Propagación de errores
Condición y estabilidad

Curso de Física Computacional

M. en C. Gustavo Contreras Mayén.

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:

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

Del teorema del valor medio en cálculo, podemos expresar

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

$$\approx\frac{f'(x_T)}{f(x_T)}(x_T-x_A)$$

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

Se utilizará ésta definición como definición de condición para funciones *f(x)* de una variable real. Entonces los números de condición serán

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

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

Ejemplos

¿las siguientes funciones están bien condicionadas?

$a.\quad f(x)=\sqrt{x} \qquad C(x)=?$

$b.\quad g(x)=x^2-1 \qquad C(x)=?$

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) < ½ 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:

a. obtener x
b. y = x + 1
c. f = sqrt(y)
d. g = sqrt(x)
e. h = f – g

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

Eficiencia

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

Debemos de aprovechar los recursos para resolver problemas más complejos y no para resolver peor problemas simples.

Por ejemplo, para calcular x**4 para 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:

$$x_2 = x * x, \quad x_4 = x_2 * x_2$$

$$y\ no\ un\ producto\ x_4 = x * x * x * x$$

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:

$$x^2 = x * x, \quad x^3 = x * x^2, \quad x^4 = x * x^3, \quad x^5 = x * x^4, \quad x^6 = x * x^5$$

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

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

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

Pero se puede mejorar más el algoritmo:

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

Que requiere de 6 multiplicaciones (no cambia el número de sumas)

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 método

n para el tercero

Algoritmo de Horner

Dado el polinomio

$$P(x)=a_0+a_1x+\cdots+a_nx^n, \quad a_n\neq 0$$

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

$$(\mathbf{1}) \quad b_n=a_n$$

$$(\mathbf{2}) \quad b_{n-1}=a_{n-1}+z*b_n$$

$$(\mathbf{3}) \quad b_{n-2}=a_{n-2}+z*b_{n-1}$$

$$\vdots$$

$$(\boldsymbol{n}) \quad b_0=a_0+z*b_1$$

Pseudocódigo

$(\mathbf{1})$ $b_n = a_n$

$(\mathbf{2})$ *repetir mientras* $n > 0$

$(\mathbf{3})$ $n = n - 1$

$(\mathbf{4})$ $b = a_0 + z * b$

$(\mathbf{5})$ *volver a* $(\mathbf{2})$

$(\mathbf{6})$ $p(z) = b$

Modelos para el desastre

Un cálculo que utiliza números que se almacenan de manera aproximada en la computadora, puede devolver una solución aproximada.

Para demostrar este hecho, consideremos que hay un valor de incertidumbre, sea x_c el valor representado en la computadora del valor exacto x, tal que:

$$x_c \simeq x\left(1+\epsilon_x\right)$$

$$x_c \simeq x(1+\epsilon_x)$$

Donde ε_x es el error relativo de x_c, el cual se espera que sea similar en magnitud al épsilón de la máquina ε_m. Si usamos ésta notación para una diferencia entre dos valores, tenemos que

$$a = b - c \rightarrow a_c \simeq b_c - c_c \simeq b(1+\epsilon_b) - c(1+\epsilon_c)$$

$$\rightarrow \frac{a_c}{a} \simeq 1 + \epsilon_b \frac{b}{a} - \frac{c}{a}\epsilon_c$$

Vemos que el error resultante en *a* es un promedio de los errores en *b* y *c*, y no hay seguridad en que los dos términos se cancelen.

El error en a_c se incrementa cuando se restan dos valores cercanos ($b \approx c$), dado que cuando se restan las cifras más significativas de ambos números, el error de las cifras menos significativas toma la forma de

$$\frac{a_c}{a} = 1 + \epsilon_a \simeq 1 + \frac{b}{a}(\epsilon_b - \epsilon_c) \simeq 1 + \frac{b}{a} max(|\epsilon_b|, |\epsilon_c|)$$

Ejercicios

1. Separación de sumas.

2. Suma de manera ascendente y descendente.

Errores de redondeo

Ahora veamos lo que sucede cuando calculamos una división sencilla, con dos números:

$$a=\frac{b}{c}\rightarrow a_c=\frac{b_c}{c_c}=\frac{b(1+\epsilon_b)}{c(1+\epsilon_c)},$$

$$\rightarrow\frac{a_c}{a}=\frac{1+\epsilon_b}{1+\epsilon_c}\simeq(1+\epsilon_b)(1-\epsilon_c)\simeq 1+\epsilon_b-\epsilon_c,$$

$$\rightarrow\frac{a_c}{a}\simeq 1+|\epsilon_b|+|\epsilon_c|$$

Se despreciaron valores pequeños de ε^2 y los errores en el valor absoluto nos indican que no correremos con tanta suerte como para conocer el valor de los errores y cancelarlos.

Podemos generalizar el modelo para estimar el error en la evaluación de una función f(x), es decir, la diferencia entre el valor de la función evaluada en x y en x_c

$$\epsilon = \frac{f(x) - f(x_c)}{f(x)} \simeq \frac{\frac{df(x)}{dx}}{f(x)} (x - x_c)$$

Veamos un ejemplo:

$$f(x)=\sqrt{(1+x)}, \qquad \frac{df}{dx}=\frac{1}{2}\frac{1}{\sqrt{(1+x)}}$$

$$\rightarrow \epsilon \simeq \frac{1}{2}\frac{1}{\sqrt{(1+x)}}(x-x_c)$$

Si evaluamos ésta expresión para x=pi/4 y tomamos hasta la cuarta cifra de x, tenemos que el error relativo es 1.5×10^{-4}

Acumulación de errores de redondeo luego de varios pasos

Existe un modelo útil para aproximar qué tanto se acumula el error en un cálculo que se realiza un gran número de veces.

El error que calculamos en una operación, equivale a un "paso" de una caminata aleatoria, más adelante en el tema de Simulaciones Monte Carlo profundizaremos el tema, pero de ahí, sabemos que la distancia total que se recorre en N pasos de longitud r, es en promedio

$$R \simeq \sqrt{(N)}\, r$$

De manera análoga, el error relativo total ε_{ro} luego de N ejecuciones, en donde el épsilon de la máquina es ε_m, es en promedio

$$\epsilon_{ro} \simeq \sqrt{(N)}\,\epsilon_m$$