Tarea 2 - Métodos numéricos
Giovanni Gamaliel López Padilla

Problema 1

Find the fourth Taylor polynomial $P_4(x)$ for the function $f(x) = xe^{x^2}$ about $x_0 = 0$.
Como se sabe que

$$e^x = \sum_{i=0}^{n} \frac{x^i}{i!}$$

entonces

$$\begin{aligned} f(x) &= xe^{x^2} \\ &= x\left(\sum_{i=0}^{n} \frac{(x^2)^i}{i!}\right) \\ &= \sum_{i=0}^{n} x\left(\frac{x^{2i}}{i!}\right) \\ &= \sum_{i=0}^{n} \frac{x^{2i+1}}{i!} \end{aligned}$$

por lo tanto, la función a implementar es:

$$f(x) = \sum_{i=0}^{n} \frac{x^{2i+1}}{i!}$$

Expandiendo esta suma hasta $n = 3$, se obtiene que:

$$\begin{aligned} f(x) &= a_0x + a_1x^3 + a_2x^5 + a_3x^7 \\ &= x(a_0 + x^2(a_1 + x^2(a_2 + a_3x^2))) \end{aligned}$$

por lo que la función recursiva es:

$$\begin{aligned} P_0 &= a_3 \\ P_1 &= a_2 + x^2P_0 \\ P_2 &= a_1 + x^2P_1 \\ P_3 &= x(a_0 + x^2P_2) \end{aligned}$$

1. **Find an upper bound for $|f(x) - P_4(x)|$, for $0 \leq x \leq 0.4$, ie find an upper bound of $|R_4(x)|$ for $0 \leq x \leq 0.4$**

   Los límites superiores que se obtuvieron se encuentran en la tabla 1.

Operación	Límite superior
$\|f(x) - P_4(x)\|$	0.0000003595
$\|R_4(x)\|$	0.0000003499

Tabla 1: Límites superiores para $|f(x) - P_4(x)|$ y $|R_4(x)|$.

2. **Approximate $\int_0^{0.4} f(x)dx$ using $\int_0^{0.4} P_4(x)dx$**

   El resultado de la integral usando el polinomio $P_4(x)$ es de 0.086784.

Con esto se comprueba la precisión que se puede obtener al escribir una función como serie de potencias. El programa de este problema se encuentra en la carpeta Problema_1. El comando para compilarlo es el siguiente:

```
gcc -Wall -o main.out main.c -lm -std=c11
```

El output esperado por la terminal es el siguiente:

```
El resultado de la integral es:     0.086784
El limite superiores encontrados son
    |f(x)-P4(x)|    = 0.0000003595
    |R4(x)|         = 0.0000003499
```

Problema 2

Implement a function to comput

$$f(x) = \frac{1}{\sqrt{x^2+1} - x} \tag{1}$$

When evaluating the previous function we can lose accuracy, transform the right hand side to avoid error (or improve the accuracy). Implement the transformed expression and compare the results with the original function. Note: use values of x greater than 10000.

La función 1 puede provocar problemas con números grandes, ya que el termino $\sqrt{x^2+1}$ puede tener un valor muy cercano a x, provocando asi el obtener el resultado de $1/0$. Racionalizando la función se obtiene lo siguiente:

$$\begin{aligned} f(x) &= \frac{1}{\sqrt{x^2+1} - x} \\ &= \frac{1}{\sqrt{x^2+1} - x}\left(\frac{\sqrt{x^2+1} + x}{\sqrt{x^2+1} + x}\right) \\ &= \frac{\sqrt{x^2+1} + x}{x^2 + 1 - x^2} \\ &= \frac{\sqrt{x^2+1} + x}{1} \\ f(x) &= \sqrt{x^2+1} + x \end{aligned} \tag{2}$$

Se crearon dos funciones donde se calculará $f(x)$ con las ecuaciones 1 y 2, como entrada recibiran un número del tipo double y daran de salida un número de tipo double. Los resultados de cada función con diferentes valores de x se encuentran enlistados en la tabla 2.

x	Ecuación 1	Ecuación 2	Diferencia
1.000000	2.414214	2.414214	0.000000
10.000000	20.049876	20.049876	0.000000
100.000000	200.005000	200.005000	0.000000
1000.000000	2000.000500	2000.000500	0.000000
10000.000000	19999.999778	20000.000050	0.000272
100000.000000	200000.223331	200000.000005	0.223326
1000000.000000	1999984.771129	2000000.000001	15.228871
10000000.000000	19884107.851852	20000000.000000	115892.148148

Tabla 2: Valores de x para las diferences funciones.

El programa de este problema se encuentra en la carpeta Problema_2. El comando para compilarlo es el siguiente:

```
gcc -Wall -o main.out main.c -lm -std=c11
```


```
gcc -Wall -o main.out main.c -lm -std=c11
```


El output esperado del programa es el siguiente:

```
-----------------------------------
Para x      = 1.000000
f(x)        = 2.414214
fr(x)       = 2.414214
RD(x)       = 0.000000
-----------------------------------
Para x      = 10.000000
f(x)        = 20.049876
fr(x)       = 20.049876
RD(x)       = 0.000000
-----------------------------------
Para x      = 100.000000
f(x)        = 200.005000
fr(x)       = 200.005000
RD(x)       = 0.000000
-----------------------------------
Para x      = 1000.000000
f(x)        = 2000.000500
fr(x)       = 2000.000500
RD(x)       = 0.000000
-----------------------------------
Para x      = 10000.000000
f(x)        = 19999.999778
fr(x)       = 20000.000050
RD(x)       = 0.000272
-----------------------------------
Para x      = 100000.000000
f(x)        = 200000.223331
fr(x)       = 200000.000005
RD(x)       = 0.223326
-----------------------------------
Para x      = 1000000.000000
f(x)        = 1999984.771129
fr(x)       = 2000000.000001
RD(x)       = 15.228871
-----------------------------------
Para x      = 10000000.000000
f(x)        = 19884107.851852
fr(x)       = 20000000.000000
RD(x)       = 115892.148148
```

Problema 3

Implement a function to compute the exponential function by using the Taylor/-Maclaurin series

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

Since we cannot add infinite terms, we can approximate this expansion by

$$e^x = \sum_{i=0}^{n} \frac{x^i}{i!}$$

Expandiendo hasta $n = 3$, se obtiene que:

$$\begin{aligned} f(x) &= a_0 + a_1 x + a_2 x^2 + a_3 x^3 \\ &= a_0 + x(a_1 + x(a_2 + x a_3)) \end{aligned}$$

Por lo que podemos obtener una ecuación recursiva de tal manera que:

$$\begin{aligned} P_0 &= a_n \\ P_1 &= a_{n-1} + x P_0 \\ P_2 &= a_{n-2} + x P_1 \\ &\vdots \\ P_i &= a_{n-i} + x P_{i-1} \\ &\vdots \\ P_n &= a_0 + x P_{n-1} \end{aligned}$$

El programa de este problema se encuentra en la carpeta Problema_3. El comando para compilarlo es el siguiente:

```
gcc -Wall -o main.out main.c -lm -std=c11
```

El output esperado del programa es el siguiente:

Caso de datos de prueba

```
Deseas usar el programa con los datos de prueba?(Y/n): y
f(2.000000) = 7.388995
```

Caso de datos de introducidos por usuario

```
Deseas usar el programa con los datos de prueba?(Y/n): n
Escribe el numero de terminos de la serie que quieres calcular:
4
Escribe el numero de x que quieres evaluar:
-2
f(-2.000000) = 0.333333
```

Problema 4

Riemann sums can be used to estimate the area under the curve $y = f(x)$ in the interval $[a, b]$. Left-and right-endpoint approximations, with subintervals of the same width, are special kinds of Riemann sums, ie,

Left-endpoint approximation:

$$L_n = \sum_{i=1}^{n} f(x_{i-1})\Delta x$$

Right-endpoint approximation:

$$R_n = \sum_{i=1}^{n} f(x_i)\Delta x$$

$$\textbf{where } \Delta x = \frac{b-a}{n} \qquad x_i = a + i\Delta x,\ i = 0, 1, \cdots, n$$

Implement functions to compute L_n and R_n . Using the function $f(x) = sinx$ over the interval $[0, \frac{\pi}{2}]$, compute L_n and R_n for n=10. Compare the previous results with

$$\int_0^{\frac{\pi}{2}} f(x)dx$$

El programa que emplea los dos algoritmos se encuentra en la carpeta Problema_4. Los resultados de cada integral se muestran en la tabla 3. En la figura 1 se muestra la representación gráfica de los algoritmos Left-endpoint y Right-endpoint.

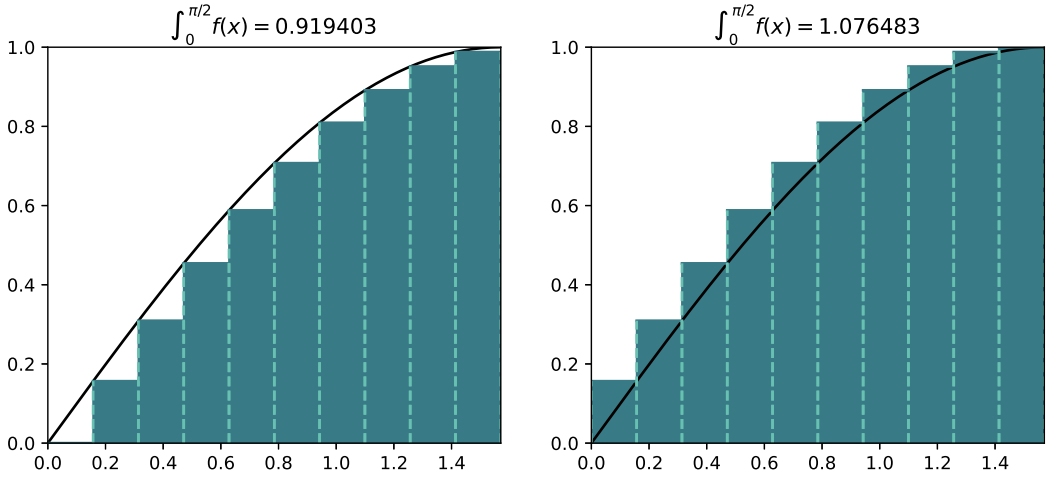

Figura 1: Representación gráfica de los algoritmos left-endpoint (izquierda) y right-endpoint (derecha).

Algoritmo	Resultado
Left-endpoint	0.919403
Right-endpoint	1.076483

Tabla 3: Resultados de los algoritmos de integración con los parámetros especificados para $f(x) = sin(x)$

La diferencia obtenida entre los dos algoritmos de integración es de 0.157080.
El programa de este problema se encuentra en la carpeta Problema_4. El comando para compilarlo es el siguiente:

```
1    gcc -Wall -o main.out main.c -lm -std=c11
```

El output esperado del programa es el siguiente:

```
    Resultado de las integrales:
        Left-endpoint:  0.919403
        Right-endpoint: 1.076483
        Diferencia:     0.157080
```