Tarea 6. Física Computacional.

Gustavo Hernández Angeles. gustavoha40@gmail.com

I. PRIMER PROBLEMA.

Aproximar las siguientes funciones usando un polinomio de Taylor de grado 7, mostrando las gráficas de todos los polinomios. NOTA: En algunas series se usan los números de Bernoulli, así que implementar su obtención dentro del programa.

CONDICIONES:

a) El programa debe tener un menú donde se pueda elegir entre cada una de las series.
b) Agregar al reporte la deducción de la expresión de cada serie (a mano).
c) El programa al ejecutarse debe generar el script de GNUPLOT.

1. $e^x \cos(x)$
2. $\frac{1}{1+9x^2}$
3. $\frac{x}{e^x-1}$
4. $e^{\sin(x)}$
5. e^{-x^2}
6. $\log_{10}(1+e^x)$
7. $\ln\cos(x)$
8. $\frac{1}{2}(e^x - e^{-x})$
9. $\cos^2 x$
10. $\ln\sqrt{1+x}$

DESARROLLO

El programa realizado para este problema se basa en un programa general para las series de Taylor de cualquier función dada utilizando el programa de la derivada n-ésima. Es decir, no ocupamos explícitamente la expresión de cada función en términos de una suma de potencias ya que el proceso es puramente numérico y funciona, al menos, hasta orden 8. Aun así, las expresiones de los polinomios hasta el grado 7 se deducen analíticamente.

El programa utilizará comandos en la terminal, los comandos disponibles en la terminal dependen del sistema operativo que se esté utilizando, por lo que al inicio nuestro programa nos solicita el sistema operativo que estamos utilizando. La selección se hace mediante el siguiente bloque de código:

```
    write(*,*)'Que sistema operativo tiene?'
    write(*,*)'1. Linux o MacOS'
    write(*,*)'2. Windows'
2    write(*,'(a)',advance='no')'Seleccion: ';read*, sistemaoperativo
    if (sistemaoperativo .lt. 1 .or. sistemaoperativo .gt.2) then
        print*,'ERROR, seleccione 1 o 2.'
        goto 2
    end if
```

Se realiza un menú mediante un listado de las funciones que se van a aproximar mediante el polinomio de Taylor, su forma en código es la siguiente:

```
    ! Menú
    write(*,*)'Listado de funciones:'
    write(*,*)'1.   exp(x)*cos(x)'
    write(*,*)'2.   1/(1+9*x**2)'
    write(*,*)'3.   x/(exp(x)-1)'
    write(*,*)'4.   exp(sin(x))'
    write(*,*)'5.   exp(-x**2)'
    write(*,*)'6.   log10(1+exp(x))'
    write(*,*)'7.   log(cos(x))'
    write(*,*)'8.   (exp(x)-exp(-x))/2'
    write(*,*)'9.   cos(x)**2'
    write(*,*)'10.  log(sqrt(1+x))'
1    write(*,'(a)',advance='no')'Seleccion: ';read*,k
    if (k .lt. 1 .or. k .gt.10) then
        print*,'ERROR, seleccione una funcion entre 1 y 10.'
        goto 1
    end if
```

Se desplegará una lista con cada una de las funciones asignándoles una etiqueta numérica a cada una, esta etiqueta será guardada en la variable entera k. Si el valor de k recibido está fuera del rango de la lista nos mostrará un mensaje de error y volverá a requerir de la lectura de k.

```
! Función seleccionada
select case (k)
case (1)
    funcion = 'exp(x)*cos(x)'
case (2)
    funcion = '1/(1+9*x**2)'
case (3)
    funcion = 'x/(exp(x)-1)'
case (4)
    funcion = 'exp(sin(x))'
case (5)
    funcion = 'exp(-x**2)'
case (6)
    funcion = 'log10(1+exp(x))'
case (7)
    funcion = 'log(cos(x))'
case (8)
    funcion = '(exp(x)-exp(-x))/2'
case (9)
    funcion = 'cos(x)**2'
case (10)
    funcion = 'log(sqrt(1+x))'
end select
```

Dependiendo del valor de la etiqueta k, se le asignará un valor de cadena a la variable FUNCION, esta cadena es precisamente la función a la cual se aproximará mediante serie de Taylor.

```
! Ejecuta el programa de Taylor con la función seleccionada
call funcionyejecutar(nombre,funcion,sistemaoperativo)
```

Es aquí donde comienza lo interesante, la variable NOMBRE en nuestro programa tiene el valor de la cadena ‘serietaylor’. Lo que hace esta subrutina llamada FUNCIONYEJECUTAR es, valga la redundancia, añadir la función al archivo del código (que lleva por nombre, en nuestro caso, serietaylor.f90) que vamos a utilizar para después ejecutarlo, su forma es la siguiente:

```
subroutine funcionyejecutar(nombre,funcion,sistemaoperativo)
    implicit none
    character(1000) :: nombre, funcion
    integer :: sistemaoperativo
    ! Copia el código en otro archivo donde se insertará la función
    if (sistemaoperativo.eq.2) call execute_command_line('copy '//trim(nombre)//'.f90 '//trim(nombre)//'_mod.f90')
    if (sistemaoperativo.eq.1) call execute_command_line('cp '//trim(nombre)//'.f90 '//trim(nombre)//'_mod.f90') !L
    open(10,file=trim(nombre)//'_mod.f90',access = 'append')
    write(10,*)'real*16 function f(x)'
    write(10,*)'real*16 :: x'
    write(10,*)'f = '//trim(funcion)
    write(10,*)'return'
    write(10,*)'end function'
    close(10)
    ! Se compila el código con la función y se ejecuta
    if (sistemaoperativo.eq.2) then
        call system('gfortran '//trim(nombre)//'_mod.f90 -o '//trim(nombre)//'.exe')
        call system(trim(nombre)//'.exe')
    else
        call system('gfortran '//trim(nombre)//'_mod.f90 -o '//trim(nombre)//'.out')
        call system('./'//trim(nombre)//'.out')
    endif
end subroutine
```

Esta subrutina realiza una copia del archivo original que no posee el código y le otorga el nombre ‘nombre_mod.f90’, después abre esa copia con el atributo open(.., access = ‘append’) ya que va a insertar al final del código la función que introdujimos con la variable ‘funcion’. Se pueden leer muchas líneas de código que constituyen únicamente el formato para que Fortran lo lea. Además, también compilará y ejecutará el código que acabamos de copiar, haciendo necesario compilar solamente un programa a la vez y habilita la posibilidad de correr el programa en cuestión con cualquier función que deseemos.

Ahora, ¿cómo funciona el programa para calcular la serie de Taylor?

```
program serietaylor
    implicit none
    real*16 :: a,xf,xi,h=0.01,serief
    integer :: n
    character(1000) :: serie='datos.dat', plot = 'serietaylor.gpl', xin, xfin
```

Para inicializar, se definen las variables que se utilizarán en todo el programa: ‘a’ será el punto alrededor de donde se haga la expansión en series de Taylor, ‘xf’ y ‘xi’ se refieren a los extremos del intervalo que se graficarán mediante GNUPLOT, el archivo de los datos se lee como la variable ‘serie = datos.dat’, ‘n’ será el orden de la expansión de serie de Taylor, las variables ‘plot’, ‘xin’,’xfin’ serán encargadas para darle formato al ejecutable de gnuplot que nos proporcionará la gráfica del polinomio calculado numéricamente.

```
! Se definen los parámetros para la expansión en serie de taylor

write(*,'(a)',advance='no')'Orden de la expansion: ';read*,n
write(*,'(a)',advance='no')'Alrededor del punto: ';read*,a
write(*,'(a)',advance='no')'Extremos del intervalo a graficar: ';read*,xi,xf
write(xin,'(f8.3)')xi
write(xfin,'(f8.3)')xf
```

Acto seguido, procede a escribir en el archivo de datos.dat los puntos del polinomio, escribiendo en cada renglón el valor de x y el valor del polinomio evaluado en tal x (que lleva por nombre serief), son escritas como primera y segunda columna respectivamente. Después se genera el script en gnuplot en el archivo 'serietaylor.gpl' con un formato. Este archivo contienen la línea 'plot "datos.dat" …' el cual establece que se graficarán los puntos que guardamos en el paso anterior.

```
    ! Escribe en un archivo los puntos de la expansión en serie

    open(10,file=serie)
    do while (xi.lt.xf)
        write(10,*)xi,serief(n,xi,a)
        xi = xi+h
    enddo
    close(10)

    ! Genera el script en gnuplot con el nombre serietaylor.gpl

    open(11,file=plot)
    write(11,*)'set xrange ['//trim(xin)//':'//trim(xfin)//']'
    write(11,*)'set mxtics 4'
    write(11,*)'set mytics 4'
    write(11,*)'set zeroaxis lt -1 lc -1'
    write(11,*)'plot "datos.dat" u 1:2 w l lw 2 t "Expansion en serie"'
    close(11)

end
```

La función 'serief(n,xi,a)' nos regresa el polinomio de Taylor de orden n, evaluado en el punto xi, haciendo la expansión alrededor de 'a'. El siguiente bloque de código nos muestra la función:

```
! Expansión en serie de taylor, requiere de la subrutina de nderivada
function serief(n,x,a)
    implicit none
    real*16 :: x,serief,nderivada, a
    integer :: n, i
    serief = 0
    do i = 0, n
        call nder(a,i,nderivada)
        serief = serief + (nderivada/gamma(real(i+1)))*(x-a)**i
    enddo
    return
end function

! Nderivada, cuadruple precisión
subroutine nder(x,n,resultado)
    implicit none
    integer :: n, k
    real*16 :: f, x, sum, resultado, combination
    real*16, parameter :: h=1.0e-4
    sum=0
    do 1 k=0,n
        combination = gamma(real(n+1))/(gamma(real(n-k+1))*gamma(real(k+1)))
        sum = sum + (-1)**(k) * combination * f(x+(n-2*k)*h)
    1 continue
    resultado = sum / ((2*h)**n)
    return
end subroutine
```

Como podemos observar, se trata de la definición de la expansión en serie de Taylor.

$$f(x) = \sum_{n=0}^{\infty} \frac{f^n(a)}{n!}(x-a)^n$$

Se utiliza la función intrínseca GAMMA para determinar el factorial, cuyo argumento se requiere real. La expresión para la n-derivada con diferencias centradas se dedujo en una tarea anterior.

```
! Esto es adicional para este problema, se inserta una línea en el .gpl del programa de Taylor
! para graficar también la función que se expande en series
open(11,file=trim(nombre)//'.gpl', access = 'append')
write(11,*)'replot '//trim(funcion)//' lc 7'
close(11)
!call system('del '//trim(nombre)//'_1.f90') ! Utilizado para borrar la copia del código.
call system(trim(nombre)//'.gpl') ! Se ejecuta la gráfica.
```

Para finalizar, se le añade al .gpl de la serie de Taylor la función que se expandió en series de Taylor y se ejecuta, completando así la ejecución del programa.

NOTA 1

En mi deducción a la solución del problema, no requerí de ninguna forma los números de Bernoulli. Sin embargo, decidí programarlos aún así para poder completar la ponderación. Dentro de la literatura consultada pude recoger la siguiente fórmula recursiva:

$$B_m = -m! \sum_{k=0}^{m-1} \frac{B_k}{(m+1-k)!\,k!}$$

Por lo que su implementación a Fortran se realiza de forma directa mediante una función recursiva de doble precisión:

```
real*8 recursive function bernoulli(n) result (bern)
    implicit none
    integer :: n, k
    real*8 :: sum
    if (n.eq.0) then
        bern = 1
    else
        sum = 0
        do k = 0, n-1
            sum = sum + bernoulli(k)/(gamma(real(k+1))*gamma(real(n+2-k)))
        end do
        bern = -gamma(real(n+1))*sum
    end if
    return
end function
```

Con el objetivo de entender más el proceso del código, lo ejecutamos con el ejercicio 1.

NOTA 2

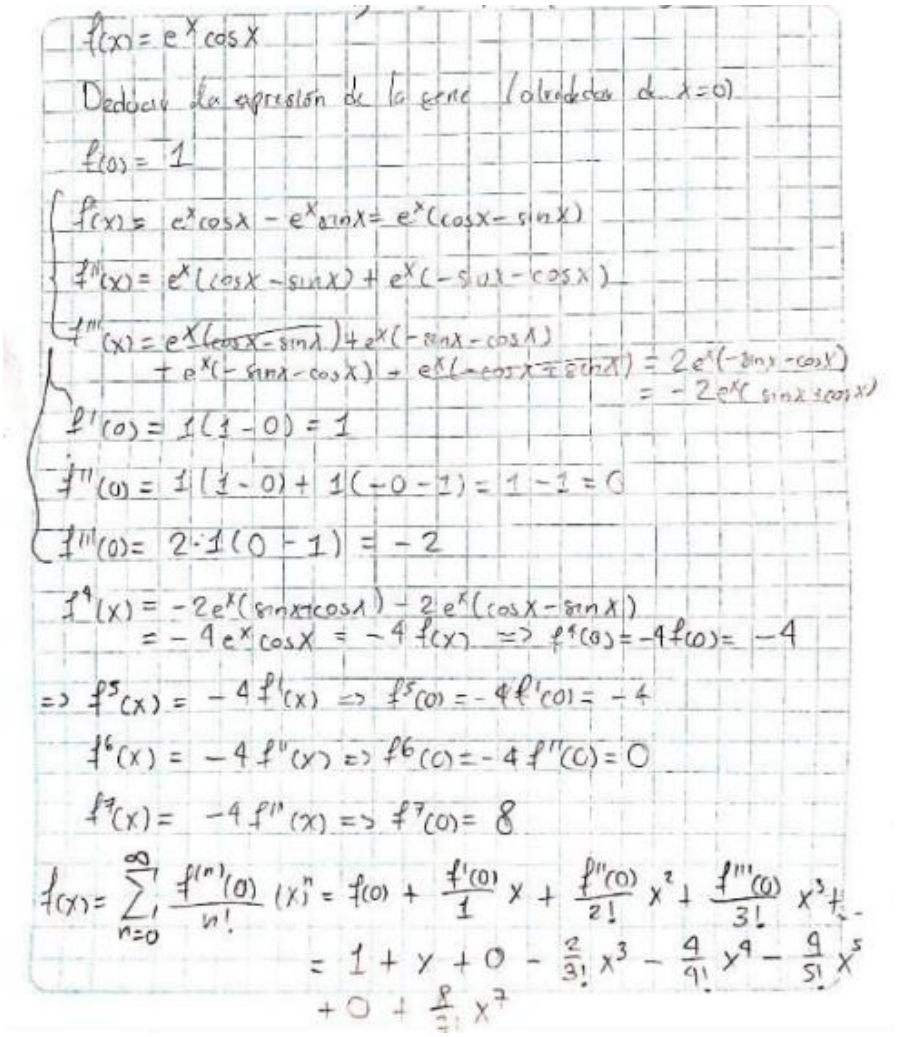
Probando el programa en un equipo con sistema operativo Linux (gracias a Diego) con la distribución Ubuntu, nos da resultados numéricos muy distintos a los que tengo en Windows. No sé exactamente a qué se deba esto, ya que los comandos de la consola funcionan perfectamente bien, son los datos numéricos que, aún con las mismas entradas, difieren mucho uno de otro. No sé si este error se experimente también en MacOS.

1. $e^x \cos(x)$

En este primer ejercicio deduje el polinomio de Taylor de grado 7 alrededor de x = 0 analíticamente, durante el proceso llegué a una ecuación diferencial un tanto curiosa. Satisfecha por la función $f(x) = e^x \cos x$:

$$\frac{d^4 f(x)}{dx^4} = -4f(x)$$

Por lo que evaluar las derivadas en x = 0 se volvió sustancialmente más sencillo. El proceso a mano es el siguiente:



De esta forma, la función $e^x \cos x$ se puede aproximar de la siguiente forma:

$$e^x \cos x \approx 1 + x - \frac{1}{3}x^3 - \frac{1}{6}x^4 - \frac{4}{5!}x^5 + \frac{8}{7}x^7$$

En la ejecución del programa, al principio se nos solicita introducir el sistema operativo que se maneja en nuestra computadora, esta selección se despliega de la siguiente forma:

```
Que sistema operativo tiene?
1. Linux o MacOS
2. Windows
Seleccion: 2
```

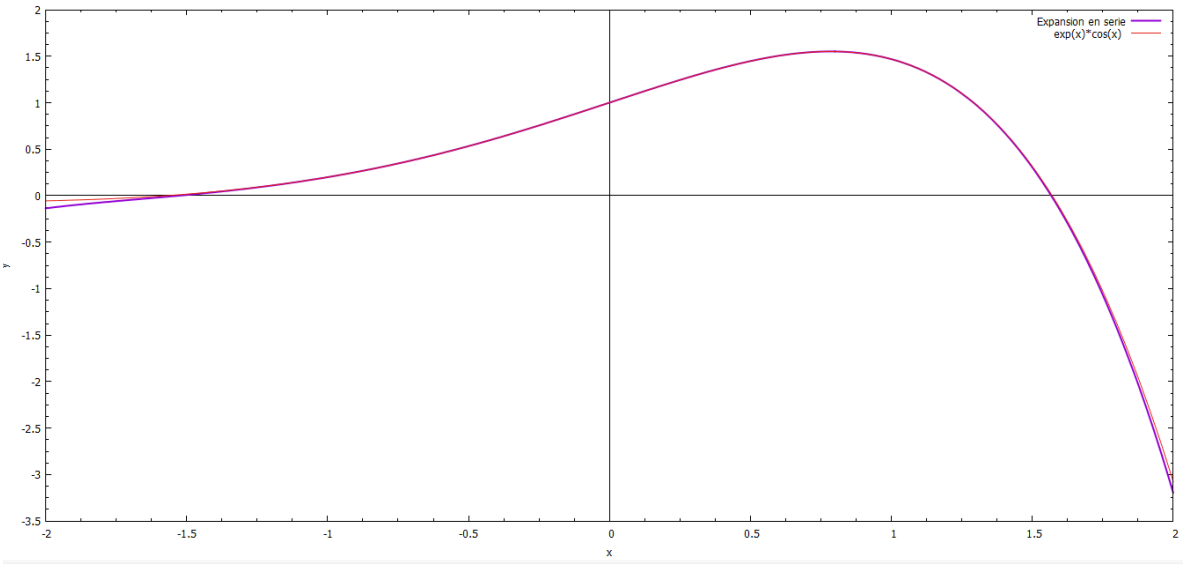
Se nos despliega un menú de lista donde seleccionaremos la función que querremos expandir en series, en este caso, es la opción 1.

```
 Listado de funciones:
 1.   exp(x)*cos(x)
 2.   1/(1+9*x**2)
 3.   x/(exp(x)-1)
 4.   exp(sin(x))
 5.   exp(-x**2)
 6.   log10(1+exp(x))
 7.   log(cos(x))
 8.   (exp(x)-exp(-x))/2
 9.   cos(x)**2
 10.  log(sqrt(1+x))
Seleccion: 1
```

Una vez que realizamos la elección de la función a expandir, el programa crea una copia del código de serietaylor.f90 donde se inserta la función que elegimos y lo ejecuta. En la terminal de Windows este procedimiento se ve con un mensaje de confirmación "1 archivo(s) copiado(s)". El programa de la serie de Taylor nos requiere declarar el orden de la expansión que deseamos, el punto alrededor y el intervalo que se va a graficar.

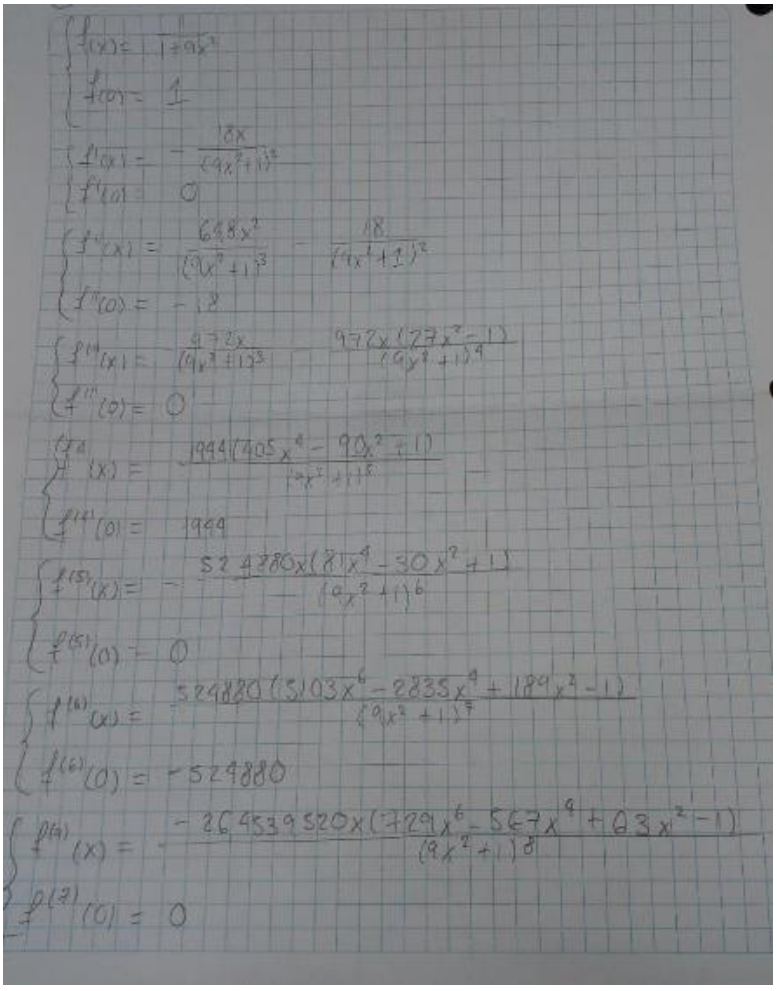
```
 4.   exp(sin(x))
 5.   exp(-x**2)
 6.   log10(1+exp(x))
 7.   log(cos(x))
 8.   (exp(x)-exp(-x))/2
 9.   cos(x)**2
 10.  log(sqrt(1+x))
Seleccion: 1
        1 archivo(s) copiado(s).
Orden de la expansion: 7
Alrededor del punto: 0
Extremos del intervalo a graficar: -2,2
```

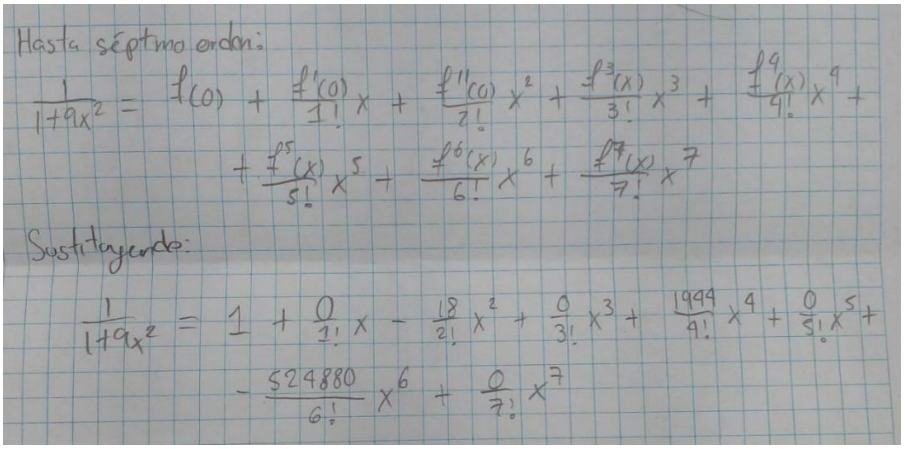
Una vez declarados los valores requeridos, se genera la siguiente gráfica en gnuplot:



2. $\frac{1}{1+9x^2}$

Para expandir esta función en Series de Taylor, necesitaremos de las primeras 7 derivadas de la función evaluadas en el punto donde se expande, en nuestro caso podemos ocupar x = 0. Así, analíticamente.





De esta forma, nuestra expansión a orden 7 de la función $\frac{1}{1+9x^2}$ nos resulta:

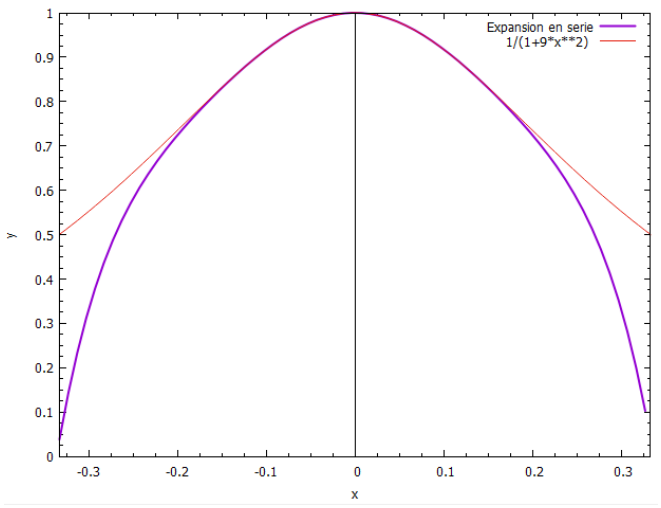
$$\frac{1}{1+9x^2} \approx 1 - 9x^2 + 81x^4 - 729x^6$$

En la ejecución del programa, nuestro menú nos mostrará como opción 2 esta misma función.

```
 Listado de funciones:
 1.   exp(x)*cos(x)
 2.   1/(1+9*x**2)
 3.   x/(exp(x)-1)
 4.   exp(sin(x))
 5.   exp(-x**2)
 6.   log10(1+exp(x))
 7.   log(cos(x))
 8.   (exp(x)-exp(-x))/2
 9.   cos(x)**2
 10.  log(sqrt(1+x))
Seleccion: 2
```

Especificando el punto alrededor de donde se realiza la expansión, el orden de la expansión y en qué intervalo se va a graficar, nos mostrará la siguiente gráfica.

```
 Listado de funciones:
 1.   exp(x)*cos(x)
 2.   1/(1+9*x**2)
 3.   x/(exp(x)-1)
 4.   exp(sin(x))
 5.   exp(-x**2)
 6.   log10(1+exp(x))
 7.   log(cos(x))
 8.   (exp(x)-exp(-x))/2
 9.   cos(x)**2
 10.  log(sqrt(1+x))
Seleccion: 2
       1 archivo(s) copiado(s).
Orden de la expansion: 7
Alrededor del punto: 0
Extremos del intervalo a graficar: -0.333,0.333
```



3. $\frac{x}{e^x-1}$

De la literatura (Wikipedia) podemos observar que precisamente esta función es la función generatriz de los números de Bernoulli B_n, dados por la siguiente definición.

$$G(x) = \frac{x}{e^x - 1} = \sum_{i=0}^{\infty} B_n \frac{x^n}{n!}$$

Por lo que nuestra expansión a orden 7 de la función $\frac{x}{e^x-1}$ resulta en la siguiente expresión:

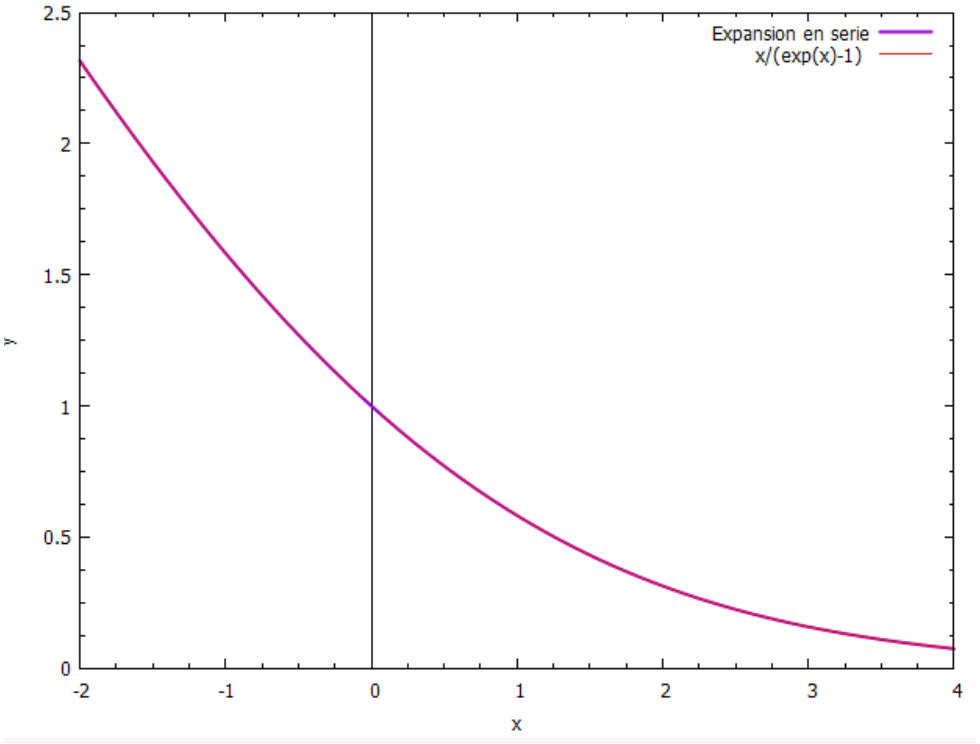
$$\frac{x}{e^x - 1} \approx 1 - \frac{1}{2}x + \frac{\frac{1}{6}x^2}{2!} - \frac{\frac{1}{30}x^4}{4!} + \frac{\frac{1}{42}x^6}{6!} + 0$$

En la ejecución del programa, nos mostrará en el menú como opción 3 esta misma función.

```
Listado de funciones:
1.   exp(x)*cos(x)
2.   1/(1+9*x**2)
3.   x/(exp(x)-1)
4.   exp(sin(x))
5.   exp(-x**2)
6.   log10(1+exp(x))
7.   log(cos(x))
8.   (exp(x)-exp(-x))/2
9.   cos(x)**2
10.  log(sqrt(1+x))
Seleccion: 3
```

Especificando el punto alrededor de donde se realiza la expansión, el orden de la expansión y en qué intervalo se va a graficar, nos mostrará la siguiente gráfica. Ha de aclararse que en esta función se decide expandir alrededor del 1 ya que existe un punto donde la computadora intenta realizar la operación de división por 0. Por lo que se decide expandir numéricamente alrededor del 1.

```
Orden de la expansion: 7
Alrededor del punto: 1
Extremos del intervalo a graficar: -2,4
```



Se prueba también que funciona el código para los números de Bernoulli:

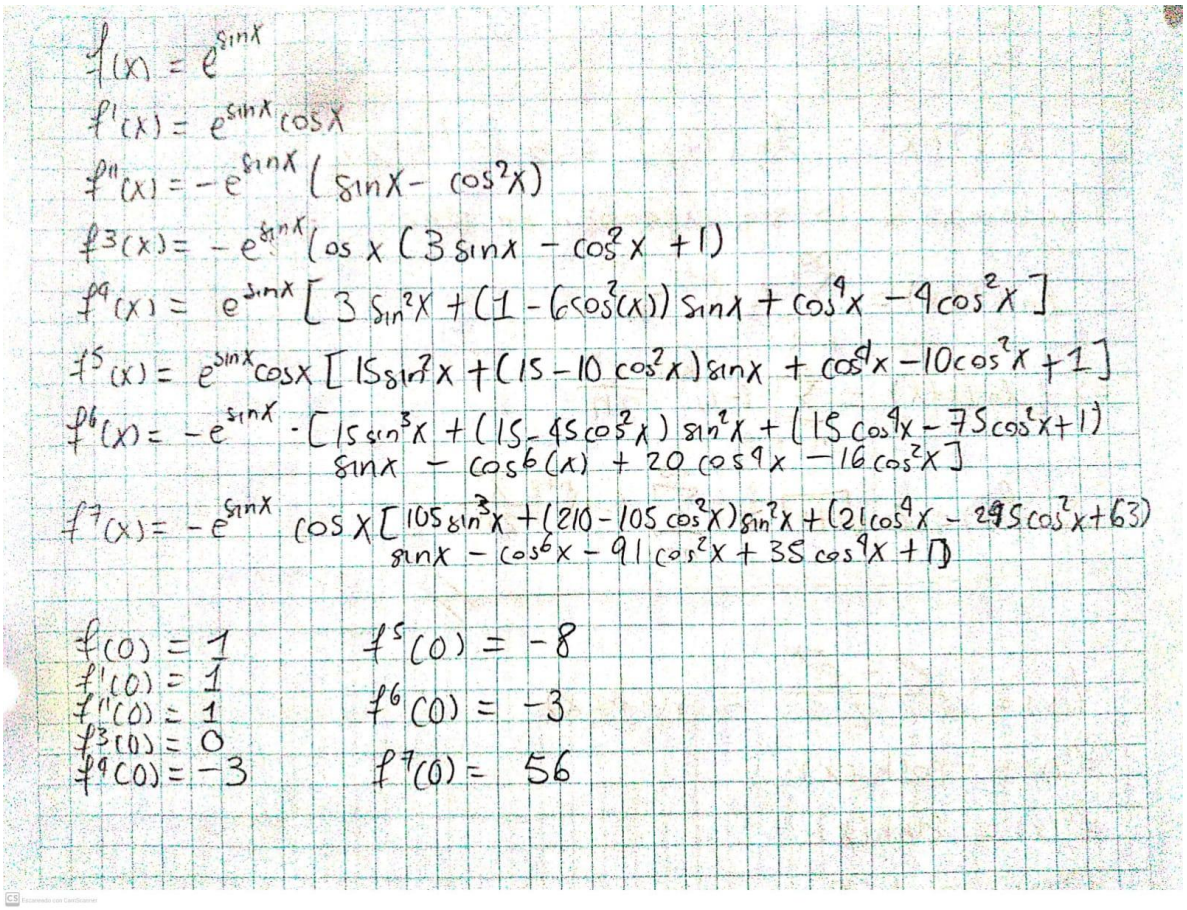
```
 ------------ Numeros de Bernoulli ---------------
 Numeros de Bernoulli a calcular:
15
           0   1.0000000000000000
           1 -0.50000000000000000
           2  0.16666666666666669
           3  -0.0000000000000000
           4  -3.3333333333333354E-002
           5   2.1681925867341953E-017
           6   2.3809523809523774E-002
           7   1.3167338923304912E-016
           8  -3.3333333333333631E-002
           9   3.7378333117580795E-016
          10   7.5757575757575649E-002
          11  -5.6550464173056457E-016
          12 -0.25311355311355221
          13   5.8255393921688205E-016
          14   1.1666666666666643
          15  -2.0555576717211055E-007
```

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
B_n	1	$\frac{-1}{2}$	$\frac{1}{6}$	0	$\frac{-1}{30}$	0	$\frac{1}{42}$	0	$\frac{-1}{30}$	0	$\frac{5}{66}$	0	$\frac{-691}{2730}$	0	$\frac{7}{6}$	0

Al ser una función recursiva, existen errores de propagación en las operaciones, podemos dar por hecho que valores con un orden de 10^{-7} son aproximadamente 0, y los valores coinciden con la tabla conseguida en un artículo sobre los números de Bernoulli.

4. $e^{\sin x}$

Para expandir esta función en Series de Taylor, necesitaremos de las primeras 7 derivadas de la función evaluadas en el punto donde se expande, en nuestro caso podemos ocupar x = 0. Así, analíticamente.



De esta forma, nuestra expansión a orden 7 de la función $e^{\sin x}$ resulta en la expresión:

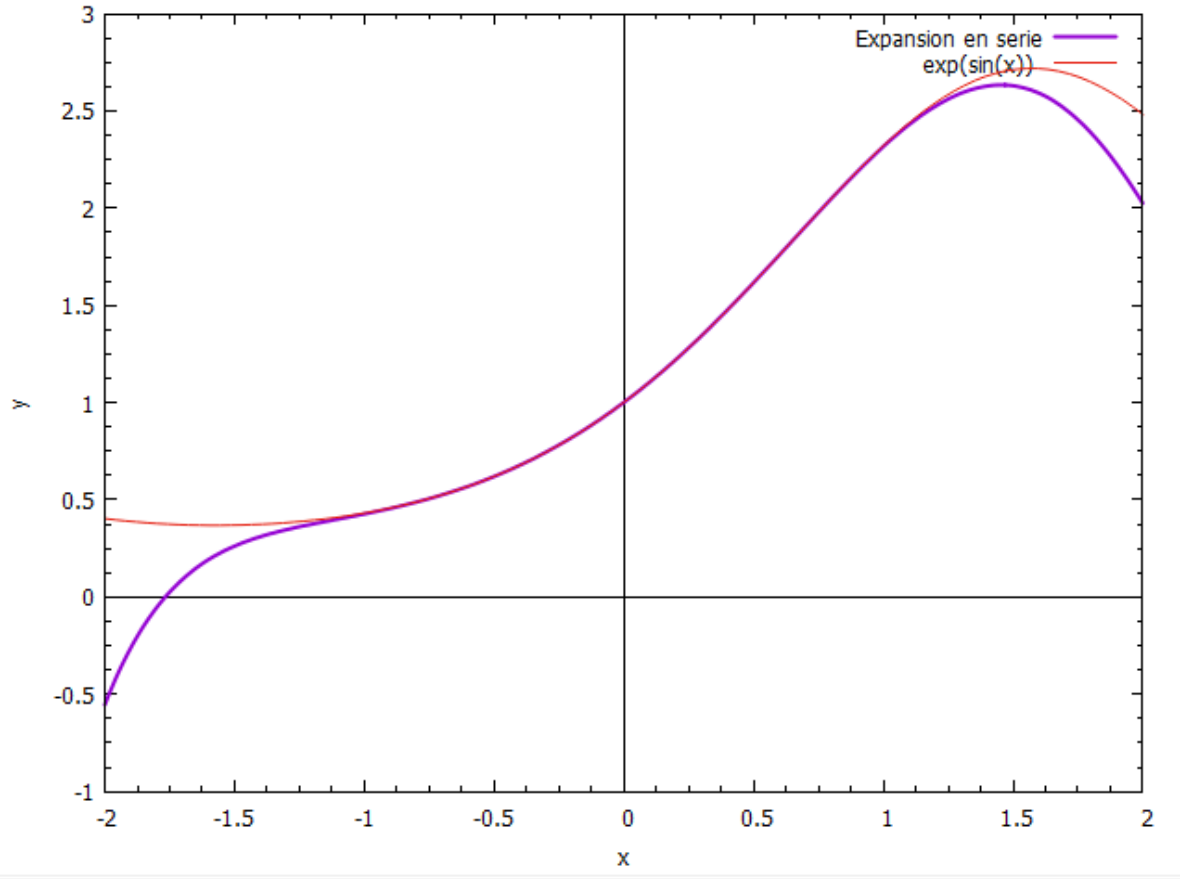
$$e^{sinx} \approx 1 + x + \frac{x^2}{2} - \frac{3x^4}{4!} - \frac{8x^5}{5!} - \frac{3x^6}{6!} + \frac{56x^7}{7!}$$

En la ejecución del programa, nos mostrará en el menú como opción 4 esta misma función.

```
Listado de funciones:
1.   exp(x)*cos(x)
2.   1/(1+9*x**2)
3.   x/(exp(x)-1)
4.   exp(sin(x))
5.   exp(-x**2)
6.   log10(1+exp(x))
7.   log(cos(x))
8.   (exp(x)-exp(-x))/2
9.   cos(x)**2
10.  log(sqrt(1+x))
Seleccion: 4
```

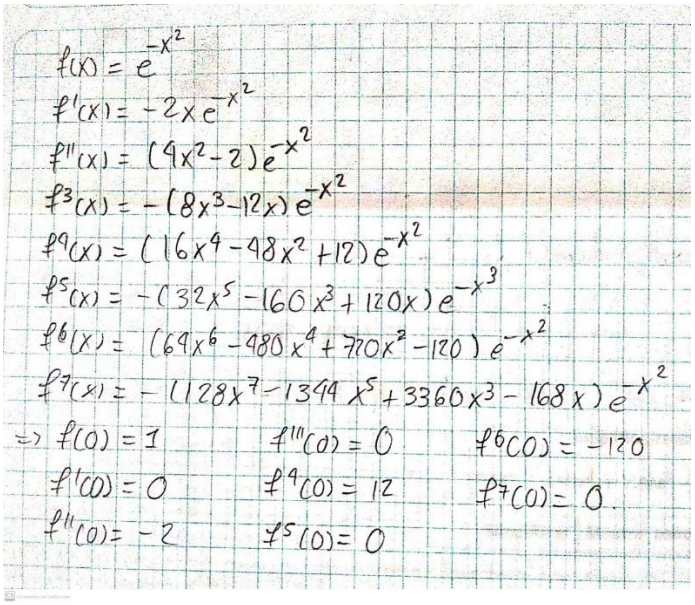
Especificando el punto alrededor de donde se realiza la expansión, el orden de la expansión y en qué intervalo se va a graficar, nos mostrará la siguiente gráfica.

```
1 archivo(s) copiado(s).
Orden de la expansion: 7
Alrededor del punto: 0
Extremos del intervalo a graficar: -2,2
```



5. e^{-x^2}

Para expandir esta función en Series de Taylor, necesitaremos de las primeras 7 derivadas de la función evaluadas en el punto donde se expande, en nuestro caso podemos ocupar x = 0. Así, analíticamente.



De esta forma, nuestra expansión a orden 7 de la función e^{-x^2} resulta en la expresión:

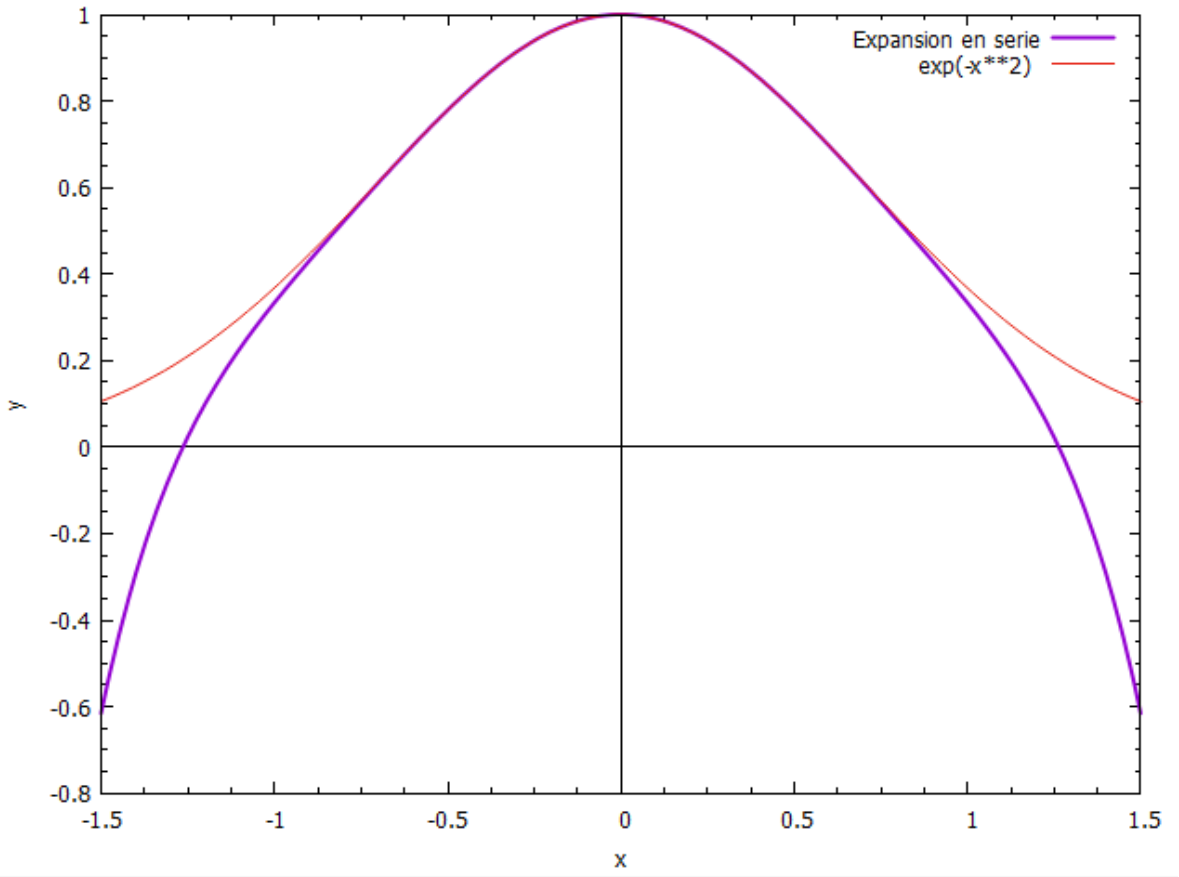
$$e^{-x^2} \approx 1 - x^2 + \frac{x^4}{2} - \frac{x^6}{6} + 0$$

En la ejecución del programa, nos mostrará en el menú como opción 5 esta misma función.

```
Listado de funciones:
1.   exp(x)*cos(x)
2.   1/(1+9*x**2)
3.   x/(exp(x)-1)
4.   exp(sin(x))
5.   exp(-x**2)
6.   log10(1+exp(x))
7.   log(cos(x))
8.   (exp(x)-exp(-x))/2
9.   cos(x)**2
10.  log(sqrt(1+x))
Seleccion: 5
```

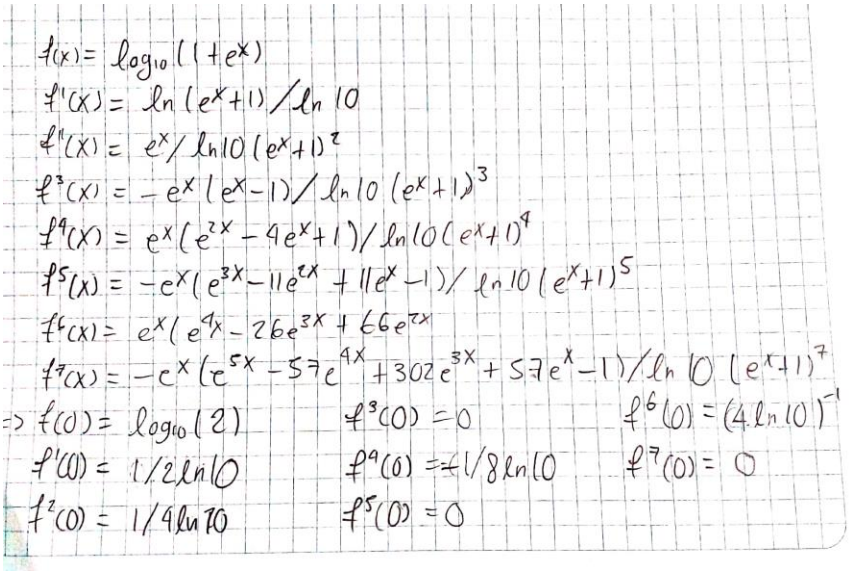
Especificando el punto alrededor de donde se realiza la expansión, el orden de la expansión y en qué intervalo se va a graficar, nos mostrará la siguiente gráfica.

```
Orden de la expansion: 7
Alrededor del punto: 0
Extremos del intervalo a graficar: -1.5,1.5
```



6. $\log_{10}(1 + e^x)$

Para expandir esta función en Series de Taylor, necesitaremos de las primeras 7 derivadas de la función evaluadas en el punto donde se expande, en nuestro caso podemos ocupar x = 0. Así, analíticamente.



De esta forma, nuestra expansión a orden 7 de la función $\log(1+e^x)$ resulta en la expresión:

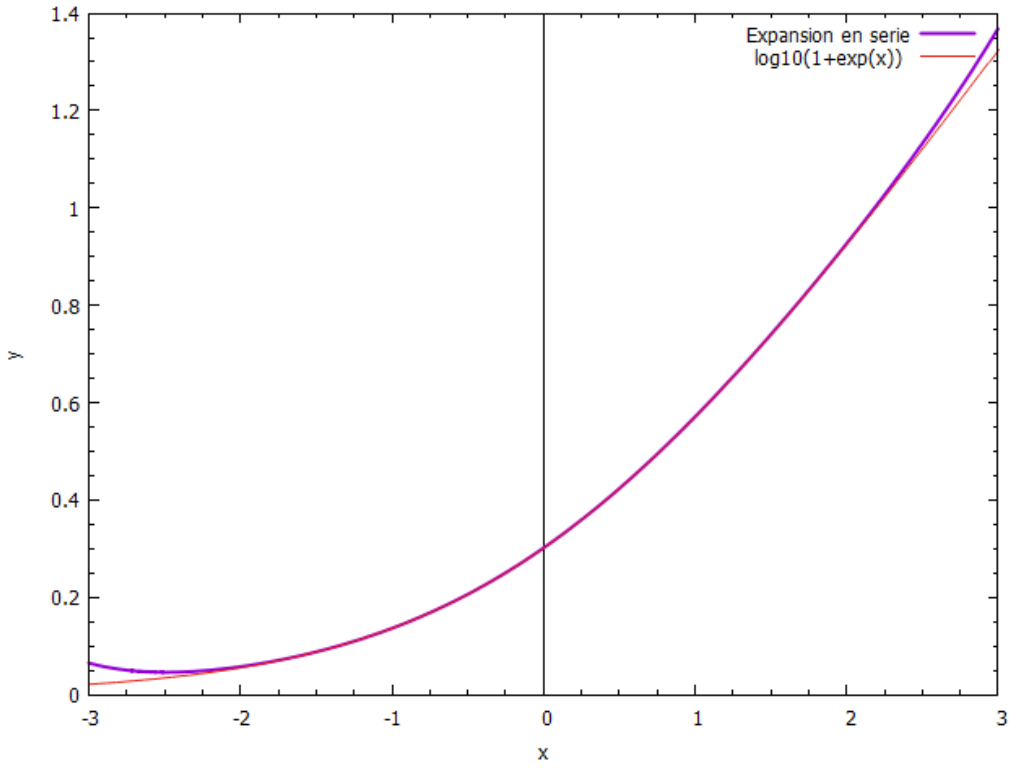
$$\log 1 + e^x \approx \log_{10} 2 + \frac{x}{2\ln 10} + \frac{x^2}{8\ln 10} - \frac{x^4}{4!\,8\ln 10} + \frac{x^6}{6!\,4\ln 10}$$

En la ejecución del programa, nos mostrará en el menú como opción 6 esta misma función.

```
Listado de funciones:
1.   exp(x)*cos(x)
2.   1/(1+9*x**2)
3.   x/(exp(x)-1)
4.   exp(sin(x))
5.   exp(-x**2)
6.   log10(1+exp(x))
7.   log(cos(x))
8.   (exp(x)-exp(-x))/2
9.   cos(x)**2
10.  log(sqrt(1+x))
Seleccion: 6
```

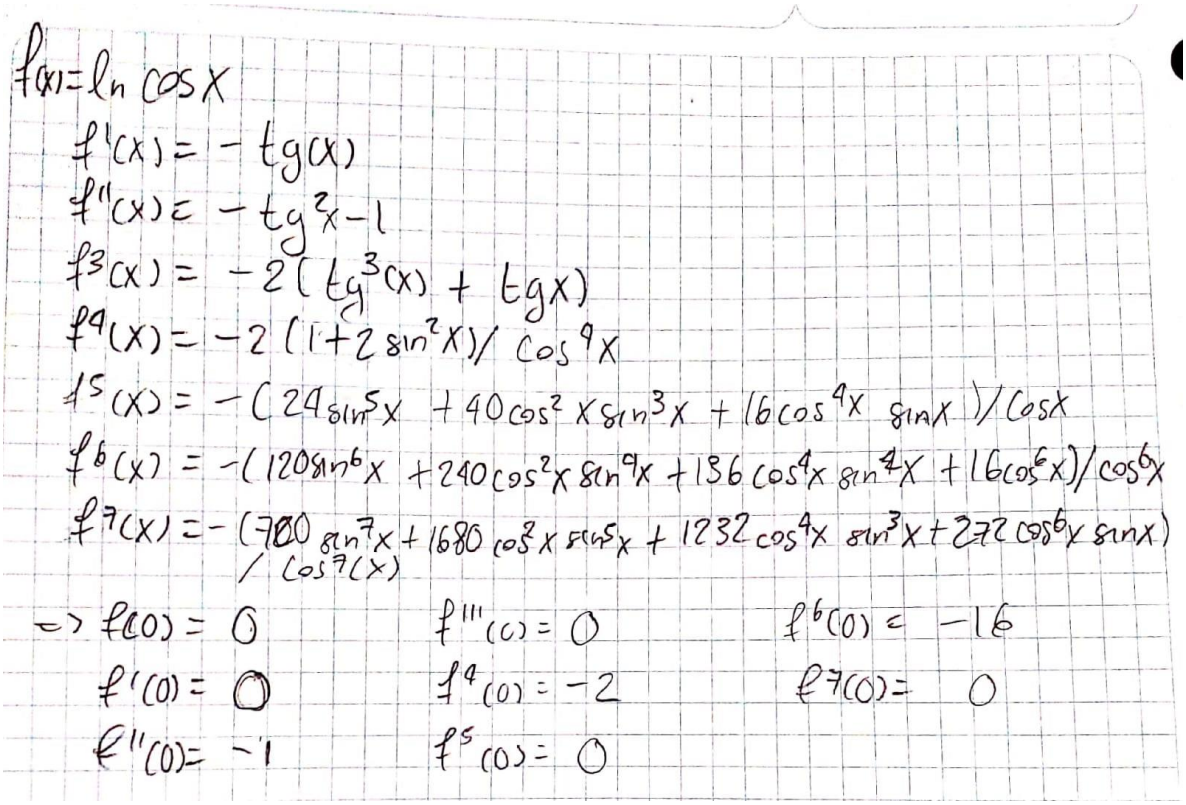
Especificando el punto alrededor de donde se realiza la expansión, el orden de la expansión y en qué intervalo se va a graficar, nos mostrará la siguiente gráfica.

```
Orden de la expansion: 7
Alrededor del punto: 0
Extremos del intervalo a graficar: -3,3
```



7. $\ln \cos x$

Para expandir esta función en Series de Taylor, necesitaremos de las primeras 7 derivadas de la función evaluadas en el punto donde se expande, en nuestro caso podemos ocupar x = 0. Así, analíticamente.



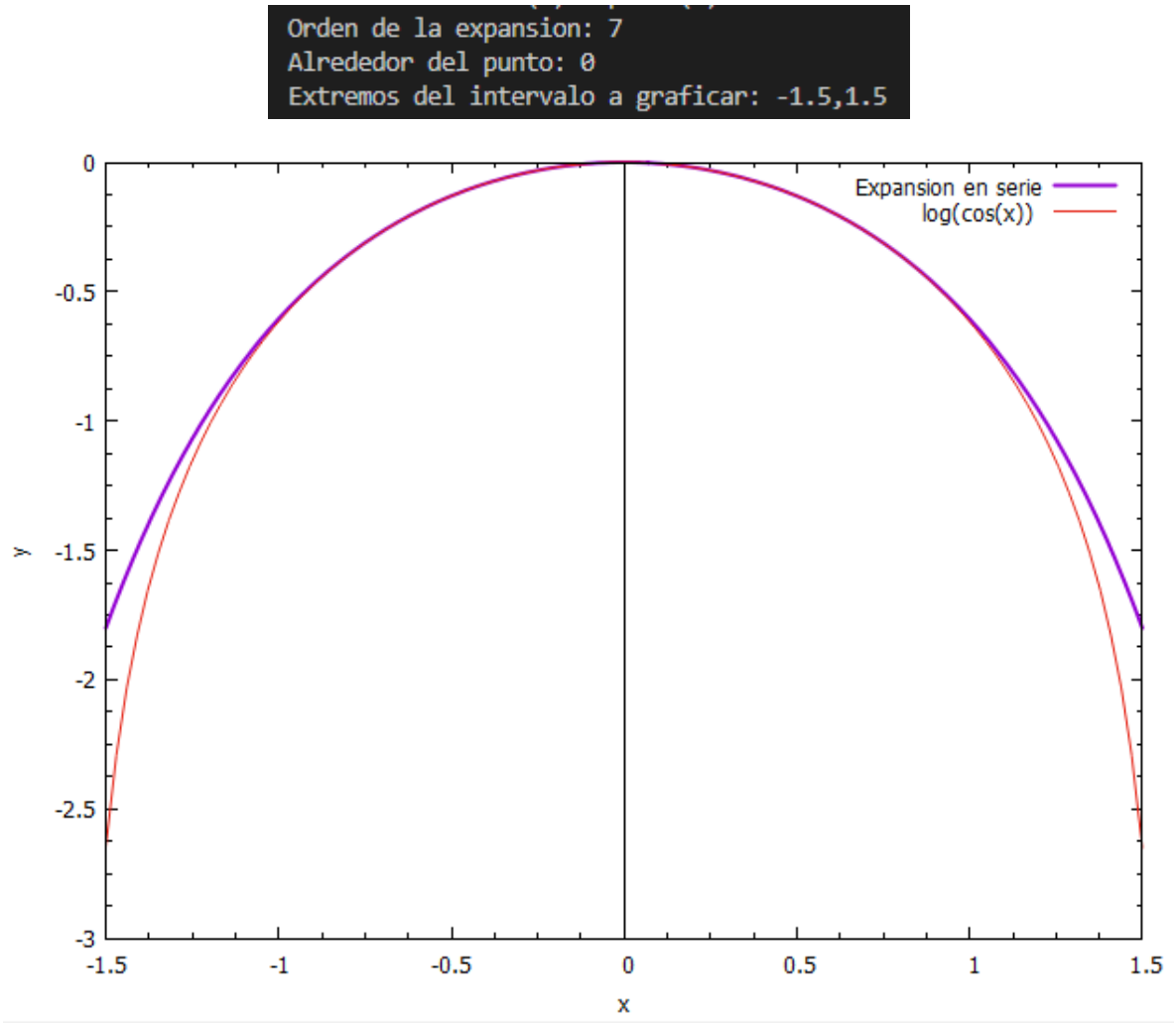
De esta forma, nuestra expansión a orden 7 de la función $\ln(\cos x)$ resulta en la expresión:

$$\ln \cos x \approx -\frac{x^2}{4} - \frac{x^4}{12} - \frac{12x^6}{6!}$$

En la ejecución del programa, nos mostrará en el menú como opción 7 esta misma función.

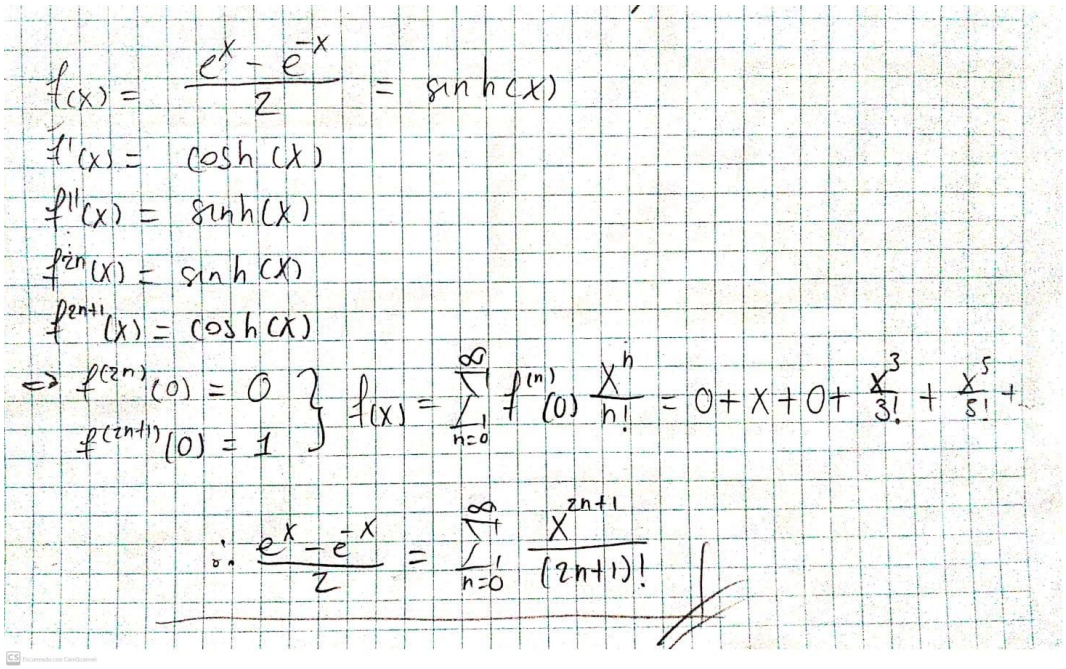
```
Listado de funciones:
1.   exp(x)*cos(x)
2.   1/(1+9*x**2)
3.   x/(exp(x)-1)
4.   exp(sin(x))
5.   exp(-x**2)
6.   log10(1+exp(x))
7.   log(cos(x))
8.   (exp(x)-exp(-x))/2
9.   cos(x)**2
10.  log(sqrt(1+x))
Seleccion: 7
```

Especificando el punto alrededor de donde se realiza la expansión, el orden de la expansión y en qué intervalo se va a graficar, nos mostrará la siguiente gráfica.



8. $\frac{1}{2}(e^{x} - e^{-x})$

Podemos observar que esta función es precisamente sinh(x), por lo que podemos derivar fácilmente su expansión en series de Taylor alrededor del 0.



De esta forma, nuestra expansión a orden 7 de la función $\frac{1}{2}(e^x - e^{-x})$ resulta en la expresión:

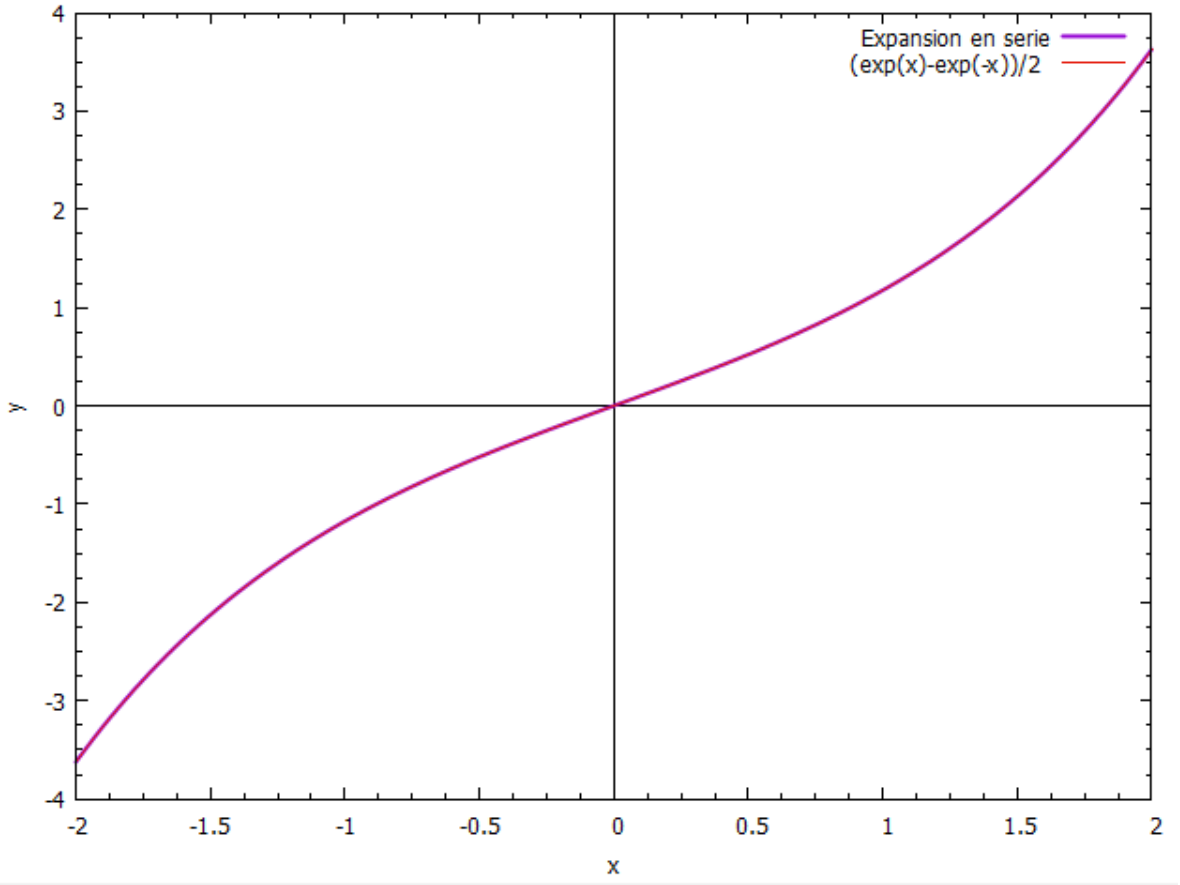
$$\frac{1}{2}(e^x - e^{-x}) \approx x + \frac{x^3}{3!} + \frac{x^5}{5!} + \frac{x^7}{7!}$$

En la ejecución del programa, nos mostrará en el menú como opción 8 esta misma función.

```
Listado de funciones:
1.   exp(x)*cos(x)
2.   1/(1+9*x**2)
3.   x/(exp(x)-1)
4.   exp(sin(x))
5.   exp(-x**2)
6.   log10(1+exp(x))
7.   log(cos(x))
8.   (exp(x)-exp(-x))/2
9.   cos(x)**2
10.  log(sqrt(1+x))
Seleccion: 8
```

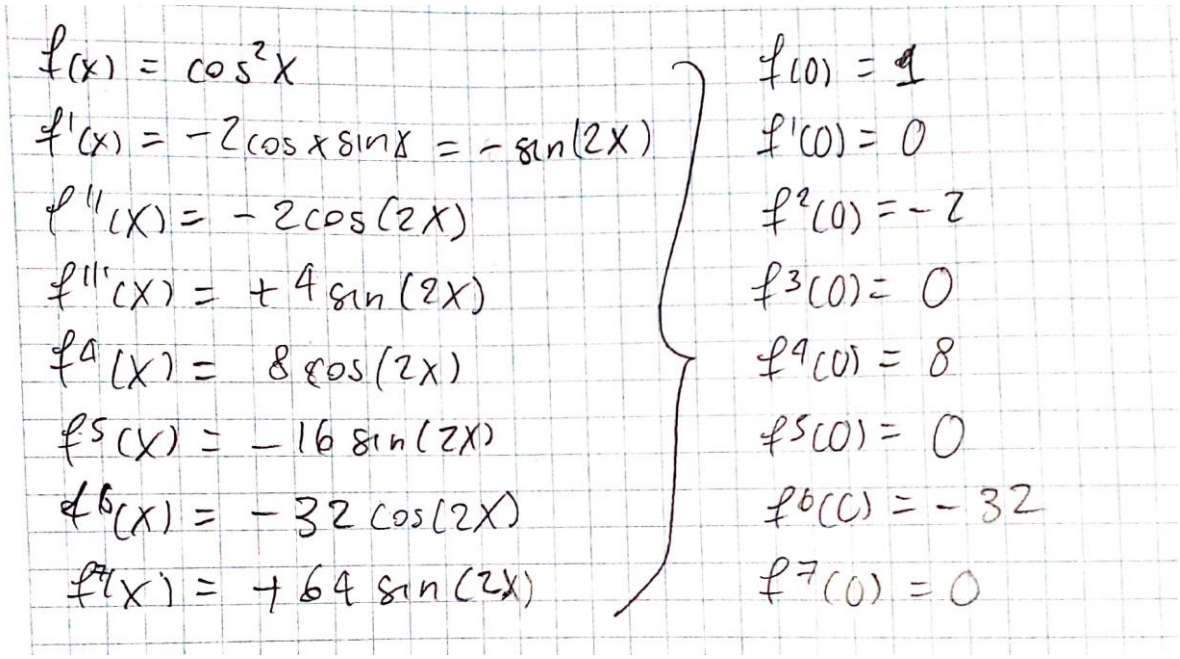
Especificando el punto alrededor de donde se realiza la expansión, el orden de la expansión y en qué intervalo se va a graficar, nos mostrará la siguiente gráfica.

```
Orden de la expansion: 7
Alrededor del punto: 0
Extremos del intervalo a graficar: -2,2
```



9. $\cos^2 x$

Para expandir esta función en Series de Taylor, necesitaremos de las primeras 7 derivadas de la función evaluadas en el punto donde se expande, en nuestro caso podemos ocupar x = 0. Así, analíticamente.



De esta forma, nuestra expansión a orden 7 de la función $\cos^2 x$ resulta en la expresión:

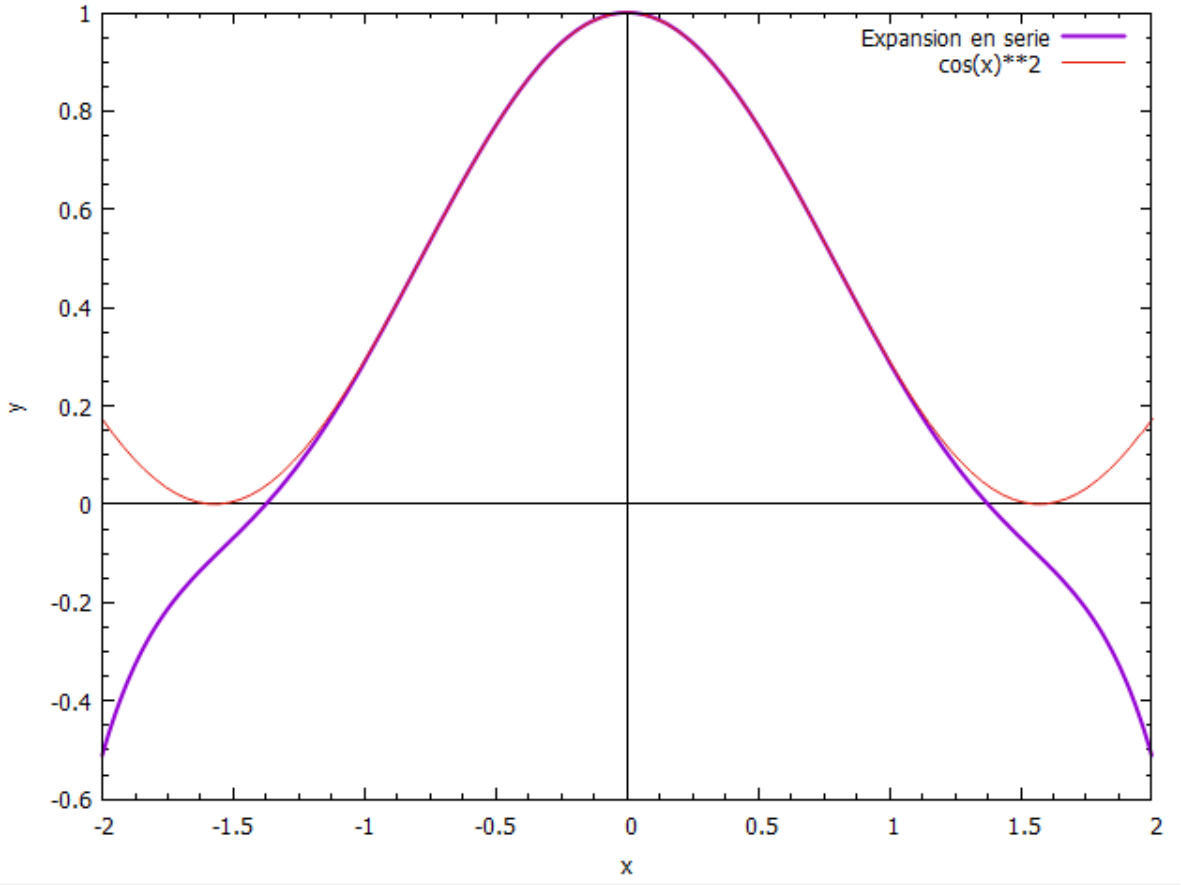
$$\cos^2 x \approx 1 - x^2 + \frac{x^4}{3} - \frac{32x^6}{6!}$$

En la ejecución del programa, nos mostrará en el menú como opción 9 esta misma función.

```
Listado de funciones:
1.   exp(x)*cos(x)
2.   1/(1+9*x**2)
3.   x/(exp(x)-1)
4.   exp(sin(x))
5.   exp(-x**2)
6.   log10(1+exp(x))
7.   log(cos(x))
8.   (exp(x)-exp(-x))/2
9.   cos(x)**2
10.  log(sqrt(1+x))
Seleccion: 9
```

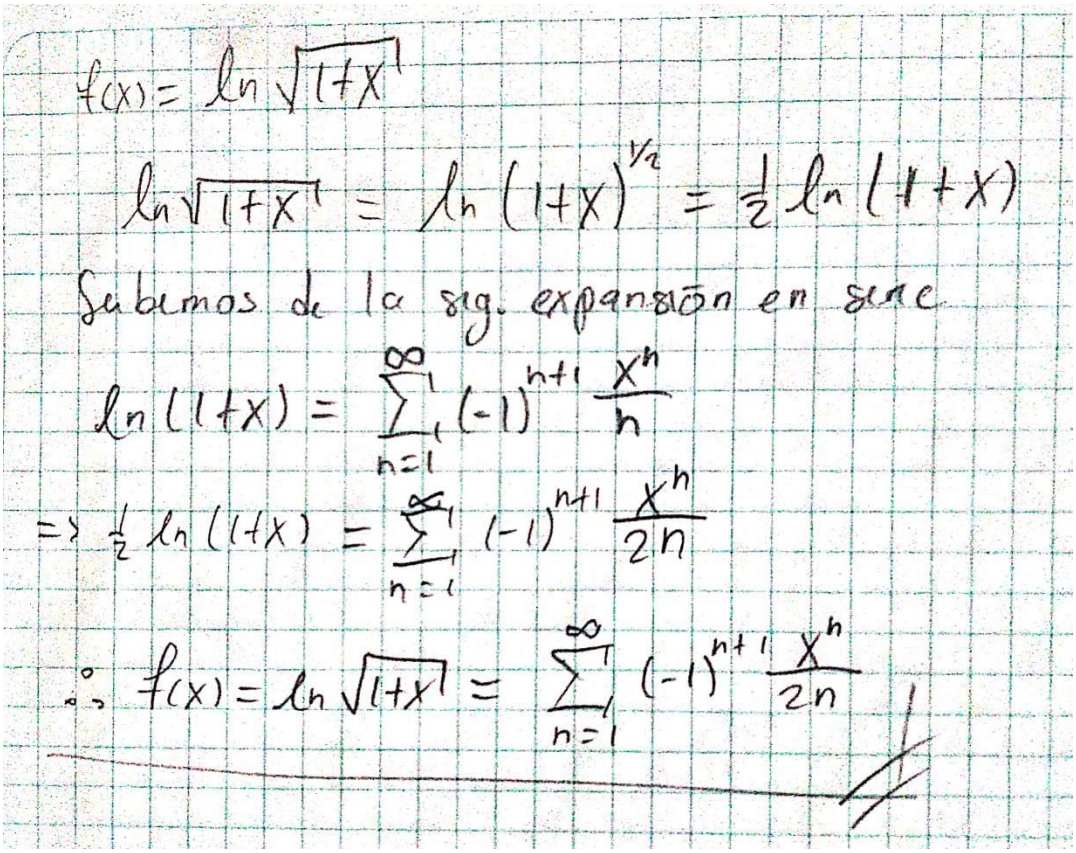
Especificando el punto alrededor de donde se realiza la expansión, el orden de la expansión y en qué intervalo se va a graficar, nos mostrará la siguiente gráfica.

```
Orden de la expansion: 7
Alrededor del punto: 0
Extremos del intervalo a graficar: -2,2
```



10. $\ln \sqrt{1+x}$

Se deduce analíticamente la expresión de la expansión en series de Taylor alrededor del 0 para esta función, de la siguiente manera:



Por lo que podemos deducir la expresión para el polinomio de Taylor de orden 7 para esta función:

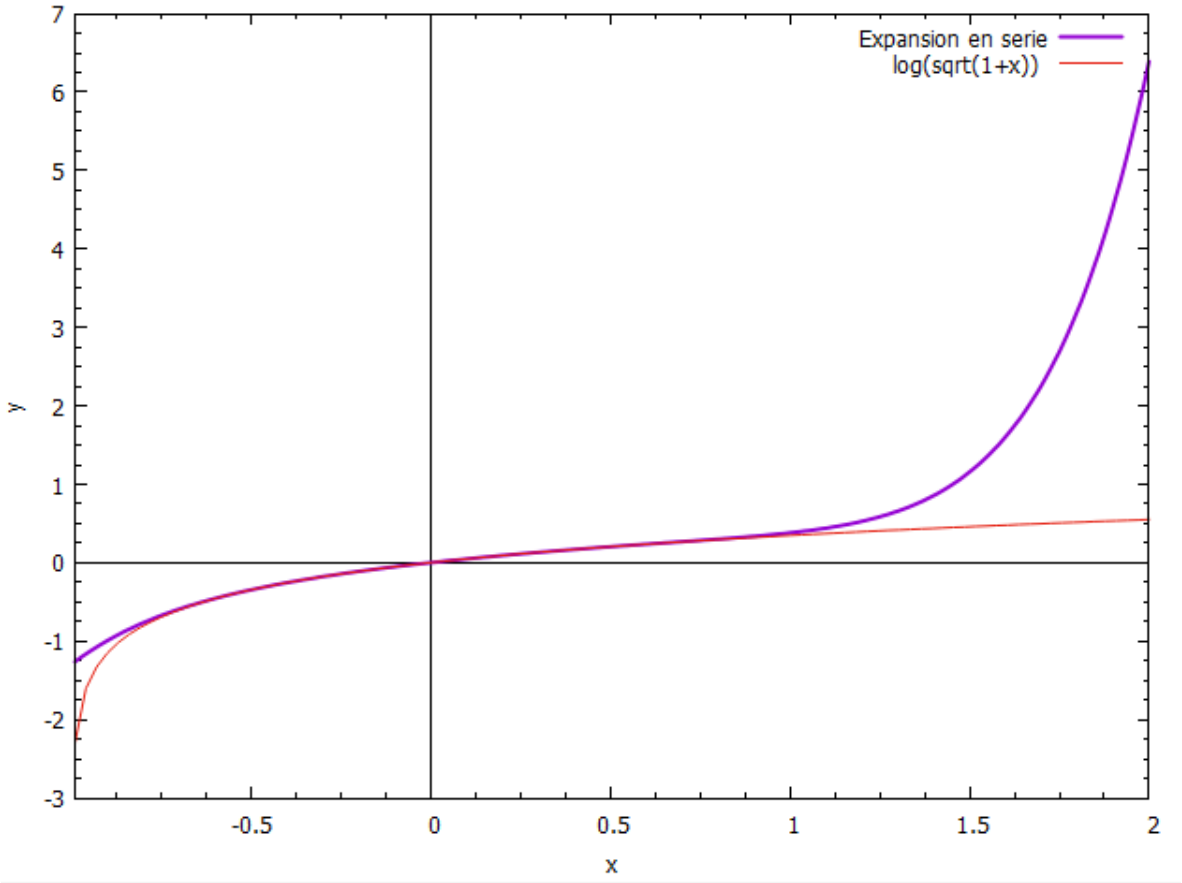
$$\ln\sqrt{1+x} \approx \frac{x}{2} - \frac{x^2}{4} + \frac{x^3}{6} - \frac{x^4}{8} + \frac{x^5}{10} - \frac{x^6}{12} + \frac{x^7}{14}$$

En la ejecución del programa, nos mostrará en el menú como opción 10 esta misma función.

```
Listado de funciones:
1.   exp(x)*cos(x)
2.   1/(1+9*x**2)
3.   x/(exp(x)-1)
4.   exp(sin(x))
5.   exp(-x**2)
6.   log10(1+exp(x))
7.   log(cos(x))
8.   (exp(x)-exp(-x))/2
9.   cos(x)**2
10.  log(sqrt(1+x))
Seleccion: 10
```

Especificando el punto alrededor de donde se realiza la expansión, el orden de la expansión y en qué intervalo se va a graficar, nos mostrará la siguiente gráfica.

```
Orden de la expansion: 7
Alrededor del punto: 0
Extremos del intervalo a graficar: -0.99,2
```



II. SEGUNDO PROBLEMA.

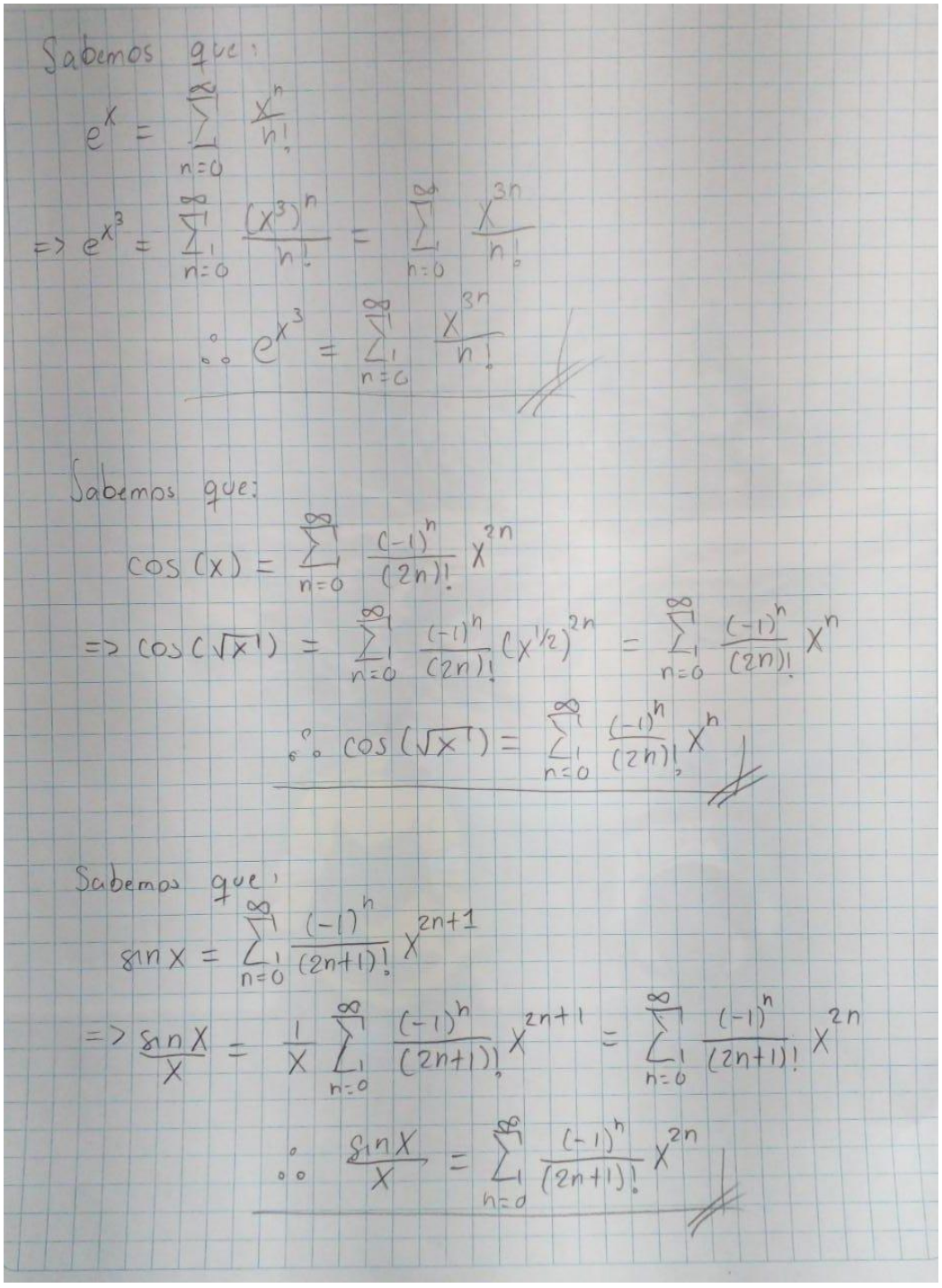
Elabore el programa en FORTRAN que calcule las siguientes integrales numéricamente usando Romberg y las resuelva usando series de potencias (x cercano a 0).

11. $\int_{-10}^{10} \frac{\sin x}{x}\, dx$
12. $\int_{0.5}^{1} e^{-x^3}\, dx$
13. $\int_{0.5}^{1} \cos\sqrt{x}\, dx$

DESARROLLO

Para este problema utilicé las ya conocidas series de Taylor tanto para la función seno, exponencial y coseno, únicamente cambiando el argumento al necesario para cada uno. Por

lo tanto, obtenemos las expresiones de las 3 funciones como series de potencias. De forma analítica se ve de la siguiente forma:



Para el programa, realicé el código tal que sirviera como otro menú. El funcionamiento es exactamente igual al del problema pasado. Para iniciar, se declaran las variables necesarias para el problema: R(20,20) es el arreglo necesario para el método de Romberg, RS(20,20) es el arreglo con el mismo objetivo que el anterior, pero guardará el resultado para la función expandida en serie de potencias, A y B son los extremos del intervalo de integración, K es la selección de la función y M es el término al que se expande en series la función seleccionada.

```
program problema2
    real*16 :: R(20,20), A, b, RS(20,20)
    integer :: n=20, k, m
```

El bloque de código que servirá como menú tiene una forma exactamente igual al del problema pasado, el funcionamiento es similar.

```
    ! Menú
    write(*,*)'Listado de funciones:'
    write(*,*)'1. exp(-x**3)'
    write(*,*)'2. cos(sqrt(x))'
    write(*,*)'3. sin(x)/x'
1    write(*,'(a)',advance='no')'Seleccion: ';read*,k
    if (k .lt. 1 .or. k .gt.3) then
        print*,'ERROR, seleccione una funcion entre 1 y 3.'
        goto 1
    end if
```

El código de la subrutina de Romberg es el que usted nos pasó, lo único que modifiqué es que no imprimiera los valores de la integral en cada iteración. Además, creé otra subrutina para el método de Romberg con las mismas modificaciones, pero ahora con la función expandida en series de potencias.

```
SUBROUTINE ROMBERG(A,B,N,R,l)
    REAL*16 ::R(20,20), f, S, A, B, H
    integer :: n
    H=B-A
    R(1,1)=(H/2.0)*(F(A,l)+F(B,l))
    DO I=2,N
        S=0
        DO K=1,2**(I-2)
            S=S+F(A+((K-0.5)*H),l)
        END DO
        R(I,1)=0.5*(R(I-1,1) + (H*S))
        H=H/2.0
    END DO
    DO J=2,N
      DO I=J,N
        R(I,J)=R(I,J-1)+((R(I,J-1)-R(I-1,J-1))/(4**(J-1))-1)
      END DO
    END DO
END SUBROUTINE
```

```
SUBROUTINE RombergSerie(A,B,N,R,l,m)
    REAL*16 ::R(20,20), serie, S, A, B, H
    integer :: n , m
    H=B-A
    R(1,1)=(H/2.0)*(SERIE(A,l,m)+serie(B,l,m))
    DO I=2,N
        S=0
        DO K=1,2**(I-2)
            S=S+serie(A+((K-0.5)*H),l,m)
        END DO
        R(I,1)=0.5*(R(I-1,1) + (H*S))
        H=H/2.0
    END DO
    DO J=2,N
      DO I=J,N
        R(I,J)=R(I,J-1)+((R(I,J-1)-R(I-1,J-1))/(4**(J-1))-1)
      END DO
    END DO
END SUBROUTINE
```

Cada subrutina tomará como función a integrar la función original y la función expandida en series de potencias, respectivamente. La forma en que se definen las funciones es mediante la elección que hayamos hecho en el menú.

```
real*16 function f(x,n)
    real*16 :: x
    integer :: n
    if (n.eq.1) f = exp(-x**3)
    if (n.eq.2) f = cos(sqrt(x))
    if (n.eq.3) f = sin(x)/x
    return
end function
```

```
real*16 function serie(x,k,n)
    real*16 :: x
    integer :: k , n
    serie = 0
    if (k.eq.1) then
        do i = 0, n
            serie = serie + (-1)**i *x**(3*i) / gamma(real(i+1))
        enddo
    else if (k.eq.2) then
        do i = 0, N
            serie = serie + (-1)**i * x**i / gamma(real(2*i + 1))
        enddo
    else
        do i = 0, N
            serie = serie + (-1)**i * x**(2*i) / gamma(real(2*i + 2))
        enddo
    end if
    return
end function
```

Una vez que están bien definidas las funciones en su versión explícita y en series de potencias, el programa le pide al usuario el intervalo de integración y el orden de la expansión en series.

```
write(*,'(a)',advance='no')'Extremos del intervalo de integracion: ';read*,A,B
write(*,'(a)',advance='no')'Orden de la expansion en series: ';read*, m
```

Una vez establecidos estos parámetros, se calcula la integral mediante Romberg de la función y de la expansión en series al orden m. Se imprimen los resultados y el error absoluto.

```
write(*,'(a)',advance='no')'El valor de la integral mediante la funcion: ';print*,R(N,1)
write(*,'(a)',advance='no')'El valor de la integral mediante la expansion: ';print*,RS(N,1)
write(*,'(a)',advance='no')'Error absoluto: ';print*,abs(R(N,1)-RS(N,1))
```

1. $\int_{0.5}^{1} e^{-x^3}\,dx$

Corremos el programa eligiendo la opción correspondiente a la función que queremos integrar, en nuestro caso es el número 1. Además, escogemos también el intervalo de integración de 0.5 a 1.0, en esta ejecución utilicé una expansión al orden 4.

```
 Listado de funciones:
 1. exp(-x**3)
 2. cos(sqrt(x))
 3. sin(x)/x
Seleccion: 1
Extremos del intervalo de integracion: 0.5,1.0
Orden de la expansion en series: 4
```

Una vez introducidos estos datos, se desplegarán automáticamente los resultados.

```
El valor de la integral mediante la funcion:   0.32259403902599825757448122769941974
El valor de la integral mediante la expansion:   0.32304988204338668920166505740398965
Error absoluto:    4.55843017388431627183829704569910912E-0004
```

Podemos observar que tan solo con 4 términos el error absoluto disminuye hasta el orden de 10^{-4}, esto se debe a que el intervalo de integración es relativamente cercano al x = 0 que es el punto alrededor de donde se generó la expansión en series de Taylor. Verificando con WolframAlpha:

Integral definida

$$\int_{0,5}^{1} e^{-x^3}\,dx = 0{,}322594$$

2. $\int_{0.5}^{1} \cos\sqrt{x}\,dx$

Corremos el programa eligiendo ahora la opción número 2. Escogeremos el mismo intervalo de integración de 0.5 a 1.0, y con el mismo argumento del problema pasado, bastará con un orden de expansión igual a 4.

```
 Listado de funciones:
 1. exp(-x**3)
 2. cos(sqrt(x))
 3. sin(x)/x
Seleccion: 2
Extremos del intervalo de integracion: 0.5,1
Orden de la expansion en series: 4
```

Una vez introducidos estos datos, se imprimen en la terminal el resultado de las integrales.

```
Orden de la expansion en series: 4
El valor de la integral mediante la funcion:   0.32433201733524663541779771403736688 6
El valor de la integral mediante la expansion:   0.32433206225198705468336229194684948 0
Error absoluto:    4.49167404192655645779094825941023945E-0008
```

Podemos observar que existe un error absoluto muy parecido al del problema anterior, probablemente sea debido a un comportamiento de las funciones muy parecido, además de utilizar el mismo intervalo de integración. Verificando con WolframAlpha:

Integral definida

$$\int_{0,5}^{1} \cos(\sqrt{x})\, dx = 0{,}324332$$

3. $\int_{-10}^{10} \frac{\sin x}{x} dx$

En este problema encontraremos dificultad al utilizar los mismos intervalos de integración que en la descripción, esto se debe a que, aún cuando la función es continúa en todos los puntos incluyendo al 0, una computadora no puede realizarlo ya que no sabe calcular límites (qué boba), haciendo explícitamente la evaluación x=0, indefiniendo la función. Lo que si podemos hacer, es modificar la expresión de tal forma que no dependamos de este error utilizando la paridad de la función, sin embargo esto modificaría la función en el código y pudiera interpretarse como trampa 😊. En un principio pretendía modificar el problema para que al encontrar una indefinición se pudiese saltar el punto, sin embargo, no encontré manera de verificar si un resultado es del tipo NaN. Por lo que, en este problema en concreto, calcularemos la integral desde un número cercano a 0 hasta el 10, y multiplicaremos manualmente este valor por 2 (paridad).

Por lo que, ejecutando este programa eligiendo la opción 3, escogiendo el intervalo de 0.001 a 10 con una expansión en series de potencias de orden 13, haciendo esta última propuesta debido a que el intervalo de integración ya no es cercano a 0.

```
 Listado de funciones:
 1. exp(-x**3)
 2. cos(sqrt(x))
 3. sin(x)/x
Seleccion: 3
Extremos del intervalo de integracion: 0.000000001,10
Orden de la expansion en series: 13
```

Se imprimen los resultados

```
El valor de la integral mediante la funcion:    1.658347593216495207080531944534694 52
El valor de la integral mediante la expansion:    1.657996073421084207255245243703081 79
Error absoluto:    3.515197954109998252867008316127299 55E-0004
```

Calcularemos ahora manualmente la integral en el intervalo completo únicamente multiplicando por 2. Nuestro error absoluto también se multiplicaría por 2. De esta forma nuestros resultados en el mismo orden presentado anteriormente son los siguientes.

$$I = 3.316695186$$

$$I_{aprox} = 3.315992147$$

$$Error\ absoluto = 7 \times 10^{-4}$$

Verificando con WolframAlpha:

Integral definida

$$\int_{-10}^{10} \frac{\operatorname{sen}(x)}{x}\, dx = 2\,\mathrm{Si}(10) \approx 3{,}3167$$

III. CONCLUSION.

Durante esta tarea aprendí muchísimo sobre cómo manejar archivos de texto en Fortran, sobre todo a cómo leer datos y cómo introducir datos con el método append a un archivo. También aprendí las dificultades de utilizar la consola (más concretamente, la subrutina intrínseca execute_command_line y system) en distintos sistemas operativos; aprender la sintaxis de distintas terminales y la necesidad de declarar el sistema operativo. Tengo duda si hay alguna forma de leer el sistema operativo directamente desde el programa, será algo que checaré. Esta nueva característica me abre muchas posibilidades con GNUPLOT y manejar archivos de texto.