

Clase adicional 7

Temas

- Análisis numérico en Java
- Búsqueda de raíces
- Integración
- Ejercicios de la clase adicional
- Ejercicios de diseño

Análisis numérico en Java

El análisis numérico es una rama de la matemática aplicada que se dedica a crear y evaluar técnicas para el empleo de ordenadores en la resolución de problemas numéricos y a estudiar su convergencia y errores.

Considere crear una función que busque la raíz de una función , $f(x) = 0$, en un intervalo $[a,b]$. Este método de búsqueda de raíces debería tener la función $f(x)$ como argumento. En Java, dado que (casi) todo debe ser un objeto, esto se realiza empaquetando la función que se desea pasar en un objeto. Posteriormente, se puede pasar el objeto como un argumento al método que busca la raíz.

El empaquetado se suele realizar declarando una interfaz que describe el objeto que contiene una función. Una interfaz es como un modelo que especifica una o varias capacidades (métodos) que se deben definir en cada clase que lo "implementa". Sólo se pueden definir como interfaces los prototipos de función (los cuerpos de estos métodos se pueden definir en las clases que implementan una interfaz concreta).

Por ejemplo, vamos a declarar la siguiente interfaz:

```
interface MathFunction
{
    public double func(double x);
}
```

Posteriormente, podemos utilizar esta interfaz para declarar una clase con una función concreta:

```
class Function1 implements MathFunction
{
    public double func(double x)
    {
        return x * x - 2; //definición para una función concreta,
        //aquí, por ejemplo, representa  $f(x) = x^2 - 2$ ;
    }
}
```

Ahora ya podemos definir nuestra función de búsqueda de raíces del siguiente modo:

```
public class RootFinder
{
    public static double rootFindingMethod( MathFunction mathf, ...)
    {
        // busca la raíz para una función concreta
        ...
    }
}
```

... y llamar a la función del siguiente modo con

```
Function1: double rootResult = RootFinder.rootFindingMethod(new Function1(),
...);
```

Por ejemplo, el siguiente código buscará la raíz para $f(x) = x^2 - 2$ utilizando el método de bisección definido en el siguiente apartado (suponga que el método de bisección está definido en la clase `RootFinder`): // método `main()` de la clase `RootFinder`

```
public static void main(String[] args) {
    double root= RootFinder.bisect(new Function1(), -1.0, 2.0, 0.0001);
    System.out.println("Raíz: " + root);
    System.exit(0); } }
```

Búsqueda de raíces

Existen cuatro métodos "elementales" para buscar raíces. Entre ellos, el método de bisección y el método de Newton se pueden utilizar para resolver problemas reales en determinadas circunstancias. Los métodos de la secante y de posición falsa (regula falsi) tienen únicamente un valor pedagógico. Consulte la referencia de fórmulas de C para analizar éstos y otros métodos si realmente necesita buscar raíces; las primeras páginas del texto de cada método son relativamente legibles y reveladoras.

En el caso de los métodos de bisección, secante y posición falsa, ofrecemos el razonamiento que se esconde detrás de cada uno de ellos y un fragmento de pseudocódigo que lo implementa. Este código es meramente ilustrativo, no es fiable y no debe utilizarse para problemas reales.

Bisección

Comience con un intervalo conocido para acotar una raíz.
Divídalo por la mitad, para reducir el intervalo a la mitad conocida y contener una raíz. Repita esta acción hasta que el intervalo sea lo suficientemente pequeño

```

public static double bisect( MathFunction f, double x1, double x2, double epsilon)
{
    double m;
    for (m= (x1+x2)/2.0; Math.abs(x1-x2)> epsilon; m= (x1+x2)/2.0)
        if (f.func(x1)*f.func(m) <= 0.0)
            x2= m;    // Utiliza el subintervalo izquierdo
        else
            x1= m;    // Utiliza el subintervalo derecho
    return m;
}

```

Secante

Uno de los métodos de búsqueda de raíces, el método secante, comienza con un intervalo conocido para contener una raíz. Llama a los puntos finales del intervalo x_1 y x_2 . El método funciona ajustando una línea recta entre los dos puntos actuales (x_1 , $f(x_1)$) y (x_2 , $f(x_2)$) y buscando el punto x^* en el que la línea interseca el eje x . El punto final x_1 se restablece con el valor de x_2 . Por su parte, el punto x^* pasa a ser el nuevo punto final (x_2). Este procedimiento se repite hasta que, o bien el intervalo es muy pequeño, o si se encuentra una raíz de la función. A continuación se incluye el código para implementar el método.

comenzar con x_1 y x_2
ajustar una línea recta entre $f(x_1)$ y $f(x_2)$
buscar x^* , la raíz de la línea recta
si x^* se aproxima a x_2 , devolver el valor como la raíz de la ecuación original
De lo contrario, restablecer el valor de x_1 con x_2 y x_2 con x^* e intentar ejecutar el método de nuevo.

```

public static double secant(MathFunction f, double x1, double x2, double epsilon)
{
    double root = x1;
    double y1, y2;
    while (Math.abs(x2 - x1) > epsilon)
    {
        y1 = f.func(x1);
        y2 = f.func(x2);
        root = x1-y1*(x2-x1)/(y2-y1);
        x1 = x2;
        x2 = root;
    }
}

```

```
    return root;
}
```

Posición falsa (regula falsi)

El método de posición falsa es prácticamente igual que el método secante. La diferencia reside en que cuando el punto x^* pasa a ser uno de los puntos finales del siguiente intervalo, el otro punto final es cualquier de los x_1 y x_2 que garantice que el nuevo intervalo sigue conteniendo una raíz de la función, esto es, el método de posición falsa selecciona el punto "previo" para mantener acotado el cero en lugar de utilizar siempre sólo los dos últimos puntos.

- comenzar con x_1 y x_2
- ajustar una línea recta entre $f(x_1)$ y $f(x_2)$
- buscar x^* , la raíz de la línea recta
- si $f(x^*)$ se aproxima a 0, devolver el valor x^* como la raíz de la ecuación original
- De lo contrario, restablecer (x_1, x_2) en (x_1, x^*) o en (x^*, x_2) (el nuevo intervalo que contiene una raíz de la función) e intentar ejecutar el método de nuevo.

```
public static double regulaFalsi(MathFunction f, double x1, double x2, double epsilon)
{
    double y1, y2, root, yroot;
    do {
        y1 = f.func(x1);
        y2 = f.func(x2);
        root = x1 - y1 * (x2 - x1) / (y2 - y1);
        yroot = f.func(root);
        if (yroot * y2 < 0.0)
            x1 = root;
        else
            x2 = root;
    } while (Math.abs(yroot) > epsilon);

    return root;
}
```

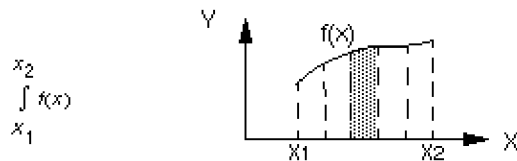
Newton

- Iniciar en el mismo punto x del eje x
- ajustar una línea tangente a $f(x)$ en x
- obtener la solución para el punto y en la intersección de la tangente con el eje x
- repetir el proceso hasta que $|x-y| < \epsilon$, y devolver y

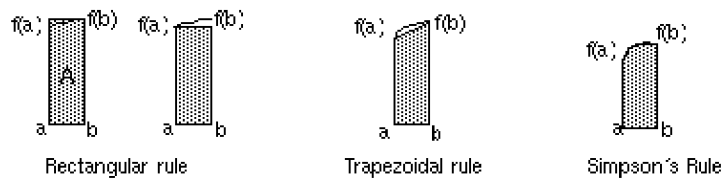
Este código está disponible en el material de clase.

Integración

La integración numérica se utiliza en el cálculo de integrales definidas. Los métodos elementales de integración numérica se basan en el sumatorio de todas las áreas pequeñas (secciones) de la curva de una función, y entre dos límites x_1 y x_2 .



Según se calcule cada área del segmento, obtendremos los siguientes métodos elementales o pedagógicos:



Regla rectangular

El método se basa en las áreas de rectángulos. El área de cada uno se calcula como: $A = f(a)\Delta x$ o $A = f(b)\Delta x$, donde $\Delta x = (b - a)$ es el área del segmento A.

Regla trapezoidal

Este método es igual de sencillo que el de la regla rectangular, pero a menudo resulta mucho más preciso. Está basado en las áreas de trapecoides. El área de cada segmento A se calcula como:

$$A = \left(\frac{f(a) + f(b)}{2} \right) \Delta x$$

Regla de Simpson

La regla de Simpson utiliza 3 puntos para aproximar la función con un polinomio de segundo grado. El área de cada segmento se calcula como:

$$A = \left(f(a) + 4f\left(a + \frac{\Delta x}{2}\right) + f(b) \right) \frac{\Delta x}{6}$$

La regla de Simpson es un método muy utilizado en la enseñanza de integración numérica para integrales definidas. Aproxima la función mediante el ajuste de parábolas a cada segmento y posteriormente suma todas las áreas de cada segmento para calcular el área de la curva. El siguiente código implementa una versión elemental de la regla de Simpson:

```
// Regla de Simpson
public static double simpson(MathFunction f, double a, double b, int nPanels)
{
    double answer = 0;
    double x = a;
```

```

double h = (b - a) / nPanels;
for(int i=1; i <= nPanels; i++)
{
    answer += (f.func(x) + 4 * f.func(x + h / 2) + f.func(x + h)) / 6;
    x = a + i * h;
}
return h * answer;
}

```

Para resolver problemas reales, debe utilizar la ampliación de la regla de Simpson, que es la versión utilizable más sencilla. El método ampliado de Simpson se trata en la Clase número 20 del material de clase.

Ejercicios de la clase adicional

1. Dadas dos funciones, $f(x)$ y $g(x)$, ¿cómo se puede calcular el valor de x donde $f(x) = g(x)$?
2. ¿Cómo se calcularía si los puntos x_1 y x_2 acotan la raíz de la función $f(x)$?
3. ¿Cuándo se utilizan los métodos de búsqueda de raíces? ¿Cuándo se utilizan los métodos de integración numérica?
4. Esboce el método de bisección para la función $f(x) = x^2 - 5x + 4$, empezando desde $a=0$ y $b=3$ para un par de iteraciones.
5. ¿Por qué se utiliza la derivada de $f(x)$ al utilizar el método de Newton para buscar la raíz de $f(x)$?

Ejercicios de diseño

Realice el diseño de las tres clases siguientes: RootFinder, Integration y NumericalTest. RootFinder implementará versiones elementales de los tres métodos de búsqueda de raíces (bisección, secante y posición falsa de los apuntes de las clases adicionales, no del material de clase). Integration implementará versiones elementales de los tres métodos de integración (rectangular, trapezoidal y Simpson). Y NumericalTest contará con un método main para probar los tres métodos de búsqueda de raíces y buscará la raíz para:

$f_1(x) = x^5 + 4x^3 + 7x - 10$; (buscar la raíz en el intervalo $-1.0, 2.0$, con $\epsilon = 0.0001$)

y probará los tres métodos de integración con la integral para la función:

$f_2(x) = x^6 + 7x^5 + 3x^4 + x^2 - 5x + 4$; (integrando desde -5.0 a 8.0 , con $n = 1000$)

Ejecute el programa y compare los resultados de estos métodos.

Sugerencia: deberá crear dos clases para representar $f_1(x)$ y $f_2(x)$. Cree estas clases implementando la interfaz `MathFunction`.