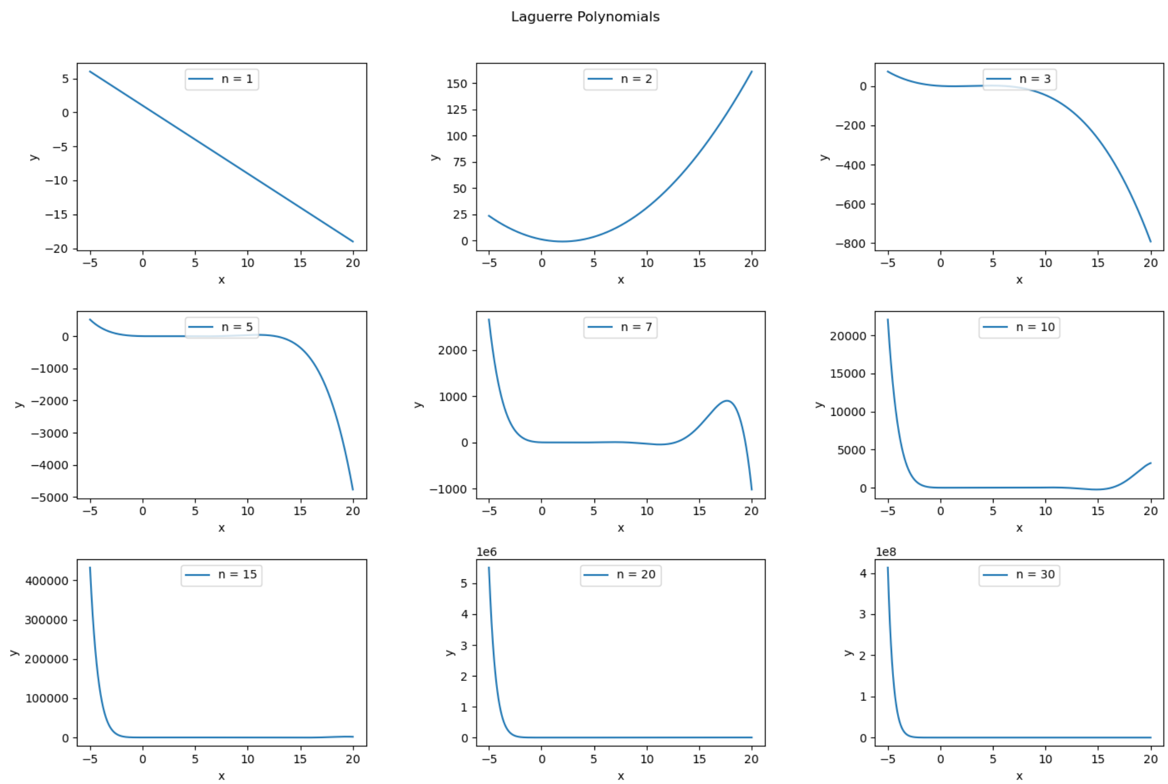

Figure 13: Laguerre polynomials

Suppose the approximation of $f(x) = e^x$ using Laguerre polynomials of degree not greater than 3 is given by:

$$f(x) \approx a_0 L_0(x) + a_1 L_1(x) + a_2 L_2(x) + a_3 L_3(x),$$

where $L_0(x), L_1(x), L_2(x), L_3(x)$ are the first four Laguerre polynomials:

$$\begin{aligned} L_0(x) &= 1, \\ L_1(x) &= -x + 1, \\ L_2(x) &= \frac{1}{2}(x^2 - 4x + 2), \\ L_3(x) &= \frac{1}{6}(-x^3 + 9x^2 - 18x + 6). \end{aligned}$$

The inner product of two functions $f(x)$ and $g(x)$ with respect to the weight function $\rho(x) =$

e^{-x} on $[0,1]$ is defined as:

$$\langle f, g \rangle = \int_0^1 f(x)g(x)e^{-x}\,dx.$$

For Laguerre polynomials, the inner product of $L_m(x)$ and $L_n(x)$ is:

$$\langle L_m, L_n \rangle = \int_0^1 L_m(x)L_n(x)e^{-x}\,dx.$$

The Hilbert matrix H is constructed using the inner products of the Laguerre polynomials:

$$H_{ij} = \langle L_i, L_j \rangle = \int_0^1 L_i(x)L_j(x)e^{-x}\,dx.$$

The vector $\mathbf{d}$ is computed using the inner products of $f(x) = e^x$ with the Laguerre polynomials:

$$d_i = \langle f, L_i \rangle = \int_0^1 e^x L_i(x)e^{-x}\,dx = \int_0^1 L_i(x)\,dx.$$

```python
for i in insert_degrees:
    calling_function(i, fungsi_awal, laguerre_poly, "laguerre")
```

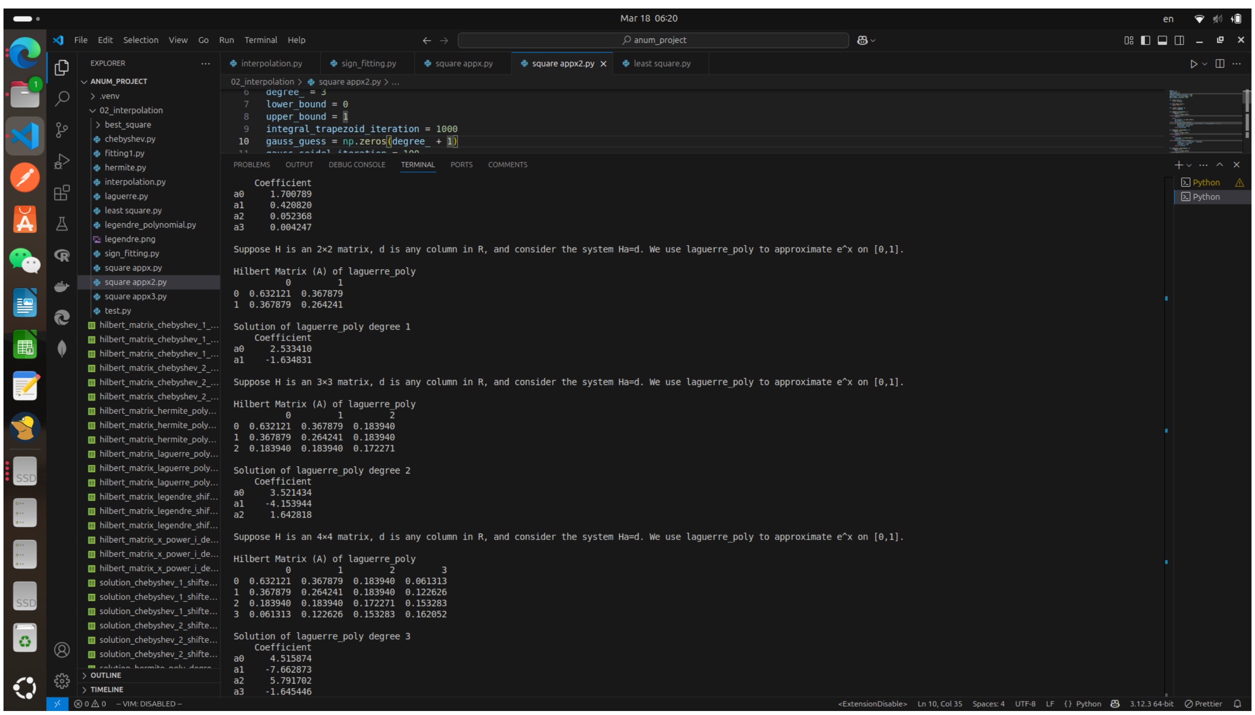

Figure 14: Hilbert Matrix and Solution from laguerre polynomials

2x2 Hilbert Matrix (H) of `laguerre_poly`

$$H = \begin{bmatrix} 0.632121 & 0.367879 \\ 0.367879 & 0.264241 \end{bmatrix}$$

Solution of `laguerre_poly` Degree 1

$$\mathbf{a} = \begin{bmatrix} a_0^* \\ a_1^* \end{bmatrix} = \begin{bmatrix} 2.533410 \\ -1.634831 \end{bmatrix}$$

3x3 Hilbert Matrix (H) of `laguerre_poly`

$$H = \begin{bmatrix} 0.632121 & 0.367879 & 0.183940 \\ 0.367879 & 0.264241 & 0.183940 \\ 0.183940 & 0.183940 & 0.139292 \end{bmatrix}$$

Solution of `laguerre_poly` Degree 2

$$\mathbf{a} = \begin{bmatrix} a_0^* \\ a_1^* \\ a_2^* \end{bmatrix} = \begin{bmatrix} 3.521434 \\ -4.153944 \\ 1.642818 \end{bmatrix}$$

4x4 Hilbert Matrix (H) of `laguerre_poly`

$$H = \begin{bmatrix} 0.632121 & 0.367879 & 0.183940 & 0.061313 \\ 0.367879 & 0.264241 & 0.183940 & 0.122626 \\ 0.183940 & 0.183940 & 0.139292 & 0.061313 \\ 0.061313 & 0.122626 & 0.061313 & 0.030653 \end{bmatrix}$$

Solution of `laguerre_poly` Degree 3

$$\mathbf{a} = \begin{bmatrix} a_0^* \\ a_1^* \\ a_2^* \\ a_3^* \end{bmatrix} = \begin{bmatrix} 4.515874 \\ -7.662873 \\ 5.791702 \\ -1.645446 \end{bmatrix}$$

```python
plt.figure(5)
for i in insert_degrees:
    calling_function(i, fungsi_awal, laguerre_poly, "laguerre")
dotted_line = np.linspace(lower_bound, upper_bound, 10)
plt.scatter(dotted_line, fungsi_awal(dotted_line), label=f'Original Function {nama_fungsi_awal()}')
plt.title(f'Approximation of {nama_fungsi_awal()} using {laguerre_poly.__name__} polynomial')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```

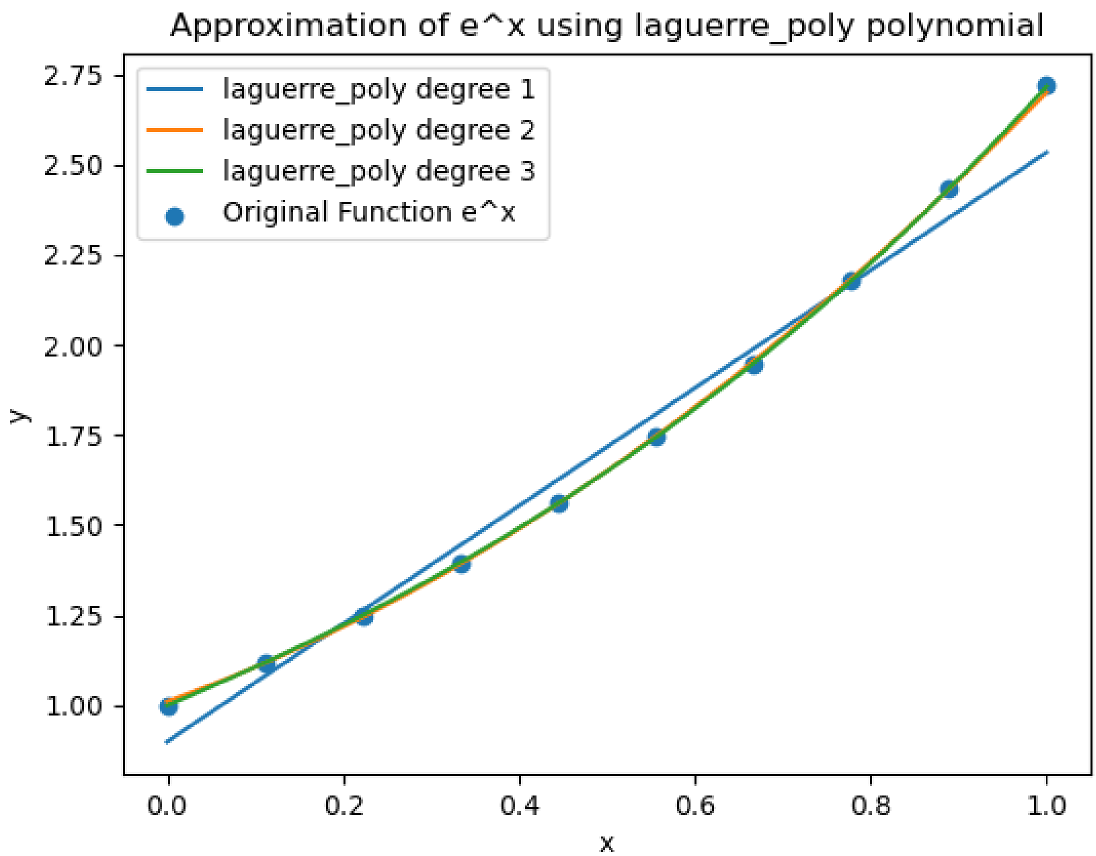

Figure 15: Approximation of e^x using leguerre polynomials

1.5 Hermite Polynomials

Definition:

$$\int_{-\infty}^{\infty} H_m(x)H_n(x)e^{-x^2}\,dx = 0 \quad \text{for} \quad m \neq n$$

Recurrence Relation:

$$H_{n+1}(x) = 2xH_n(x) - 2nH_{n-1}(x)$$

```
def hermite_poly(degree, x):
    if degree == 0:
        return np.ones_like(x)
    elif degree == 1:
        return 2 * x
    else:
        H_previous = np.ones_like(x)
        H_current = 2 * x
        for i in range(1, degree):
            H_new = 2 * x * H_current - 2 * i * H_previous
```

```
            H_previous = H_current
            H_current = H_new
        return H_current
```

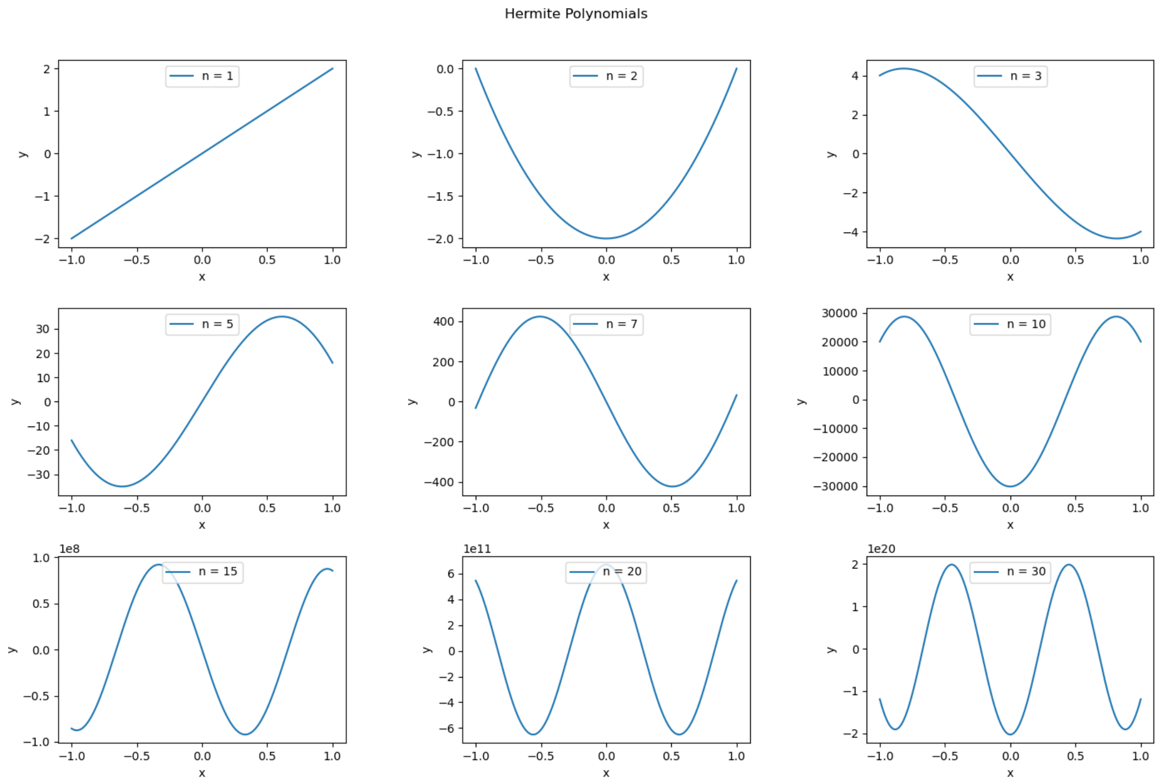

Figure 16: Hermite Polynomials

Suppose the approximation of $f(x) = e^x$ using Hermite polynomials of degree not greater than 3 is given by:

$$f(x) \approx a_0 H_0(x) + a_1 H_1(x) + a_2 H_2(x) + a_3 H_3(x),$$

where $H_0(x), H_1(x), H_2(x), H_3(x)$ are the first four Legendre polynomials:

$$\begin{aligned} H_0(x) &= 1, \\ H_1(x) &= 2x, \\ H_2(x) &= 4x^2 - 2, \\ H_3(x) &= 8x^3 - 12x. \end{aligned}$$

The inner product of two functions $f(x)$ and $g(x)$ with respect to the weight function $\rho(x) = e^{-x^2}$ on $[-1, 1]$ is defined as:

$$\langle f, g \rangle = \int_{-1}^{1} f(x) g(x) e^{-x^2} \, dx.$$

For Hermite polynomials, the inner product of $H_m(x)$ and $H_n(x)$ is:

$$\langle H_m, H_n \rangle = \int_{-1}^{1} H_m(x) H_n(x) e^{-x^2}\, dx.$$

The Hilbert matrix H is constructed using the inner products of the Hermite polynomials:

$$H_{ij} = \langle H_i, H_j \rangle = \int_{-1}^{1} H_i(x) H_j(x) e^{-x^2}\, dx.$$

The vector $\mathbf{d}$ is computed using the inner products of $f(x) = e^x$ with the Hermite polynomials:

$$d_i = \langle f, H_i \rangle = \int_{-1}^{1} e^x H_i(x) e^{-x^2}\, dx = \int_{-1}^{1} e^x H_i(x) e^{-x^2}\, dx.$$

```python
for i in insert_degrees:
    calling_function(i, fungsi_awal, hermite_poly, "hermite")
```

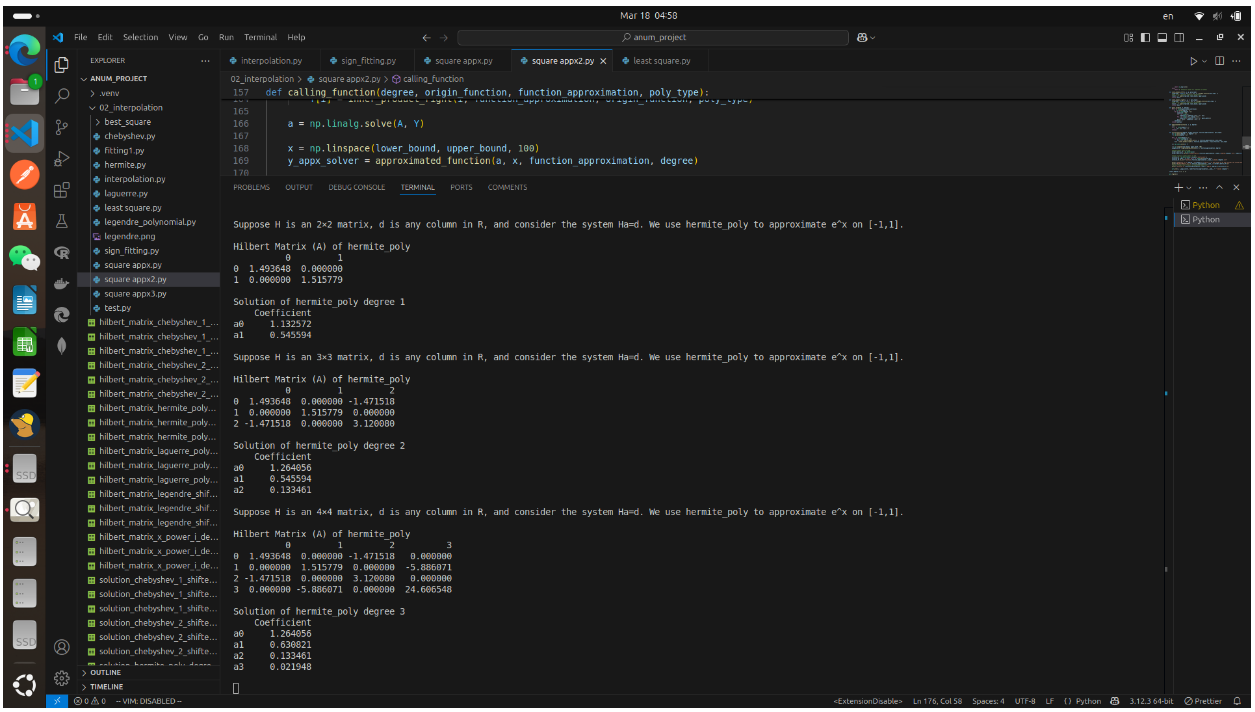

Figure 17: Hilbert Matrix and Solution from hermite polynomials

Hilbert Matrices and Solutions for Hermite Polynomial Approximation.
2x2 Hilbert Matrix (H) of `hermite_poly`

$$H = \begin{bmatrix} 1.493648 & 0.000000 \\ 0.000000 & 1.515779 \end{bmatrix}$$

Solution of `hermite_poly` Degree 1

$$\mathbf{a} = \begin{bmatrix} a_0^* \\ a_1^* \end{bmatrix} = \begin{bmatrix} 1.132572 \\ 0.545594 \end{bmatrix}$$

3x3 Hilbert Matrix (H) of `hermite_poly`

$$H = \begin{bmatrix} 1.493648 & 0.000000 & -1.471518 \\ 0.000000 & 1.515779 & 0.000000 \\ -1.471518 & 0.000000 & 3.120080 \end{bmatrix}$$

Solution of `hermite_poly` Degree 2

$$\mathbf{a} = \begin{bmatrix} a_0^* \\ a_1^* \\ a_2^* \end{bmatrix} = \begin{bmatrix} 1.264056 \\ 0.545594 \\ 0.133461 \end{bmatrix}$$

4x4 Hilbert Matrix (H) of `hermite_poly`

$$H = \begin{bmatrix} 1.493648 & 0.000000 & -1.471518 & 0.000000 \\ 0.000000 & 1.515779 & 0.000000 & -5.886071 \\ -1.471518 & 0.000000 & 3.120080 & 0.000000 \\ 0.000000 & -5.886071 & 0.000000 & 24.606548 \end{bmatrix}$$

Solution of `hermite_poly` Degree 3

$$\mathbf{a} = \begin{bmatrix} a_0^* \\ a_1^* \\ a_2^* \\ a_3^* \end{bmatrix} = \begin{bmatrix} 1.264056 \\ 0.630821 \\ 0.133461 \\ 0.021948 \end{bmatrix}$$

```
plt.figure(6)
for i in insert_degrees:
    calling_function(i, fungsi_awal, hermite_poly, "hermite")
dotted_line = np.linspace(lower_bound, upper_bound, 10)
plt.scatter(dotted_line, fungsi_awal(dotted_line), label=f'Original
    Function {nama_fungsi_awal()}')
plt.title(f'Approximation of {nama_fungsi_awal()} using {hermite_poly
    .__name__} polynomial')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```

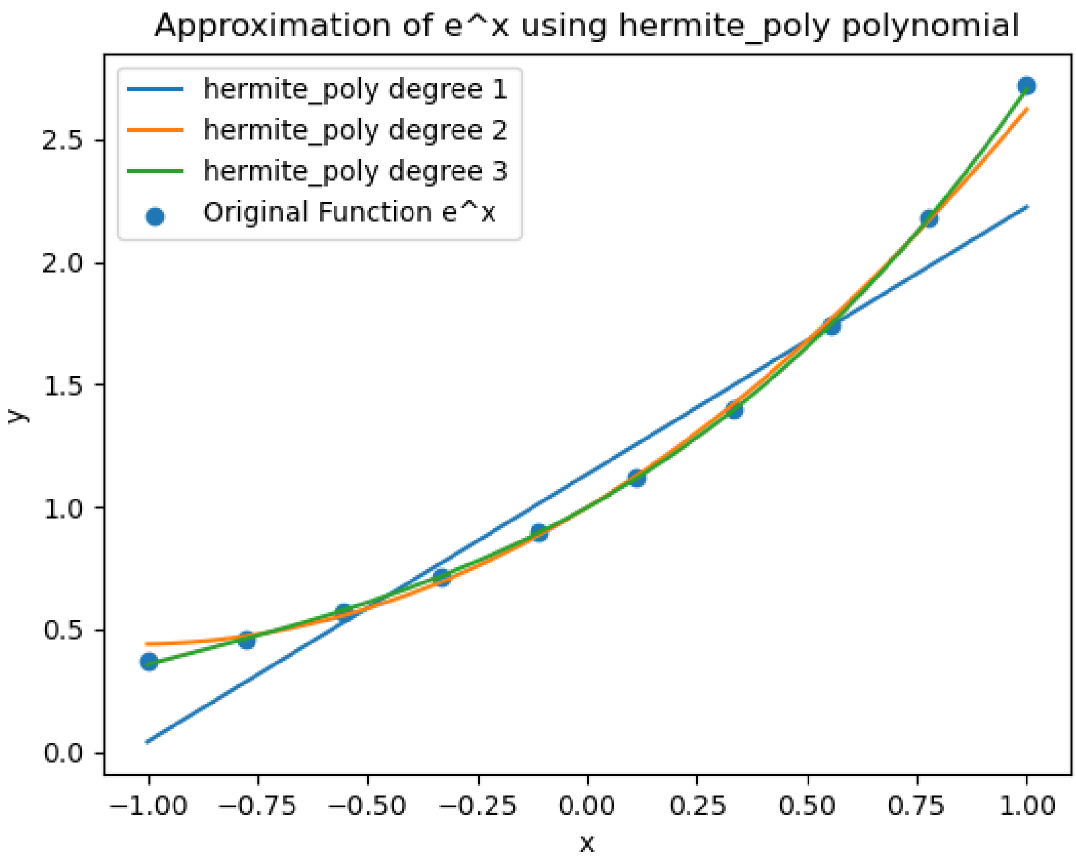

Figure 18: Approximation of e^x using hermite polynomials