Universidad Carlos III
Grado en ingeniería informática
Curso Estructura de Computadores 2023-24
Práctica 1
Curso 2023-24

Fecha: **22/10/2023** - ENTREGA: **1**

GRUPO: **81**

Alumnos: **Mario Ramos Salsón (100495849) y Miguel Fidalgo García (100495770)**

ÍNDICE

EJERCICIO 1

```
factorial:

addi sp sp -8
    fsw fs0 4(sp)
    fsw fs1 0(sp)

    li t0 1
    fcvt.s.w ft0 t0
    fcvt.s.w fs1 t0
    fcvt.s.w fs0 t0
    fcvt.s.w fa0 a0

    bucle_f: bgt t0 a0 end_f
        fmul.s fs1 fs1 fs0
        fadd.s fs0 fs0 ft0
        addi t0 t0 1
        j bucle_f

    end_f:
        fmv.s fa0 fs1

        flw fs0 4(sp)
        flw fs1 0(sp)
        addi sp sp 8
        jr ra
```

Para la realización de este ejercicio, hemos decidido apoyarnos en una función auxiliar “Factorial” para realizar cada uno de los apartados de una manera más sencilla y óptima, ya que reducimos notoriamente la cantidad de código repetido.

En esta función, haciendo uso de la pila para evitar perder los registros “fs0” y “fs1”, conseguimos calcular el factorial de un número pasado por el registro “a0”, para posteriormente devolverlo por el mismo registro. Esto es de gran utilidad ya que para cada interacción tanto del número “e” como del seno o del coseno, hay que calcular el factorial de algún número. Por ello, haciendo uso de esta función, conseguimos optimizar espacio y código.

```python
def factorial(num: int) -> int:
      acum, i = 1, 1
      while i <= num:
            acum = acum * i
            i += 1
      return acum
```

SENO Y COSENO

Esta función consigue calcular de forma precisa el seno de un número expresado en radianes que se pasa como parámetro a través del registro “fa0”. En esta función hemos decidido hacer un gran uso de la pila, sobre todo para no perder en ningún momento la dirección de retorno guardada en el registro “ra”, como a su vez para guardar cada registro

de tipo “s” que utilicemos. Esto nos ayuda a cumplir con el convenio de paso de parámetros para realizar una función acorde a los estándares.

Cabe destacar que a la hora de calcular el valor de (-1)^n, en vez de optar por hacer un bucle para calcular la potencia, hemos decidido analizar si el valor de “n” es par o impar para así devolver el número “1” o “-1” respectivamente. Esto nos ha parecido una manera más óptima de plantear el problema ahorrándonos tanto recursos del programa como tiempo.

```python
resto = n%2

if resto == 0:
      c = 1     # Si la n es par (-1)^n es 1
else:
      c = 1     # Si la n es impar (-1)^n es -1
```

Una vez calculado el valor de (-1)^n, calcularemos x^(2n + 1). Para ello en este caso, hemos realizado un bloque de código que multiplique muchas veces el valor de (2n + 1) para simular la operación de elevar. Este valor (2n + 1) lo hemos almacenado en el registro s2, y posteriormente lo hemos usado para la función factorial. La operación de x^(2n + 1) la hemos almacenado en el registro “ft5” y hemos convertido el valor almacenado del registro s4 (1 o -1) en un número expresado en coma flotante mediante la función “fcvt.s.w” para a la hora de dividir que no de problemas.

```python
for _ in range(a):  # Para multiplicar x^(2n + 1)
      b = b*x

d = c*b             # (-1)^n * x^(2n + 1)
```

Para finalizar, usaremos el valor previamente calculado en “s2” (2n + 1) para pasárselo a la función factorial como parámetro. Previamente, convertimos dicho valor en un número

expresado en coma flotante, ya que esto nos ayudará considerablemente a ganar precisión a la hora de realizar las iteraciones. Esto es debido a que un número expresado en IEEE 754, es capaz de almacenar más rango que un binario natural, por lo que conseguimos cumplir el requisito de que no debe haber un error mayor del 0.001. Una vez que dicha función nos devuelva el factorial, simplemente operamos con los otros valores calculados previamente, e incrementamos el registro que almacena todo el sumatorio, para una vez finalizado todo el programa devolverlo.

```python
e = factorial(a)    # Calculamos el factorial de (2n + 1)
f = d/e             # Dividendo/divisor ((-1)^n * x^(2n + 1)) / (2n + 1)
resultado += f
return resultado
```

Para la función coseno, realizaremos exactamente el mismo procedimiento que en el seno, cambiando únicamente el valor de (2n + 1) por (2n). Además, los registros pueden ser los mismos, ya que ambas funciones no se solapan y no modifican los registros de la otra, lo que generaría un problema a la hora de ejecutar el código.

TANGENTE

Para el cálculo de la tangente, usaremos la expresión matemática (sen/cos). Para ello, llamaremos primero a la función seno con la expresión (jal ra sen) y posteriormente a la función coseno con la expresión (jal ra cos). Como hemos hecho en las funciones anteriores, guardamos en la pila el valor del registro “ra” para poder volver a él una vez finalizada la función, pero en este caso hemos optado por guardamos también en la pila el valor de “fa0”. Esto nos ayuda a ahorrarnos un registro, ya que como vamos a usar ese valor dos veces, una para el seno y otra para el coseno, lo que hacemos es sacarlo de la pila y pasarlo directamente al registro “fa0” antes de llamar a la siguiente función, ahorrándonos un registro.

```python
def tg(x: int) -> int:

      f1 = sen(x)
      f2 = cos(x)

      return (f1/f2)
```

NÚMERO E

Para el número “e”, hemos realizado x iteraciones, en las cuales dividimos el número 1 entre el factorial de la n correspondiente a cada iteración. Al igual que en la función seno, coseno y tangente, nos guardamos nada más llamar a la función el valor del registro “ra” en la pila para no perderlo. Además, operaremos constantemente con registros flotantes para ganar precisión en los cálculos y conseguir un error mucho menor del 0.001.

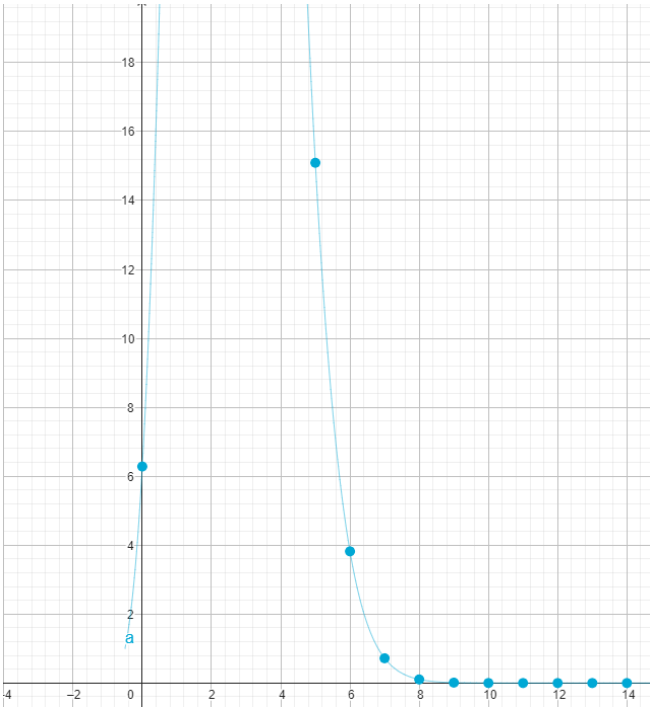
```python
def E() -> int:
      total = 0

      for n in range(x):
            a = factorial(n)
            total = total + (1/a)
      return total
```

GRÁFICAS

Podemos observar que el seno dentro del rango (-2π, 2π), empieza a tener un error menor del 0.001 a partir de la décima iteración. Haremos 11 repeticiones.

$$a(x) = \frac{(-1)^x (2\pi)^{2x+1}}{(2x+1)!}$$

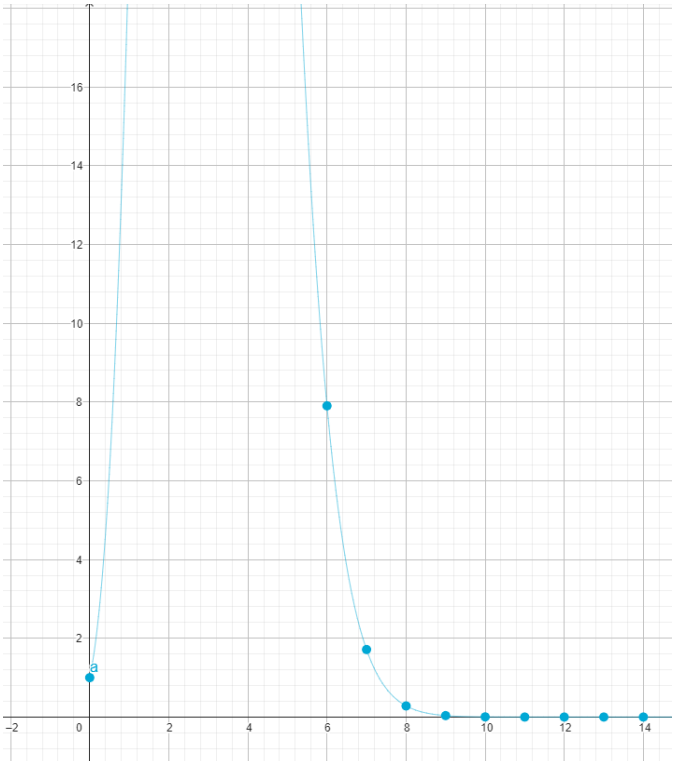
x	a(x)
1	41.341702240...
2	81.605249276...
3	76.705859753...
4	42.058693944...
5	15.094642576...
6	3.8199525848...
7	0.7181223017...
8	0.1042291622...
9	0.0120315859...
10	0.0011309237...
11	0.000088235336
12	0.0000058056...
13	0.0000003264...
14	0.0000000158...



Podemos observar que el coseno dentro del rango (-2π, 2π), empieza a tener un error menor del 0.001 a partir de la décima iteración. Haremos 11 repeticiones.

$$a(x) = \frac{(-1)^x (2\pi)^{2x}}{(2x)!}$$

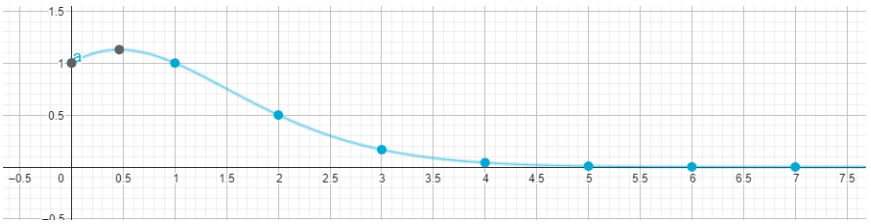
x	a(x)
0	1
1	19.739208802...
2	64.939394022...
3	85.456817206...
4	60.244641371...
5	26.426256783...
6	7.9035363713...
7	1.7143907110...
8	0.2820059684...
9	0.0363828411...
10	0.0037798342...
11	0.0003229910...



Para finalizar, poniendo la función “e” en geogebra y dando valores desde el 0 hasta el 9 por ejemplo, observamos que empieza a haber un error menor de 0.001 a partir de la sexta iteración. Haremos 7 repeticiones.

x	a(x)
0	1
1	1
2	0.5
3	0.166666666...
4	0.041666666...
5	0.008333333...
6	0.001388888...
7	0.000198412...
8	0.000024801...
9	0.000002755...

$$a(x) = \frac{1}{x!}$$



EJERCICIO 2

El algoritmo de este ejercicio, una vez obtenemos la función seno correcta consiste en lo siguiente: guardamos en la pila todo lo que vamos a modificar para no perder su valor durante la ejecución dela función SinMatrix (ra, s5, s4, s3, s2, s1 y s0) y después establecemos los contadores:

```
        li s0 0  # Contador externo i
        li s1 0  # Contador interno j
        mv s2 a2 # Limite contador i (filas), suponiendo que el parametro a2 sea N filas
        mv s3 a3 # Limite contador j (columnas), suponiendo que el parametro a3 sea M
columnas
        mv s4 a0 # Direccion inicio matrixA
        mv s5 a1 # Direccion inicio matrixB

bucle_1:bge s0 s2 fin1   # Bucle externo para las filas (i)
bucle_2:bge s1 s3 fin2   # Bucle interno para las columnas (j)
        flw fa0 0(s4)    # Nos guardamos el valor en la posicion [i][j] = [s0][s1]
        jal ra sen
        fsw fa0 0(s5)    # Escribimos el seno en la misma posicion
        addi s4 s4 4     # Nos movemos a la siguiente posicion en memoria de la matrixA
(+4)
        addi s5 s5 4     # Nos movemos a la siguiente posicion en memoria de la matrixB
(+4)
        addi s1 s1 1     # Incrementamos en 1 el contador interno (j)
        j bucle_2
fin2:   addi s0 s0 1     # Incrementamos en 1 el contador externo (i)
        li s1 0          # Reseteamos el contador i
        j bucle_1
```

La idea del algoritmo es que se establezcan dos contadores, el externo para las filas y el interno para las columnas. Cargamos el primer valor de la matrizA en el registro fa0 ya que luego lo usaremos como parámetro para llamar a la función seno, que nos devolverá el resultado también en fa0 por lo que escribimos ese valor en la misma posición pero en la matrizB. Una vez hecho esto deberemos aumentar las direcciones de memoria s4 y s5 en 4 ya que es el tamaño que separa cada dato en la matriz y aumento en 1 el contador interno. Cuando terminamos la fila, aumentamos en 1 el contador externo y reseteamos el contador interno (de las columnas j) a 0.

```python
def SinMatrix(matrizA: list, matrizB: list, filas: int, columnas: int) -> list:
  # i es el contador externo, filas el limite del contador que se pasa por parametro
  for i in range(filas):
    # j es el contador interno, columnas el limite del contador que se pasa por parametro
    for j in range(columnas):
      a = matrizA[i][j] # Guardamos el valor de la matriz[i][j] (como si lo guardara en fa0)
      b = sin(a) # Llamo al seno pasando por parametro a
      matrizB[i][j] = b # Escribimos el resultado del seno en la misma posicion de la otra matriz
      # No necesitamos aumentar el contador de j porque el for lo hace automaticamente
      # Lo mismo pasa con el de la i y con resetear la j entre unas filas y otras
```

BATERÍA DE PRUEBAS

Datos a introducir:	Descripción de la prueba:	Resultado esperado:	Resultado obtenido:	Comentarios (errores)
Número 7	**Seno(7) Coseno(7) Tangente(7)**	**Seno: 0.6569 Coseno:0.7539 Tangente: 0.8714**	**Seno: 38.30600 Coseno: 7.170000 Tangente: 5.34250**	**Función factorial con enteros, sufría desbordamiento.**
Número 3	**Seno(3) Coseno(3) Tangente(3)**	**Seno: 0.1411200 Coseno: -0.989992 Tangente: -0.1425**	**Seno: -10380352946 Coseno: 1878337519 Tangente:-5.5263514**	**Desbordamiento debido a más de 6 iteraciones.**

Infinito	**Seno(inf)** **Coseno(inf)** **Tangente(inf)**	**Seno: NaN** **Coseno: Nan** **Tangente: NanN**	**Seno: NaN** **Coseno: Nan** **Tangente: NaN**	**En este caso, al no existir no nos da error.**
Número 0	**Seno(0)** **Coseno(0)** **Tangente(0)**	**Seno: 0.0** **Coseno: 1.0** **Tangente: 0.0**	**Seno: 0.0** **Coseno: 1.0** **Tangente: 0.0**	**En este caso la prueba fue exitosa.**
Número π/2	**Seno(π/2)** **Coseno(π/2)** **Tangente(π/2)**	**Seno: 1.0** **Coseno: 0.0** **Tangente: Inf (aprox)**	**Seno: 0.999999** **Coseno: -6.42796 e-8** **Tangente: -15557035**	**Al π ser infinito es imposible que de el número exacto, pero la aproximación es correcta.**
Número 5	**Seno(5)** **Coseno(5)** **Tangente(5)**	**Seno: -0.9589** **Coseno: 0.2836** **Tangente: -3.3805**	**Seno: -0.9589** **Coseno: 0.2836** **Tangente: -3.3805**	**Al haber solucionado los errores, el resultado es correcto.**
Ejercicio 2 (primera versión)	**Ejecución con una matriz aleatoria.**	**Seno de cada número de la matriz.**	**Se obtienen más valores en la matriz de destino que en la de origen**	**Esto es debido a un erróneo uso de la pila, que modificaba el valor de los contadores al llamar a la f seno.**
Ejercicio 2 (versión final)	**Ejecución con una matriz aleatoria.**	**Seno de cada número de la matriz**	**Se obtienen correctamente los valores esperados.**	**Todo funciona correctamente con valores en el rango [-2π, 2π]**

```
fmv.x.w t1 ft1
if_inf: bne t1 zero tg_normal
        # Si el cos es 0, la tangente es infinita
        li t0 0x7F800000 # Cargo el infinito en t0
        fmv.w.x ft0 t0   # Lo muevo a ft0
        j tg_inf
tg_normal: fdiv.s ft0 ft0 ft1 # Resultado si cos != 0
```

Hemos incluido un condicional en el código para que si el cos = 0 (ft1) se carga un infinito en el resultado de la tangente porque tg = sen/cos = 1/0 = inf.

Después de obtener dos errores, con el número 7 y con el 3, hemos solucionado el problema del desbordamiento en el factorial mediante el uso de registros en coma flotante para que no pierda información con los decimales. Además, el problema de las iteraciones se ha resuelto también con este uso de registros en coma flotante asegurando que cualquier valor que introducimos va a aparecer con un error menor del 0.001.