1) Vamos a ver el desarrollo que hemos hecho para la expresión de los polinomios base de Lagrange y del polinomio interpolador de Lagrange. Recordamos que los polinomios base de Lagrange, siendo $\{x_i\}_i$ la secuencia de puntos que nos dan para poder calcularlos, vienen dados por

$$L_i = \prod_{j=0, j\neq i} \frac{x - x_j}{x_i - x_j}$$

se construyen en función de los puntos dados. Ahora, viendo el script en Python para la construcción de dichos polinomios

```
def polinomiosLagrange (list):
    result = []

    for i in range(len(list)):
        l_i =  1
        for j in range(len(list) ):
            if j == i :
                continue

            l_i = l_i*(x-list[j])/(list[i]-list[j])
        result.append(l_i)
    return result
```

definimos el método *polinomiosLagrange* que recibe como parámetro una lista (la secuencia de puntos a usar, i.e, $\{x_i\}_i$) y devuelve una lista, que comprendo los L_i polinomios base, construidos de forma iterativa. Iteramos sobre la lista de puntos (en el primer *for*) y, vamos creando el polinomio i-ésimo con un segundo *for* que utilizamos para, iterar de nuevo sobre dicha lista para hacer el producto definido arriba. Con todo esto, almacenamos en la variable *result* todos los polinomios base.

2) Ahora, vamos a ver la definición usada para construir el polinomio interpolador de Lagrange. Este polinomio, por lo visto en teoría, se construye utilizando los polinomios base y la función aproximar. Veamos la expresión de dicho polinomio:

$$p_n = \sum_{i=0}^{n} L_i\, f(x_i)$$

siendo L_i los polinomios base, f la función a aproximar, $x_i \in \{x_i\}$ el punto i-ésimo que usamos.

Teniendo esta definición, vemos el siguiente método de Python para el cálculo:

```
def polinomiosInterpoladores(list_x,f):
    p_Lagrange = polinomiosLagrange(list_x)
    p_inter_n =0
    for i in range(len(list_x)):
        p_inter_n  += f.subs({'x':list_x[i]})*p_Lagrange[i]
    return p_inter_n
```

polinomiosInterpoladores recibe como parametro *list_ x* y f, siendo *list_ x* la secuencia de puntos $\{x_i\}_i$ y f la función a aproximar. Dentro del método, llamamos a *polinomiosLagrange*, para obtener los polinomios base asociados a dichos puntos. Luego, iteramos sobre *list_ x* para hacer el sumatorio anterior, donde la *f.subs* es un método de la libreria de cálculo simbólico de Python para sustituir cada valor de la lista *list_ x* dentro de f (y obtener así $f(x_i)$) y multiplicarlo por el polinomio base i-ésimo. Devolvemos después el polinomio interpolador, como la suma del producto mencionado.

A) Si aplicamos los programas al caso resuelto en clase, vamos a obtener los siguientes resultados, donde el código sería

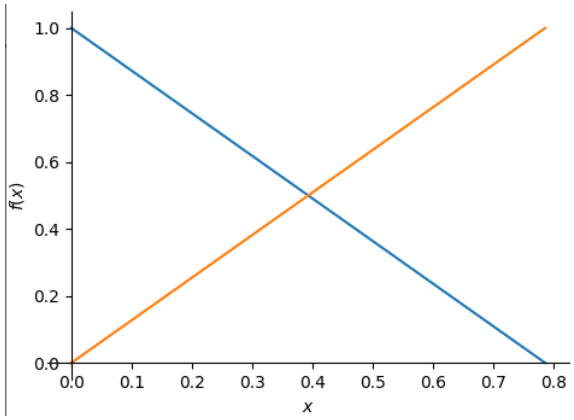

Figura 1.

Esta es la figura correspondiente a los polinomios base del primer ejemplo, de donde obtenemos

$$L_0 = -4\,\frac{x - \pi/4}{\pi} \simeq 1 - 1.27\,x \qquad L_1 = 4\,\frac{x}{\pi} \simeq 1.27\,x$$

con dos puntos, $x_0 = 0$ y $x_1 = \pi/4$, de donde obtenemos la siguiente aproximación

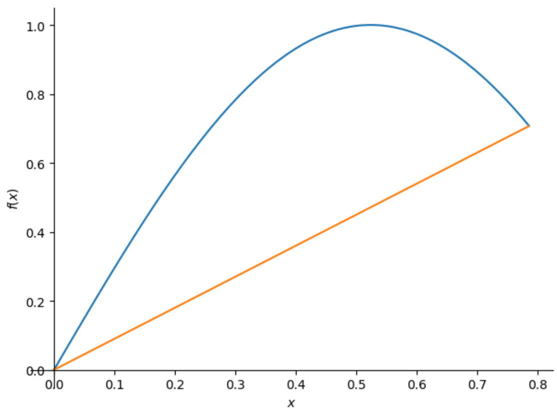

Figura 2.

y, si ahora seguimos el ejercicio y lo hacemos con 3 puntos, los polinomios base nos quedarán

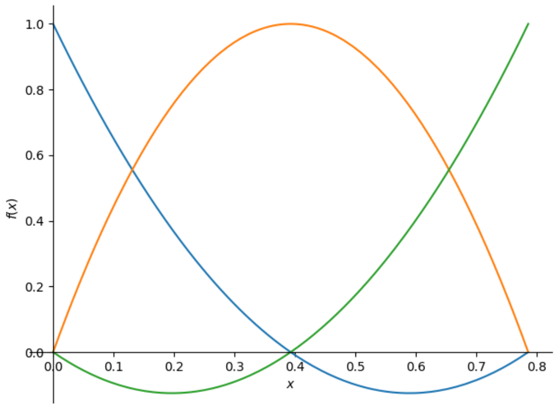

Figura 3.

y la aproximación quedará tal que

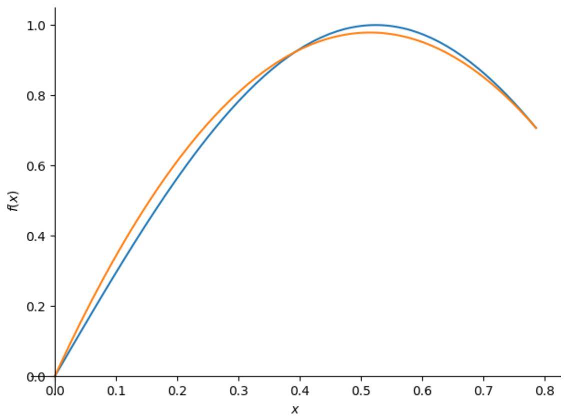

Figura 4.

es decir, obtenemos los mismos resultados que el ejercicio.

B) Ahora, si definimos la función $f(x) = e^{-x} + \cos\left(\frac{4\,x}{\pi}\right)$, tomando el intervalo $[0, 2]$, vamos a diferenciar los casos de

1. 2 puntos

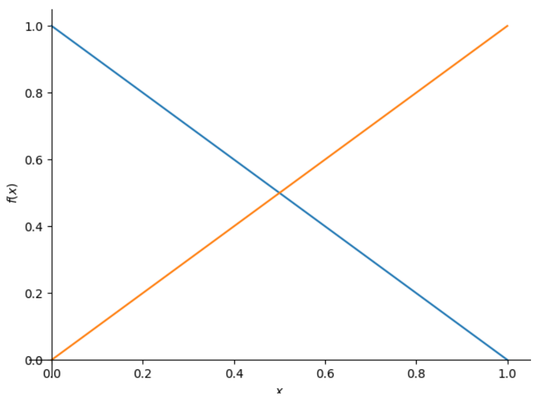

Figura 5.

con $L_0 = 1 - x$ y $L_1 = x$, de donde obtenemos la siguiente aproximación

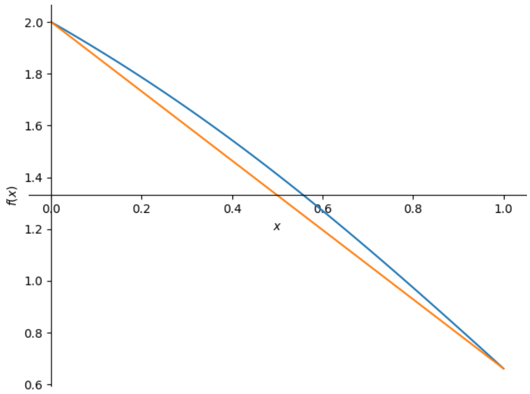

Figura 6.

2. 3 puntos

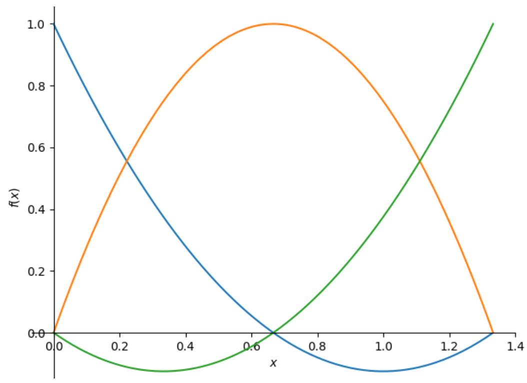

Figura 7.

con $L_0 = -0.75\,(1.0 - 1.5\,x)\,(x - 1.33333333333333)$, $L_1 = -2.25\,x\,(x - 1.33333333333333)$, y $L_2 = 1.125\,x\,(x - 0.666666666666667)$, de donde obtenemos la siguiente aproximación

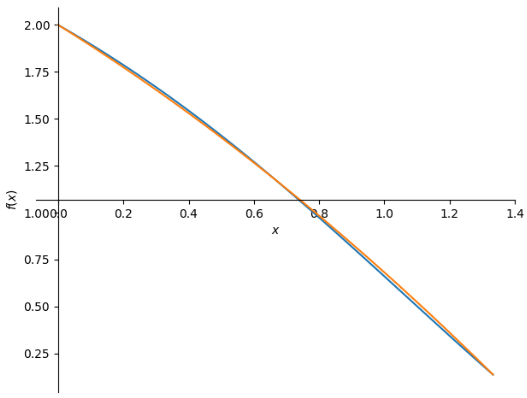

Figura 8.

3. 4 puntos

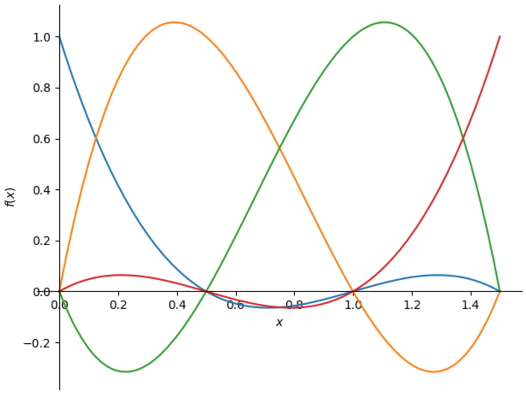

Figura 9.

con $L_0 = 0.666666666666667\,(1.0 - 2.0\,x)\,(x - 1.5)\,(x - 1.0)$, $L_1 = 4.0\,x\,(x - 1.5)\,(x - 1.0)$, $L_3 = -4.0\,x\,(x - 1.5)\,(x - 0.5)$ y $L_4 = 1.33333333333333\,x\,(x - 1.0)\,(x - 0.5)$, de donde la aproximación es

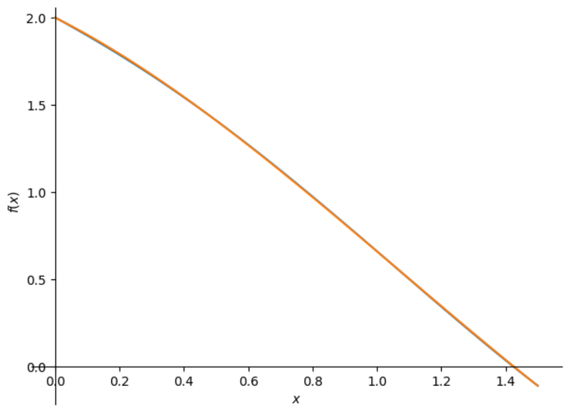

Figura 10.