ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA EN SISTEMAS

METODOS NUMERICOS ICCD412

Taller 03 series de Taylor y polinomios de Lagrange

JUAN FRANCISCO PINTO ANDRANGO

GR1CC

FECHA DE ENTREGA 11 DE NOVIEMBRE DEL 2025

grafique las curvas de las series de tylor de varios ordenes para los siguientes datos y grafique

-$cos(x), X_0 = 0$

-$(1/x - 1)$, *X_0=0.5*

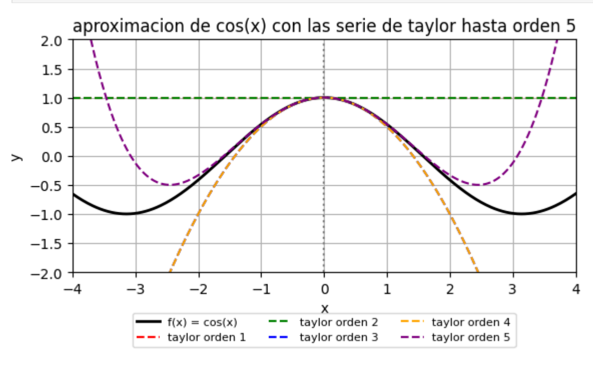
-$ln(x), X_0 = 1$

In [ ]:

```python
# funcion que calcula la serie de tylor de una funcion a partir de  un X_0 hasnta el ornden n

import sympy as sp

def serie_taylor(f, x, x0, orden):

    taylor = f.series(x, x0, orden).removeO()
    return taylor
```

In [19]:

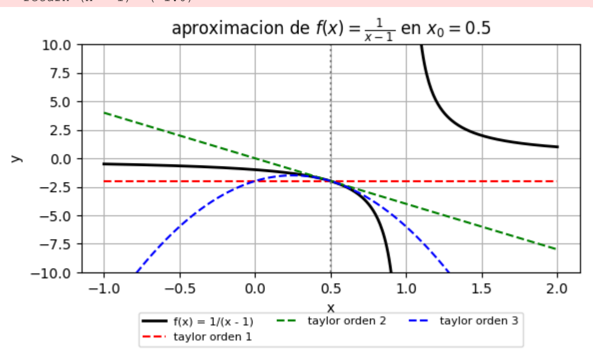
```python
import numpy as np
import matplotlib.pyplot as plt

# funcion para trabajar con el punto de expancion y el orden maximo del cos(x), x_0=0

x = sp.Symbol('x')
f = sp.cos(x)
x0 = 0
orden_max = 5

f_real = sp.lambdify(x, f, modules=['numpy'])
x_vals = np.linspace(-4, 4, 400)
y_real = f_real(x_vals)

color_list = ['red', 'green', 'blue', 'orange', 'purple', 'brown', 'cyan', 'magenta', 'gray', 'olive']

# funcion para generar la grafica

plt.figure(figsize=(6, 3))
plt.plot(x_vals, y_real, label='f(x) = cos(x)', color='black', linewidth=2)

# funcion para calcular y graficar la serie de taylor

for orden in range(1, orden_max + 1):
    taylor_expr = serie_taylor(f, x, x0, orden)
    f_taylor = sp.lambdify(x, taylor_expr, modules=['numpy'])

    y_taylor = f_taylor(x_vals)
    y_taylor = np.asarray(y_taylor)

    if y_taylor.size == 1:
        y_taylor = np.full_like(x_vals, float(y_taylor))
    if np.iscomplexobj(y_taylor):
        y_taylor = y_taylor.real

    color = color_list[(orden - 1) % len(color_list)]
    plt.plot(x_vals, y_taylor, linestyle='--', linewidth=1.5,
             color=color, label=f'taylor orden {orden}')

# configuraiones para el grafico

plt.axvline(x=x0, color='gray', linestyle=':')
plt.ylim(-2, 2)
plt.xlim(-4, 4)
plt.title(f'aproximacion de cos(x) con las serie de taylor hasta orden {orden_max}', fontsize=12)
plt.xlabel('x', fontsize=10)
plt.ylabel('y', fontsize=10)
plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.15), ncol=3, fontsize=8)
plt.grid(True)
plt.tight_layout()
plt.subplots_adjust(bottom=0.1)

plt.show()
```



In [13]:

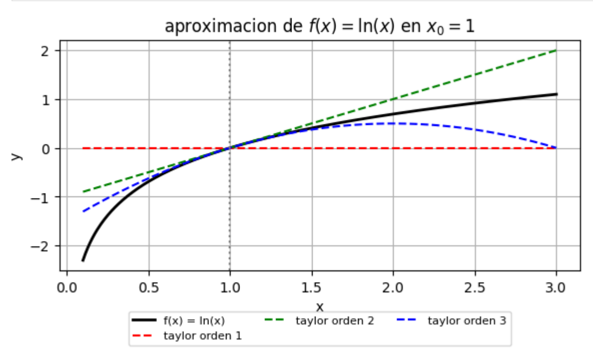
```python
import sympy as sp
import numpy as np
import matplotlib.pyplot as plt

# # funcion para trabajar con el punto de expancion y el orden maximo de la funcion (1/x-1),X_0=0.5

x = sp.Symbol('x')
f = 1 / (x - 1)
x0 = 0.5
orden_max = 3


f_real = sp.lambdify(x, f, modules=['numpy'])
x_vals = np.linspace(-1, 2, 400)
y_real = f_real(x_vals)

# funion para generar la grafica y calculo de las series de tylor

plt.figure(figsize=(6, 3))
plt.plot(x_vals, y_real, label='f(x) = 1/(x - 1)', color='black', linewidth=2)

color_list = ['red', 'green', 'blue', 'orange', 'purple', 'cyan']

for orden in range(1, orden_max + 1):
    taylor_expr = serie_taylor(f, x, x0, orden)
    f_taylor = sp.lambdify(x, taylor_expr, modules=['numpy'])
    y_taylor = f_taylor(x_vals)

    y_taylor = np.asarray(y_taylor)
    if y_taylor.size == 1:
        y_taylor = np.full_like(x_vals, float(y_taylor))
    if np.iscomplexobj(y_taylor):
        y_taylor = y_taylor.real

    color = color_list[(orden - 1) % len(color_list)]
    plt.plot(x_vals, y_taylor, linestyle='--', linewidth=1.5,
             color=color, label=f'taylor orden {orden}')

#Configuraciones para generar la grafica

plt.axvline(x=x0, color='gray', linestyle=':')
plt.title(r'aproximacion de $f(x) = \frac{1}{x - 1}$ en $x_0 = 0.5$', fontsize=12)
plt.xlabel('x', fontsize=10)
plt.ylabel('y', fontsize=10)
plt.ylim(-10, 10)
plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.15), ncol=3, fontsize=8)
plt.grid(True)
plt.tight_layout()
plt.subplots_adjust(bottom=0.1)
plt.show()
```

```
<lambdifygenerated-73>:2: RuntimeWarning: divide by zero encountered in power
  return (x - 1)**(-1.0)
```



In [14]:

```python
import sympy as sp
import numpy as np
import matplotlib.pyplot as plt

# funcion para trabajar con el punto de expancion y el orden maximo de la funcion ln(x),X_0=1

x = sp.Symbol('x')
f = sp.log(x)
x0 = 1
orden_max = 3

f_real = sp.lambdify(x, f, modules=['numpy'])
x_vals = np.linspace(0.1, 3, 400)
y_real = f_real(x_vals)

# funcion para calcular las series de tylor

plt.figure(figsize=(6, 3))
plt.plot(x_vals, y_real, label='f(x) = ln(x)', color='black', linewidth=2)

color_list = ['red', 'green', 'blue', 'orange', 'purple', 'cyan']

for orden in range(1, orden_max + 1):
    taylor_expr = serie_taylor(f, x, x0, orden)
    f_taylor = sp.lambdify(x, taylor_expr, modules=['numpy'])
    y_taylor = f_taylor(x_vals)

    y_taylor = np.asarray(y_taylor)
    if y_taylor.size == 1:
        y_taylor = np.full_like(x_vals, float(y_taylor))
    if np.iscomplexobj(y_taylor):
        y_taylor = y_taylor.real

    color = color_list[(orden - 1) % len(color_list)]
    plt.plot(x_vals, y_taylor, linestyle='--', linewidth=1.5,
             color=color, label=f'taylor orden {orden}')

# configuraciones para hacer la grafica

plt.axvline(x=x0, color='gray', linestyle=':')
plt.title(r'aproximacion de $f(x) = \ln(x)$ en $x_0 = 1$', fontsize=12)
plt.xlabel('x', fontsize=10)
plt.ylabel('y', fontsize=10)
plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.15), ncol=3, fontsize=8)
plt.grid(True)
plt.tight_layout()
plt.subplots_adjust(bottom=0.1)
plt.show()
```



Encuentre el polinomio de Lagrange para los siguientes datos y grafique

1) $(0,0),(30,0.5),(60,3\sqrt{2}),(90,1)$

2) $(1,1),(2,2),(3,2)$

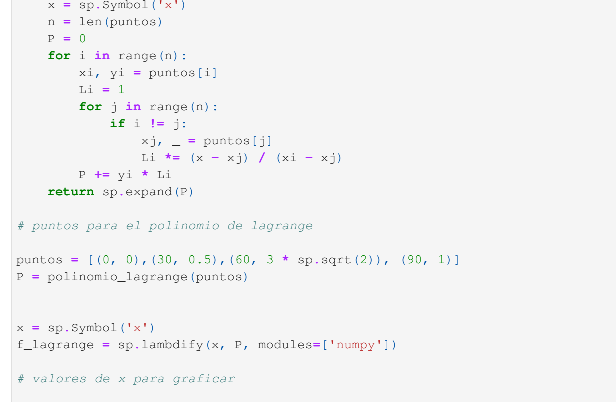
3) $(-2,5),(1,7),(3,11),(7,34)$

In [20]:

```python
# funcion del polinomio de lagrange a partir de una lista de (X,Y)

import sympy as sp

def polinomio_lagrange(puntos):

    x = sp.Symbol('x')
    n = len(puntos)
    P = 0

    for i in range(n):
        xi, yi = puntos[i]
        Li = 1
        for j in range(n):
            if i != j:
                xj, _ = puntos[j]
                Li *= (x - xj) / (xi - xj)
        P += yi * Li

    return sp.expand(P)
```

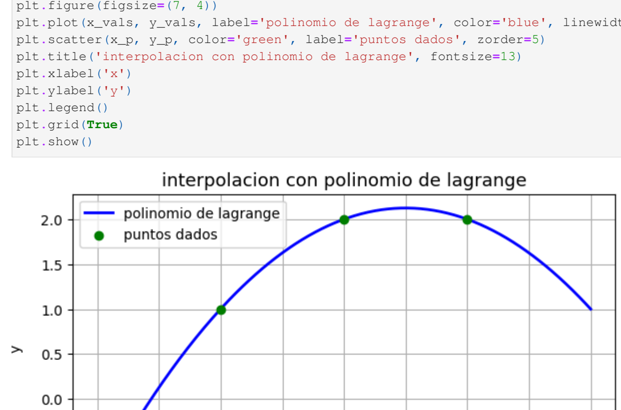
In [30]:

```python
import sympy as sp
import numpy as np
import matplotlib.pyplot as plt

# funcion del polinomio de lagrange a partir de (X_i, Y_i)

def polinomio_lagrange(puntos):
    x = sp.Symbol('x')
    n = len(puntos)
    P = 0
    for i in range(n):
        xi, yi = puntos[i]
        Li = 1
        for j in range(n):
            if i != j:
                xj, _ = puntos[j]
                Li *= (x - xj) / (xi - xj)
        P += yi * Li
    return sp.expand(P)

# puntos para el polinomio de lagrange

puntos = [(0, 0),(30, 0.5),(60, 3 * sp.sqrt(2)), (90, 1)]
P = polinomio_lagrange(puntos)


x = sp.Symbol('x')
f_lagrange = sp.lambdify(x, P, modules=['numpy'])

# valores de x para graficar

x_vals = np.linspace(0, 90, 300)
y_vals = f_lagrange(x_vals)


x_p, y_p = zip(*[(float(px), float(sp.N(py))) for px, py in puntos])

# configuraciones para graficar

plt.figure(figsize=(7, 4))
plt.plot(x_vals, y_vals, label='polinomio de lagrange', color='blue', linewidth=2)
plt.scatter(x_p, y_p, color='black', label='puntos dados', zorder=5)
plt.title('interpolacion con polinomio de lagrange', fontsize=13)
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.grid(True)
plt.show()
```



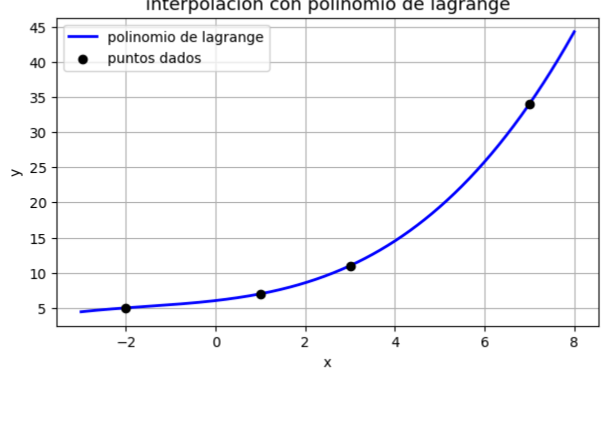
In [28]:

```python
import sympy as sp
import numpy as np
import matplotlib.pyplot as plt

#  funcion del polinomio de lagrange a partir de (X_i, Y_i)

def polinomio_lagrange(puntos):
    x = sp.Symbol('x')
    n = len(puntos)
    P = 0
    for i in range(n):
        xi, yi = puntos[i]
        Li = 1
        for j in range(n):
            if i != j:
                xj, _ = puntos[j]
                Li *= (x - xj) / (xi - xj)
        P += yi * Li
    return sp.expand(P)


puntos = [(1, 1),(2, 2),(3, 2)]

# calcular polinomio de Lagrange
P = polinomio_lagrange(puntos)


x = sp.Symbol('x')
f_lagrange = sp.lambdify(x, P, modules=['numpy'])

# valores de x para graficar

x_vals = np.linspace(0, 4, 300)
y_vals = f_lagrange(x_vals)


x_p, y_p = zip(*[(float(px), float(sp.N(py))) for px, py in puntos])

# configuracion para generar la grafica
plt.figure(figsize=(7, 4))
plt.plot(x_vals, y_vals, label='polinomio de lagrange', color='blue', linewidth=2)
plt.scatter(x_p, y_p, color='green', label='puntos dados', zorder=5)
plt.title('interpolacion con polinomio de lagrange', fontsize=13)
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.grid(True)
plt.show()
```



In [27]:

```python
import sympy as sp
import numpy as np
import matplotlib.pyplot as plt

#funcion del polinomio de lagrange a partir de (X_i, Y_i)

def polinomio_lagrange(puntos):
    x = sp.Symbol('x')
    n = len(puntos)
    P = 0
    for i in range(n):
        xi, yi = puntos[i]
        Li = 1
        for j in range(n):
            if i != j:
                xj, _ = puntos[j]
                Li *= (x - xj) / (xi - xj)
        P += yi * Li
    return sp.expand(P)


puntos = [(-2, 5), (1, 7),(3, 11), (7, 34)]

# calcular polinomio de lagrange

P = polinomio_lagrange(puntos)

x = sp.Symbol('x')
f_lagrange = sp.lambdify(x, P, modules=['numpy'])

# valores de x para graficar
x_vals = np.linspace(-3, 8, 400)
y_vals = f_lagrange(x_vals)

# coordenadas de los puntos dados
x_p, y_p = zip(*[(float(px), float(sp.N(py))) for px, py in puntos])

# configuracion para generar la grafica
plt.figure(figsize=(7, 4))
plt.plot(x_vals, y_vals, label='polinomio de lagrange', color='blue', linewidth=2)
plt.scatter(x_p, y_p, color='black', label='puntos dados', zorder=5)
plt.title('interpolacion con polinomio de lagrange', fontsize=13)
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.grid(True)
plt.show()
```



link del repositorio de git-hub

https://github.com/JuanfranPinto/Metodos-Numericos-