

Clase 27: Cálculo Numérico

Veremos cómo el computador puede ayudarnos en la solución de problemas numéricos.

1 El computador y la representación de los números

El computador representa los números con una cantidad finita de cifras. En el caso de los números enteros (int), esto implica que solo se trabaja sobre un rango comprendido entre aproximadamente -2.000.000.000 y 2.000.000.000. En los números reales (punto flotante), los números no solo tienen un rango definido, sino que además tienen una precisión finita también. Esto implica errores como que el número 1/3 no puede ser representado en base decimal con un número finito de cifras.

¿Qué importancia tiene esto para un ingeniero? De partida, tiene que tener presente que los cálculos que efectúan con el computador tienen errores, que no siempre son despreciables, y sobre todo, que se acumulan.

En la práctica, por ejemplo, no tiene sentido comparar dos números reales entre sí. Considera el siguiente código, donde el computador debería escribir "ok" en la pantalla.

```
double sexto=1.0/6.0;
double uno=sexto+sexto+sexto+sexto+sexto+sexto;
if (uno==1.0)
    con.println ("ok");
```

Si haces la prueba, verás que no lo hace. El código correcto es

```
final double tolerancia=0.000001; // constante
if (abs(uno-1.0)<tolerancia)
    con.println ("ok");
```

1.1 Error de cancelación

Hay veces en las cuales los errores de redondeo hacen que se pierda precisión en un cálculo. Considera la resta 0.3572-0.3581, cuyo resultado es 0.0009. En los dos números de la resta se tienen 4 cifras significativas, en el resultado normalizado sólo 1. Siempre que se pueda, debe tratarse de evitarse este tipo de operaciones.

2 Encontrar raíces numéricamente

Si uno sabe que una función es continua, que tiene valor negativo en x_1 y valor positivo en x_2 , entonces se puede encontrar un cero mediante el

método de búsqueda binaria.

El método consiste en evaluar la función en $(x_1+x_2)/2$. Si este valor es negativo, x_1 toma el valor $(x_1+x_2)/2$, si no, x_2 toma ese valor, achicando el intervalo donde la raíz se encuentra. Esto se repite hasta que la diferencia entre x_1 y x_2 sea menor a un cierto valor de término o el valor absoluto de la función en la semisuma de x_1 y x_2 sea menor a una cierta tolerancia.¹

```
class Función {
    double valor(double x) {
        ...
    }
    double binaria (double izq, double der) {
        double x1=izq;
        double x2=der;
        boolean izqNegativo=(valor(x1)<0);
        while (Math.abs(x2-x1)>epsilon) {
            double raiz=(x1+x2)/2;
            if (izqNegativo && valor(raiz)<0)
                x1=raiz;
            else
                x2=raiz;
        }
        return raiz;
    }
}
```

Un método más refinado consiste en el método de la secante. En este caso, en vez de promediar x_1 y x_2 se "pesan" los valores de $f(x_1)$ y $f(x_2)$ para iterar.

2.1 Newton-Raphson

Existe un método llamado de Newton-Raphson para encontrar el cero de una función que consiste en calcular la sucesión

$$x_{n+1} \leftarrow x_n - \frac{f(x_n)}{f'(x_n)}$$

donde x_{n+1} es la aproximación que se obtiene para la raíz en la iteración $(n+1)$.

La fórmula de Herón, es un caso particular de Newton-Raphson. Sirve para calcular la raíz cuadrada de un número x . La sucesión es

$$r \leftarrow \frac{r + \frac{x}{r}}{2}$$

¹ Estos son dos criterios distintos, ¿Por qué?

El siguiente programa calcula la raíz cuadrada de un número usando el método de Herón con un error máximo épsilon:

```
double raizCuadrada (double x, double epsilon) {
    double raiz=1.0;
    while (true) {
        raiz=(raiz+x/raiz)/2.0;
        if (Math.abs(raiz-x/raiz)<epsilon)
            break;
    }
    return raiz;
}
```

En el método de Newton Raphson puede no haber convergencia ¿Qué se puede agregar al código para que el computador no se "cuelgue"?

3 Evaluación de polinomios

Al hacer cálculos numéricos hay que fijarse en obtener una buena precisión con la menor cantidad de cálculos posibles. Hay veces que distintos métodos para calcular un mismo resultado demoran tiempos distintos.

Por ejemplo. para calcular el valor del polinomio

$$y(x) = a_3x^3 + a_2x^2 + a_1x + a_0$$

se puede hacer

$$y(x) = a_3x \cdot x \cdot x + a_2x \cdot x + a_1x + a_0$$

Generalizando, para un polinomio de orden n, se obtiene una cantidad de multiplicaciones proporcional a n^2 y una cantidad de sumas igual a n, en cambio, si el polinomio se evalúa astutamente usando la regla de Horner, se hacen sólo n multiplicaciones y n sumas:

$$y(x) = ((a_3x + a_2)x + a_1)x + a_0$$

En Java,

```
function horner (double a[], int n) {
    double sum=a[n];
    for (int i=n-1;i>=0;i--)
        sum=sum*x+a[i];
    return sum;
}
```

4 Aproximación de series infinitas

Muchas funciones matemáticas pueden ser representadas por series infinitas de terminos. Por ejemplo,

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

$$\sin x = \frac{x}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \dots$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \dots$$

$$\lg(1+x) = \frac{x^1}{1} - \frac{x^2}{2} + \frac{x^3}{3} - \dots$$

En las series para seno y coseno x está en radianes, en la serie para $\log(1+x)$ se obtienen resultados válidos sólo para valores absolutos de x menores a 1 . Por ejemplo, la siguiente función calcula el seno de x usando 7 términos:

```
double xcua=x*x;
double term=x;
double sine=x;
for (int i=1;i<8;i++) {
    term=-(term*xcua)/(2*i*(2*i+1));
    sine=sine-(term*xcua)/(2*i*(2*i+1));
}
con.println (sine);
```

5 Integración numérica

Existe un método sumamente simple para calcular áreas: el método del trapecio. Supón que se tiene una curva $y=f(x)$, y que se quiere encontrar el area entre la curva y el eje de las abscisas entre las coordenadas $x=x_1$ y $x=x_n$.

Si se divide la distancia entre x_1 y x_n en intervalos de tamaño delta, tal que $\text{delta}=(x_n-x_1)/(n-1)$, se puede aproximar el área total mediante la suma de trapecios:

$$(y_1+y_2)*\text{delta}/2+(y_2+y_3)*\text{delta}/2+\dots+(y_{n-1}+y_n)*\text{delta}/2;$$

Factorizando queda

$$((y_1+y_n)/2+y_2+y_3+\dots+y_{(n-1)})\cdot\delta.$$

En Java, es tan simple como

```
double suma=(y[0]+y[n-1])/2;
for (int i=1;i<n-1;i++)
    suma+=y[i];
double area=suma*delta;
```

5.1 El método de Simpson

En el método del trapecio una aproximaba la curva mediante trapecios. El "Método de Simpson" asume que cada par de "tajadas" adyacentes de la curva tiene la forma de una parábola.

El área de cada par de tajadas es delta por un tercio de la suma de los valores de la función en los extremos más cuatro veces el valor en el medio.

```
double f[]= new double [7];
double delta = Math.PI/6;
for (i=0;i<7;i++)
    f[i]=Math.sin(i*delta);
double par=f[1]+f[3]+f[5];
double impar=f[2]+f[4];
double simp=delta+(f[0]+4*par+2*impar+f[6]))/3;
con.println ("metodo de simpson "+simp);
```

6 Solución a un sistema de ecuaciones lineales

Con un computador es bastante facil solucionar un sistema de ecuaciones lineales del estilo

$$Ax=b$$

Uno puede guardar un sistema de ecuaciones en tres arreglos, una matriz con los coeficientes "A", un vector con la incógnita y otra matriz con los coeficientes del vector solución.

$$\begin{aligned} a[0,0]x_0+ a[0,1]x_1+ a[0,2]x_2+ a[0,3]x_3 &=b[0] \\ a[1,0]x_0+ a[1,1]x_1+ a[1,2]x_2+ a[1,3]x_3 &=b[1] \\ a[2,0]x_0+ a[2,1]x_1+ a[2,2]x_2+ a[2,3]x_3 &=b[2] \\ a[3,0]x_0+ a[3,1]x_1+ a[3,2]x_2+ a[3,3]x_3 &=b[3] \end{aligned}$$

Esta matriz debe convertirse en una triangular superior

$$\begin{array}{cccc} a(0,0) & a(0,1) & a(0,2) & a(0,3) \\ 0 & a(1,1) & a(1,2) & a(1,3) \\ 0 & 0 & a(2,2) & a(2,3) \\ 0 & 0 & 0 & a(3,3) \end{array}$$

De esta matriz se puede despejar inmediatamente $x[3]$. Con este valor, se puede despejar $x[2]$, luego $x[1]$ y finalmente $x[0]$.

7 Método de los mínimos cuadrados

Considera la situacion donde las medidas de una cantidad dependiente "y" son hechas para ciertos valores de una variable independiente x, y se sabe de la teoría que la relación entre x e y es lineal, es decir,

$$y = ax + b$$

El método de los mínimos cuadrados consiste en buscar valores para a y b tales que el cuadrado de las distancias de los puntos (x_i, y_i) a la recta $y = ax + b$ sea mínima.

El cuadrado de la distancia de un punto (x_i, y_i) a la recta es

$$(y_i - (ax_i + b))^2$$

por lo que el problema se reduce a minimizar la expresión

$$\sum (y_i - (ax_i + b))^2$$

Diferenciando parcialmente con respecto a las dos variables, a y b, se obtiene el siguiente sistema de ecuaciones:

$$\sum x_i a + nb = \sum y_i$$

$$\sum x_i^2 a + \sum x_i b = \sum x_i y_i$$

de donde se puede despejar fácilmente a y b.

8 Uso de funciones "genéricas"

Mezclar el uso de enlace dinámico con la computación numérica es algo natural y muy provechoso, pues permite crear código genérico, como ya se vio en el caso de encontrar raíces.

Para trabajar con una función particular basta extender la clase y redefinir el método valor.

```
class Coseno extends Funcion {  
    double valor (double d) {  
        return Math.cos(d);  
    }  
}  
class Seno extends Funcion {  
    double valor (double d) {  
        return Math.sin(d);  
    }  
}
```