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

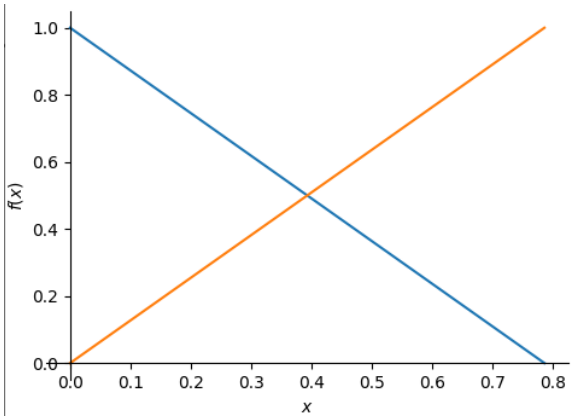

Figura 1.

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

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

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

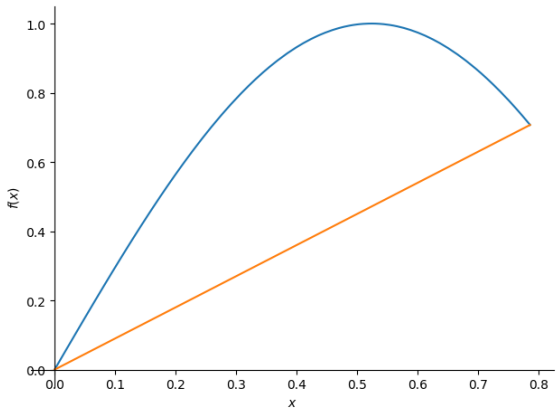

Figura 2.

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

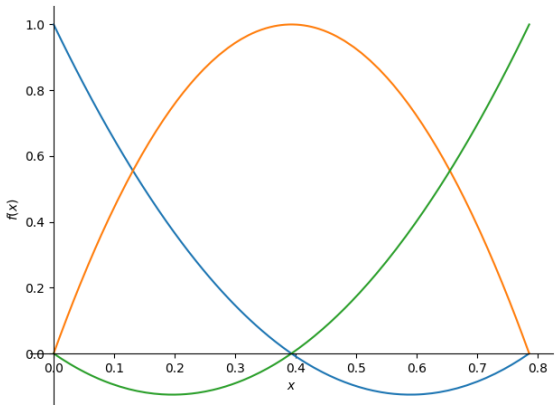

Figura 3.

y la aproximación quedará tal que

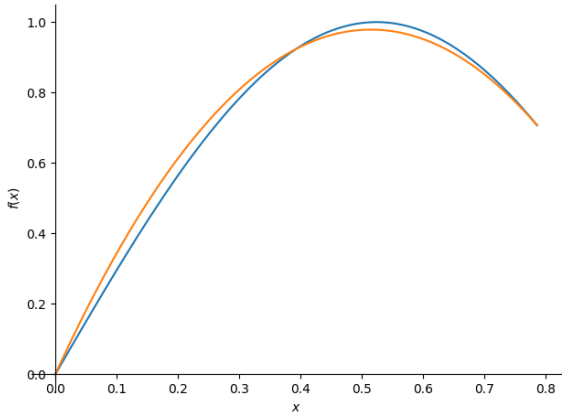

Figura 4.

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

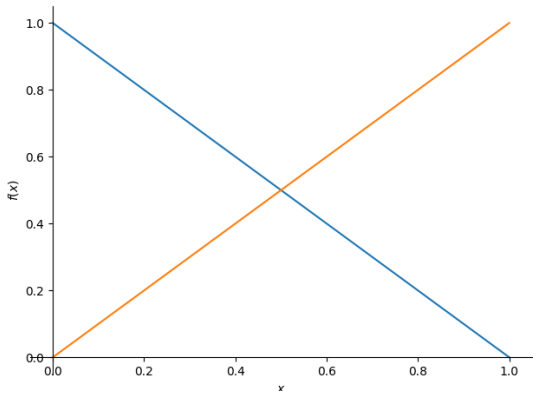

Figura 5.

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

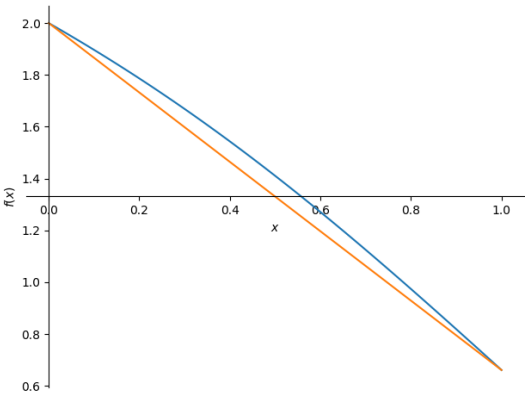

Figura 6.

2. 3 puntos

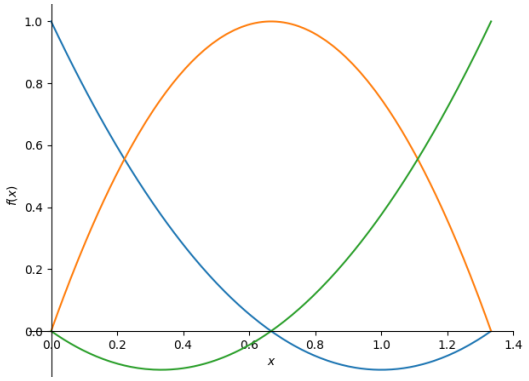

Figura 7.

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

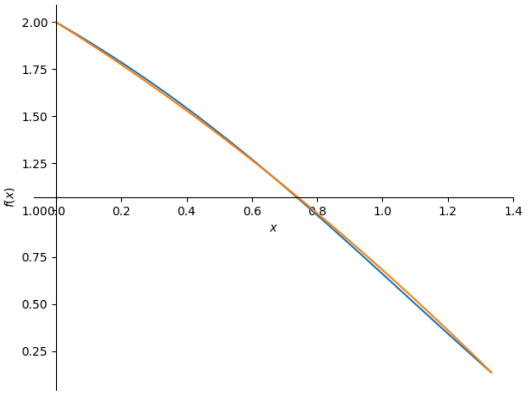

Figura 8.

3. 4 puntos

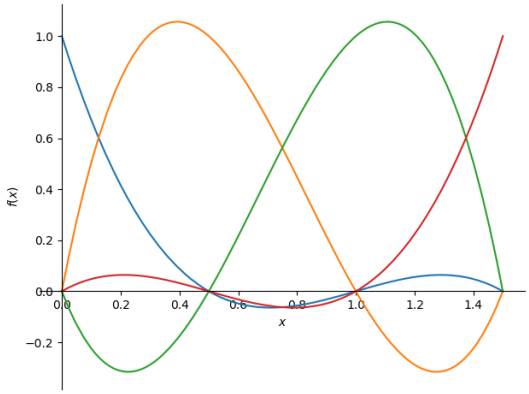

Figura 9.

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

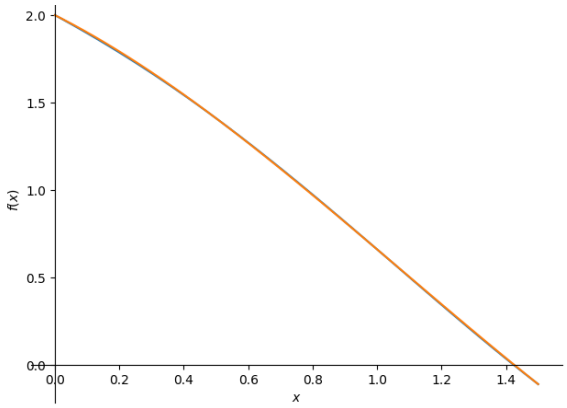

Figura 10.

Integración

1) Sabemos, por lo visto en clase, que a partir de la interpolación vista en el apartado anterior podemos sacar todas las fórmulas con las que aproximamos la integral. Así, las fórmulas del rectángulo son simplemente las fórmulas cuando tenemos 1 punto, y vienen dadas de la siguiente manera

$$p_n(x) = f(a) \to I \simeq \int_a^b p_n(x)\,\mathrm{dx} = f(a)\,(b-a) \quad \text{(extremo izquierdo)}$$

$$p_n(x) = f(b) \to I \simeq \int_a^b p_n(x)\,\mathrm{dx} = f(b)\,(b-a) \qquad \text{(extremo derecho)}$$

$$p_n(x) = f\left(\frac{b+a}{2}\right) \to I \simeq \int_a^b p_n(x)\,\mathrm{dx} = f\left(\frac{b+a}{2}\right)(b-a) \qquad \text{(punto central)}$$

de donde, para calcular dichas fórmulas, podemos usar, en Python, los siguientes métodos:

```
def fRectanguloExtIzdo(f,a,b):
    i = N(integrate(f,(x,a,b)))
    sol = N(f.subs(x,a)*(b-a))
    return sol, abs(sol-i)/i

def fRectanguloExtDcho(f,a,b):
    i = N(integrate(f,(x,a,b)))
    sol = N(f.subs(x,b)*(b-a))
    return [sol, abs(sol - i) / i]

def fRectanguloMedio(f,a,b):
    i = N(integrate(f,(x,a,b)))
    sol = N(f.subs(x,(a+b)/2)*(b-a) * (b - a))
    return [sol, abs(sol - i) / i]
```

En los tres métodos tenemos los mismos parámetros, la función f, y los extremos del intervalo, $[a, b]$.

Respectivamente, hacemos lo siguiente. Calculamos la integral con *integrate(f, (x, a, b))* (la x indica la variable que estamos sustituyendo) con el método definido para eso por la librería de cálculo simbólico de Python, y luego calculamos con $f(a)\,(b-a)$ ($f(b)\,(b-a)$, $f\Big(\frac{b+a}{2}\Big)\,(b-a)$, respectivamente) con *f.subs(x,a)*(b-a)*, que es la solución de nuestra aproximación. Entonces, devolvemos un *array* de dos elementos, en primer lugar la solución que damos con la fórmula del rectángulo, y en segundo el error que cometemos utilizandola, comparado con la integral real.

2) La fórmula del trapecio es la fórmula de Newton-Cotes cuando usamos dos puntos, y, como hemos visto en teoría, viene dada por

$$p_n(x) = f(a) + \frac{f(b)-f(a)}{b-a}(x-a) \to I \simeq \int_a^b p_n(x)\,\mathrm{dx} = (b-a)\,\frac{f(a)+f(b)}{2}$$

y nosotros la hemos implementado con el siguiente método

```
def fTrapecio(f,a,b):
    i = N(numpy.trapz([a, b]))
    sol = N((b-a)*(f.subs(x,a)+f.subs(x,b))/2)
    return sol, abs(sol - i) / i
```

donde la función *numpy.trapz* es la función que calcula la integral con la fórmula del trapecio dada por Python (para comparar con la nuestra y estudiar el error, como el comando *trapz* en matlab) y calculamos nuestra solución simplemente aplicando la fórmula. Devolvemos, también, dos soluciones, la solución (calculada con la fórmula) y el error relativo.

3) Ahora, vamos a utilizar la fórmula de Simpson $1/3$. Es la aplicación de lo obtenido con Newton-Cotes, a 3 puntos. Sabemos de teoría que tiene la integral la siguiente expresión:

$$I \simeq \frac{b-a}{6}\,(f(x_0) + 4\,f(x_1) + f(x_2))$$

```
def fSimpson(f,xi):
    a = N(integrate(f, (x, xi[0], xi[2])))
    D = 6
    c = [1,4,1]
    ret = 0
    for i in range(3):
        ret += c[i]*f.subs(x,xi[i])
    sol = N(ret*(xi[2]-xi[0])/D)
    return sol, abs(sol-a) / a
```

donde primero calculamos la integral dada por Python (*integrate...*) y después, tomando D y los coeficientes visto en clase, calculamos el resultado como la suma (en un *for*, para hacer el sumatorio), y devolvemos la solución (la operación implementada) y el error que nos da respecto a la integral calculada.

A) El problema visto en clase es $\int_0^2 x^3 - 2\,x^2 + 1\,\mathrm{dx}$. Si utilizamos los tres métodos vistos anteriormente, e implementados en Python, la respuesta que obtenemos por pantalla para cada uno es:

- Fórmula rectángulo extremo derecho:

  El resultado de este es $\int_0^2 p_n(x) = 2$

- Fórmula rectángulo extremo izquierdo:

  El resultado de este es $\int_0^2 p_n(x) = 2$

- Fórmula rectángulo punto medio:

  El resultado de este es $\int_0^2 p_n(x) = 0$

- Fórmla trapecio:

  El resultado de este es $\int_0^2 p_n(x) = 2$

- Fórmula Simpson 1/3:

  El resultado de este es $\int_0^2 p_n(x) = 0.666666667$

Y, el resultado de la integral del enunciado es 0.66666666667, luego coincide únicamente con la fórmula de Simpson 1/3, y las otras se alejan bastante del resultado deseado.

B) Vamos a aplicar los programas desarrollados a la integral $\int_0^{2\pi} \cos(x^2 - 1)\,\mathrm{dx}$.

La función que hemos definido tiene esta forma

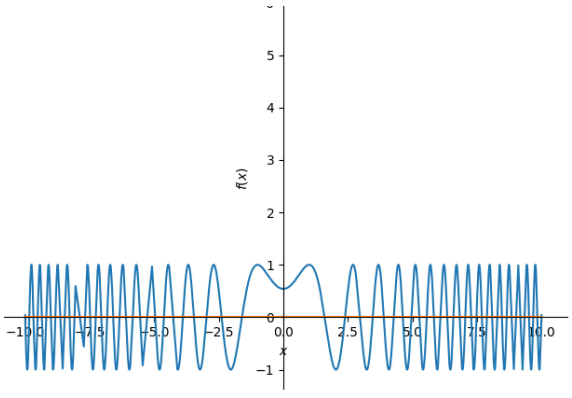

Figura 11.

Y, con las fórmulas definidas anteriormente, obtenemos:

- Fórmula rectángulo extremo derecho:

  El resultado de este es $\int_0^{2\pi} p_n(x) = 4.46986814633572$, y el error calculado da $\approx$3.8528

- Fórmula rectángulo extremo izquierdo:

  El resultado de este es $\int_0^{2\pi} p_n(x) = 3.39481950966595$, y el error calculado da ≈2.6856

- Fórmula rectángulo punto medio:

  El resultado de este es $\int_0^{2\pi} p_n(x) = -5.33950724275201$, y el error calculado da ≈6.7969

- Fórmla trapecio:

  El resultado de este es $\int_0^{2\pi} p_n(x) = 3.93234382800083$, y el error calculado da ≈0.2517

- Fórmula Simpson 1/3:

  El resultado de este es $\int_0^{2\pi} p_n(x) = -2.24889021916773$, y el error calculado da ≈3.4415

Pregunta Optativa) La fómula de Milne viene dada cuando tenemos 5 puntos en Newton-Cotes, y la aproximación de la integral tiene una expresión así

$$I = (b-a)\,\frac{7\,f(a) + 32\,f(x_1) + 12\,f(x_2) + 32\,f(x_3) + 7\,f(b)}{90}$$

Nosotros, en Python, hemos implementado este método para aplicar la fórmula descrita arriba

```
def fMilne(f, xi):
    a = N(integrate(f, (x, xi[0], xi[4])))
    D = 90
    c = [7, 32, 12, 32, 7]
    ret = 0
    for i in range(5):
        ret += c[i] * f.subs(x, xi[i])
    sol = N(ret * (xi[4] - xi[0]) / D)
    return sol, abs(sol - a) / a
```

que tiene un funcionamiento análogo al método usado para la fórmula de Simpson.

Entendemos de este enunciado que, lo que tenemos que hacer, es aplicar Milne a subintervalos y luego sumar las áreas que nos dan estos. Nosotros lo hemos hecho hasta dividiendo N=6 veces, es decir, hasta $2^6 = 64$ subintervalos, y viendo la evolución del error en estos, tratando de aproximar el valor de la integral

$$\int_1^3 (\cos(x) - x\,\mathrm{sen}(x))\,\mathrm{dx}$$

El resultado que hemos obtenido es el siguiente

Iteración 1: solución -3.50997966244149, error -0.000300133227982169

Iteración 2: solución -3.51027565125068, error -0.00000414441879614813

Iteración 3: solución -3.51027973279580, error -6.28736724905821E-8

Iteración 4: solución -3.51027979469422, error -9.75259872859624E-10

Iteración 5: solución -3.51027979565426, error -1.52113877049942E-11

Iteración 6: solución -3.51027979566924, error -2.38031816479634E-13

Donde en cada iteración tenemos $2^{\{\text{iteración}n\}}$ subintervalos.