Tema 1 - Escalas, condición y estabilidad

Curso de Física Computacional

M. en C. Gustavo Contreras Mayén

Facultad de Ciencias - UNAM

2 de septiembre de 2017

Precisión de la máquina

Una preocupación importante en el cálculo computacional para la representación en punto flotante que se utliza para almacenar números es que se cuenta con una precisión limitada.

Precisión de la máquina

Como hemos visto, para una máquina de 32 bits, los números de precisión simple son buenos para $6-7$ decimales, mientras que los de precisión doble son buenos para $15-16$ lugares.

Precisión de la máquina

Como hemos visto, para una máquina de 32 bits, los números de precisión simple son buenos para $6-7$ decimales, mientras que los de precisión doble son buenos para $15-16$ lugares.

Para ver cómo la precisión limitada afecta a los cálculos, consideremos la suma en la computadora de dos palabras de precisión simple:

Precisión de la máquina

Como hemos visto, para una máquina de 32 bits, los números de precisión simple son buenos para $6 - 7$ decimales, mientras que los de precisión doble son buenos para $15 - 16$ lugares.

Para ver cómo la precisión limitada afecta a los cálculos, consideremos la suma en la computadora de dos palabras de precisión simple:

$$7 + 1.0 \times 10^{-7}$$

Representación de los valores

La comptuadora extrae estos números de la memoria y los almacena en cadenas de bits

$$
\begin{aligned}
7 &= 0\;10000010\;1110\;0000\;0000\;0000\;0000\;000 \\
10^{-7} &= 0\;01100000\;1101\;0110\;1011\;1111\;1001\;010
\end{aligned}
$$

Suma de los dos números

Debido a que los exponentes son diferentes, no es correcto sumar las mantisas, y así el exponente del número más pequeño se hace más grande mientras que disminuye progresivamente la mantisa desplazando bits a la derecha (insertando ceros) hasta que ambos números tengan el mismo exponente.

$$10^{-7} \quad = \quad 0\ 01100000\ 1101\ 0110\ 1011\ 1111\ 1001\ 010$$

$$
\begin{aligned}
10^{-7} &= 0\ 01100000\ 1101\ 0110\ 1011\ 1111\ 1001\ 010 \\
&= 0\ 01100001\ 0110\ 1011\ 0101\ 1111\ 1100\ 101\ (0)
\end{aligned}
$$

$$
\begin{aligned}
10^{-7} &= 0\ 01100000\ 1101\ 0110\ 1011\ 1111\ 1001\ 010 \\
&= 0\ 01100001\ 0110\ 1011\ 0101\ 1111\ 1100\ 101\ (0) \\
&= 0\ 01100010\ 0011\ 0101\ 1010\ 1111\ 1110\ 010\ (10)
\end{aligned}
$$

$$
\begin{aligned}
10^{-7} &= 0\ 01100000\ 1101\ 0110\ 1011\ 1111\ 1001\ 010 \\
&= 0\ 01100001\ 0110\ 1011\ 0101\ 1111\ 1100\ 101\ (0) \\
&= 0\ 01100010\ 0011\ 0101\ 1010\ 1111\ 1110\ 010\ (10) \\
&\ldots
\end{aligned}
$$

$$
\begin{aligned}
10^{-7} &= 0\ 01100000\ 1101\ 0110\ 1011\ 1111\ 1001\ 010 \\
&= 0\ 01100001\ 0110\ 1011\ 0101\ 1111\ 1100\ 101\ (0) \\
&= 0\ 01100010\ 0011\ 0101\ 1010\ 1111\ 1110\ 010\ (10) \\
\dots \\
&= 0\ 10000010\ 0000\ 0000\ 0000\ 0000\ 00000\ 000\ (0001101\dots \\
&\to 7 + 1.0 \times 10^{-7} = 7
\end{aligned}
$$

Dígitos significativos

Debido a que no hay espacio para almacenar los últimos dígitos, éstos se pierden, y después de todo este largo proceso, la suma sólo devuelve 7 como respuesta (tenemos un error de truncamiento)

Dígitos significativos

Debido a que no hay espacio para almacenar los últimos dígitos, éstos se pierden, y después de todo este largo proceso, la suma sólo devuelve 7 como respuesta (tenemos un error de truncamiento)

En otras palabras, debido a que un equipo de 32 bits almacena sólo 6 o 7 decimales, ignora efectivamente cualquier cambio más allá del sexto decimal.

Precisión de la máquina

La pérdida de precisión anterior se categoriza definiendo la precisión de la máquina ϵ_m como el máximo número positivo que en la computadora, se puede sumar al número almacenado como 1 sin cambiarlo:

$$1_c + \epsilon_m = 1_c \tag{1}$$

Por consiguiente, un número arbitrario x puede considerarse relacionado con su representación en punto flotante x_c por

$$x_c = x(1 \pm \epsilon) \qquad |\epsilon| \leq \epsilon_m$$

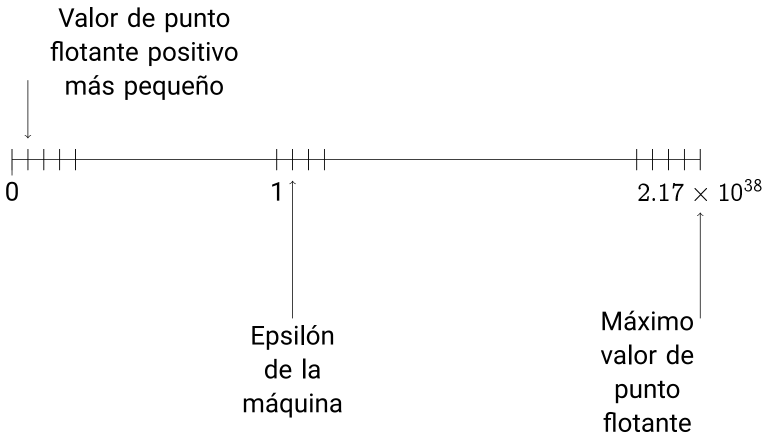
donde no se conoce el valor real de ϵ.

En otras palabras, con excepción de las potencias de 2 que están representadas exactamente, debemos suponer que todos los números de precisión simple contienen un error en el sexto decimal y que todos los dobles tienen un error en el decimoquinto lugar.

Y como siempre ocurre con los errores, debemos suponer que no sabemos cuál es el error, porque si lo supiéramos: ¡entonces lo eliminaríamos!

En consecuencia, los argumentos que planteamos con respecto a los errores son siempre aproximados, y eso es lo mejor que podemos hacer.

El epsilón de la máquina



Calculando el epsilón de la máquina

Una tarea inicial que debemos de atender, es el cálculo de la precisión de la máquina, a la que llamaremos **epsilón de la máquina**.

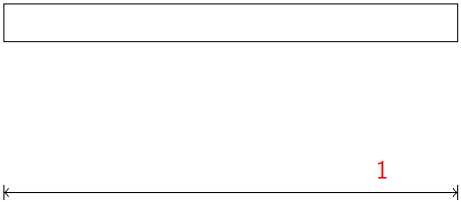
Calculando el epsilón de la máquina

Una tarea inicial que debemos de atender, es el cálculo de la precisión de la máquina, a la que llamaremos **epsilón de la máquina**.

Veamos cómo calcular con `python` ese valor:

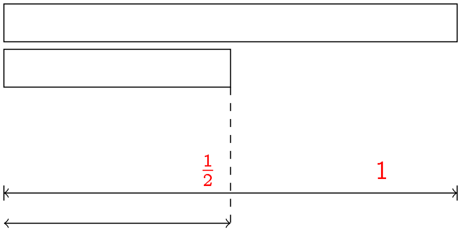
Punto de partida

Tomamos el valor de 1



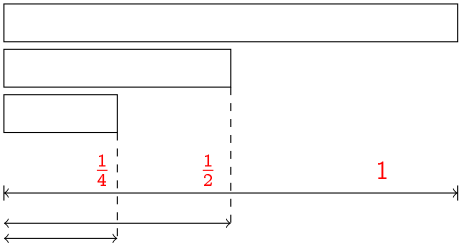
Punto de partida

Dividimos a la mitad esta unidad



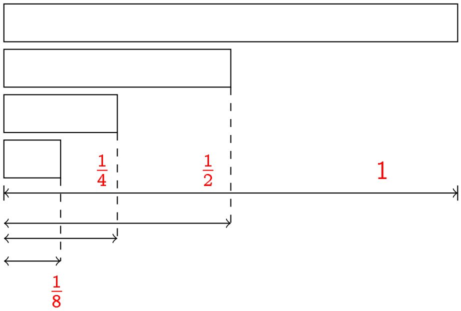
Punto de partida

Volvemos a dividir a la mitad



Punto de partida

Seguimos dividiendo a la mitad ...



Calculemos el epsilón con `python`

```
t = 1.0;

while 1+ t != 1:
    eps = t
    t = t/2

print ('el epsilon de la maquina es: ',
   eps)
```

Calculemos el epsilón con python

```
t = 1.0;

while 1+ t != 1:
    eps = t
    t = t/2

print ('el epsilon de la maquina es: ',
   eps)
```

En mi equipo obtengo el siguiente resultado:
El cero de la maquina es:
$2.220446049250313e - 16$

Nuevos conceptos

Una vez que se ha establecido la clasificación del error, definiremos los conceptos de:

- Error absoluto verdadero.
- Error relativo verdadero.
- Error relativo aproximado.

todos ellos como una suma o consecuencia de los errores de redondeo y truncamiento.

Error absoluto verdadero

Supóngase que $\hat{p}$ es una aproximación a p.

El error absoluto verdadero se define con la siguiente expresión:

$$E_v = \left| p - \hat{p} \right|$$

Esta definición de error, lo cuantifica en términos brutos.

No obstante, una medida que puede describir con mayor detalle o proporción el error, es aquella que lo expresa en términos porcentuales.

Para ello se emplea el error verdadero relativo.

Error relativo verdadero

Supóngase que $\hat{p}$ es una aproximación a p. El error relativo verdadero se calcula con la siguiente expresión:

$$e_v = \frac{|p - \hat{p}|}{p}$$

El resultado suele expresarse en términos porcentuales.

Error relativo aproximado

El error relativo aproximado, mide el error de un método numérico, determinando el error de la iteración actual respecto el error surgido en la iteración anterior:

$$e_a = \frac{\left|\widehat{x}_i - \widehat{x}_{i-1}\right|}{\widehat{x}_i}$$

Donde x_i es la aproximación actual a x y x_{i-1} es la aproximación anterior a x.

En métodos numéricos suele establecerse una tolerancia porcentual como criterio de paro, tal que el error relativo aproximado de un método, no exceda dicha tolerancia.

$$e_a < t$$

donde t, es tolerancia fijada de antemano.

A menor tolerancia se tiene mayor precisión en la aproximación al valor verdadero, sin embargo esto implica un aumento en el número de iteraciones requeridas para detener el método.

Contaminación en los cálculos.

Un error en un cálculo numérico “contamina” las sucesivas evaluaciones.

Esta propagación puede describirse en términos de dos conceptos relacionados: los de estabilidad y condición.

Condición

La condición de una función $f(x)$ mide la sensibilidad de los valores de $f(x)$ a pequeños cambios de x, se define como:

$$C = \left| \frac{E_{rel}(f(x))}{E_{rel}(x)} \right|$$

Del teorema del valor medio en cálculo, podemos expresar

$$f(x_T) - f(x_A) \approx f'(x_t)(x_T - x_A) \to E_{rel}(f(x)) \approx$$
$$\approx \frac{f'(x_T)}{f(x_T)}(x_T - x_A)$$

luego

$$C \approx \left| x_T \frac{f'(x_T)}{f(x_T)} \right|$$

Se utilizará ésta definición como definición de condición para funciones $f(x)$ de una variable real.

Entonces los números de condición serán

$$C = \left| x \frac{f'(x)}{f(x)} \right|$$

1. Para un x dado $0 < C(x) < 1$ se dirá que el problema está bien condicionado, y cuanto menor sea C, mejor condicionado.

Se utilizará ésta definición como definición de condición para funciones $f(x)$ de una variable real.

Entonces los números de condición serán

$$C = \left| x \frac{f'(x)}{f(x)} \right|$$

1. Para un x dado $0 < C(x) < 1$ se dirá que el problema está bien condicionado, y cuanto menor sea C, mejor condicionado.
2. Si $C(x) > 1$, el problema estará mal condicionado.

Se utilizará ésta definición como definición de condición para funciones $f(x)$ de una variable real.

Entonces los números de condición serán

$$C = \left| x \frac{f'(x)}{f(x)} \right|$$

1. Para un x dado $0 < C(x) < 1$ se dirá que el problema está bien condicionado, y cuanto menor sea C, mejor condicionado.
2. Si $C(x) > 1$, el problema estará mal condicionado.
3. Si $C(x) = 1$, el error relativo se mantiene.

Ejemplos

Las siguientes funciones están bien condicionadas?

1. $f(x) = \sqrt{x}$ $\qquad C(x) = ?$
2. $g(x) = x^2 - 1$ $\qquad C(x) = ?$

Estabilidad

La estabilidad de un algoritmo describe la sensibilidad de un método numérico específico respecto a los inevitables errores de redondeo cometidos durante su ejecución en aritmética de precisión finita.

Consideremos la siguiente función:

$$f(x) = \sqrt{x+1} - \sqrt{x}$$

Su número de condición es:

$$C(x) = \left| x \frac{f'(x)}{f(x)} \right| = \frac{x}{2\sqrt{x}\sqrt{x+1}}$$

Vemos que $C(x) < \frac{1}{2}$ para $x > 0$, por lo que la función está bien condicionada pero ...

El algoritmo para calcular x de tal forma que se vayan realizando las operaciones, es:

1. obtener x

El algoritmo para calcular x de tal forma que se vayan realizando las operaciones, es:

1. obtener x
2. $y = x + 1$

El algoritmo para calcular x de tal forma que se vayan realizando las operaciones, es:

1. obtener x
2. $y = x + 1$
3. $f = sqrt(y)$

El algoritmo para calcular x de tal forma que se vayan realizando las operaciones, es:

1. obtener x
2. $y = x + 1$
3. $f = sqrt(y)$
4. $g = sqrt(x)$

El algoritmo para calcular x de tal forma que se vayan realizando las operaciones, es:

1. obtener x
2. $y = x + 1$
3. $f = sqrt(y)$
4. $g = sqrt(x)$
5. $h = f - g$

El algoritmo para calcular x de tal forma que se vayan realizando las operaciones, es:

1. obtener x
2. $y = x + 1$
3. $f = sqrt(y)$
4. $g = sqrt(x)$
5. $h = f - g$

El algoritmo para calcular x de tal forma que se vayan realizando las operaciones, es:

1. obtener x
2. $y = x + 1$
3. $f = sqrt(y)$
4. $g = sqrt(x)$
5. $h = f - g$

es inestable para x grandes, dado el paso 5, por lo que debemos de re-estructurar la función.

Eficiencia

Debemos evitar que todo algoritmo sea inestable. Si existieran varios métodos para evaluar una misma función, entonces conviene utilizar aquel que sea más eficiente, es decir, más rápido.

Eficiencia

Debemos evitar que todo algoritmo sea inestable. Si existieran varios métodos para evaluar una misma función, entonces conviene utilizar aquel que sea más eficiente, es decir, más rápido.

Hay que aprovechar al máximo los recursos: hardware, software, algoritmos, para resolver problemas más complejos y no para resolver peor problemas simples.

Por ejemplo, para calcular $x * *4$ para un x dado, no es buena idea calcular $x * *4.0$ (exponente en punto flotante).

La mejor idea consiste en economizar el cálculo en dos pasos:

$$
\begin{aligned}
x2 &= x * x \\
x4 &= x2 * x2
\end{aligned}
$$

y no un producto $x4 = x * x * x * x$

Ejemplo: Evaluación de polinomios

Supongamos que queremos evaluar el polinomio:

$$P(x) = 2 + 4x - 5x^2 + 2x^3 - 6x^4 + 8x^5 + 10x^6$$

Contando con que cada potencia de exponente k entero como $k-1$ productos, tendríamos que el total de productos para evaluar en forma directa es:

$$1 + 2 + 3 + 4 + 5 + 6 = 21$$

Además de seis sumas.

Una mejora en el algoritmo , es calcular primero las potencias de forma sucesiva:

$$\begin{aligned} x^2 &= x * x \\ x^3 &= x * x^2 \\ x^4 &= x * x^3 \\ x^5 &= x * x^4 \\ x^6 &= x * x^5 \end{aligned}$$

De tal forma que se añade un producto por cada potencia, para un total de productos:

$$1 + 2 + 2 + 2 + 2 + 2 = 11$$

Con el polinomio

$$P(x) = 2 + 4x - 5x^2 + 2x^3 - 6x^4 + 8x^5 + 10x^6$$

podemos mejorar el algoritmo de la siguiente manera

$$P(x) = 2 + x\left\{4 + x\left(-5 + x\left[2 + x\left(-6 + x\left\{8 + x * 10\right\}\right)\right]\right)\right\}$$

Para evaluar un polinomio de grado n en el que ninguno de los coeficientes es cero, se necesitan

Para evaluar un polinomio de grado n en el que ninguno de los coeficientes es cero, se necesitan

$\frac{n(n+1)}{2}$	Productos para el primer método
$2n-1$	para el segundo métodos
n	para el tercero

Antes de escribir una línea de código, hay que revisar la manera en que podemos optimizar la solución del problema.

Algoritmo de Horner

Dado el polinomio

$$P(x) = a_0 + a_1 x + \ldots + a_n x^n \qquad a_n \neq 0$$

La evaluación de $P(x)$ para cierto valor de $x = z$ se puede realizar en n pasos mediante:

$$\begin{aligned} b_n &= a_n \\ b_{n-1} &= a_{n-1} + z * b_n \\ b_{n-2} &= a_{n-2} + z * b_{n-1} \\ &\vdots \\ b_0 &= a_0 + z * b_1 \end{aligned}$$

Pseudocódigo

1. $b_n = a_n$

Pseudocódigo

1. $b_n = a_n$
2. repetir mientras $n > 0$

Pseudocódigo

1. $b_n = a_n$
2. repetir mientras $n > 0$
3. $n = n - 1$

Pseudocódigo

1. $b_n = a_n$
2. repetir mientras $n > 0$
3. $n = n - 1$
4. $b = a_n + z * b$

Pseudocódigo

1. $b_n = a_n$
2. repetir mientras $n > 0$
3. $n = n - 1$
4. $b = a_n + z * b$
5. volver al paso 2

Pseudocódigo

1. $b_n = a_n$
2. repetir mientras $n > 0$
3. $n = n - 1$
4. $b = a_n + z * b$
5. volver al paso 2
6. $p(z) = b$

Completa la siguiente tabla

$$P(x) = 2 + 4x - 5x^2 + 2x^3 - 6x^4 + 8x^5 + 10x^6$$

Evalúa el polinomio $P(x)$ en:

x	$P(x)$
-1.5	
-0.65	
0.1	
1.4	
2.87	

$$P(x) = 2 + 4x - 5x^2 + 2x^3 - 6x^4 + 8x^5 + 10x^6$$

El polinomio $P(x)$ evaluado es:

-1.5	$0.781\,25$
-0.65	$-4.506\,83$
0.1	$2.351\,49$
1.4	$98.559\,68$
2.87	$6758.702\,45$

Extendiendo la respuesta al problema

¿Cómo resolver el problema usando una función? ¿mostrando una tabla con formato de salida? usando una gráfica que muestre $P(x)$ y un conjunto de datos evaluados? ¿Evaluar el error relativo?

Extendiendo la respuesta al problema

Ya contamos con las herramientas necesarias para extender la respuesta al problema, en nuestro código podemos agregar funciones, ajustar formatos de salida en los resultados, graficar el polinomio y los puntos (o un conjunto diferente de puntos), y obtener el error relativo. Basta con que le dediquemos un poco más de tiempo.

Pasos a resolver

1. Conviene definir una función que resuelva la evaluación del método de Horner.

Pasos a resolver

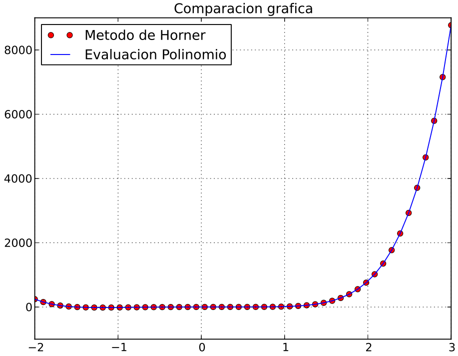
1. Conviene definir una función que resuelva la evaluación del método de Horner.
2. Para obtener el error relativo, se debe de evaluar el polinomio y considerar que los valores obtenidos, son los valores exactos.

Pasos a resolver

1. Conviene definir una función que resuelva la evaluación del método de Horner.
2. Para obtener el error relativo, se debe de evaluar el polinomio y considerar que los valores obtenidos, son los valores exactos.
3. Comparamos los resultados mediante una gráfica que represente los dos resultados de la evaluación.

Resultado gráfico más completo

En la gráfica se muestra un conjunto de datos que se evalúan y posteriormente con la función (en línea continua) se compara, podemos ver que los resultados son prácticamente los mismo.



Estructuramos el código en python .

Definimos mediante dos listas:

1. Los valores donde queremos evaluar el polinomio $P(x)$.

```
# Valores de x0 para evaluar P(x0)
x0=[-1.5, -0.65, 0.1, 1.4, 2.87]

# Coeficientes a de P(x)
a=[2,4,-5,2,-6,8,10]
```

Definimos mediante dos listas:

1. Los valores donde queremos evaluar el polinomio $P(x)$.
2. Los coeficientes de $P(x)$.

```
# Valores de x0 para evaluar P(x0)
x0=[-1.5, -0.65, 0.1, 1.4, 2.87]

# Coeficientes a de P(x)
a=[2,4,-5,2,-6,8,10]
```

Definimos una función que resuelva por Horner.

```
# Metodo de Horner

def P_Horner(x):
    P_Hor=0
    for n in range(len(a)-1,-1,-1):
        P_Hor=a[n]+P_Hor*x
    return P_Hor
```

Definimos una función que evalúe directamente $P(x)$.

```python
# Evaluacion directa

def P_Directo(x):
    return 2+4*x-5*x**2+2*x**3-6*x**4+8*x**
        5+10*x**6
```

Calculamos el error relativo.

```
# Calculo de error relativo

def Err_Rel(p,p_): return (p-p_)/p*100
```

Mostramos el error relativo de los puntos a evaluar.

```
# Evaluacion de valores de P(x0)

for i in range(len(x0)):
    print ("P(%.2f) =" %x0[i],P_Horner(x0[i
        ]), "; Error rel. =", Err_Rel(
        P_Directo(x0[i]),P_Horner(x0[i])))
```

Comparamos los resultados con una gráfica.

```
import matplotlib.pyplot as plt
import numpy as np

x=np.linspace(-2.,3.)

plt.plot(x,P_Horner(x),'ro', label='Metodo de Horner')

plt.plot(x,P_Directo(x), label='Evaluacion Polinomio')

plt.title('Comparacion grafica')
plt.legend(loc='upper left')

plt.grid(True)
plt.show()
```