Simple definite integral solver – Resolvedor simple de integral definida

Gabriel Nicolás González Ferreira
Instituto Superior de Formación Técnica N° 151 (ISFT 151)
gabrielinuz@fi.mdp.edu.ar

Abstract

Este documento consiste en una resumida explicación del diseño y desarrollo de un resolvedor simple de integral definida, está confeccionado en carácter de trabajo final para la Materia Análisis Matemático 2 a cargo de los profesores Julio Riera y Anabella Di Marco.

1. Introducción

Para la realización de este trabajo se eligió el lenguaje de programación c++ ya que este lenguaje apoya el paradigma orientado a objetos (POO) sin sacrificar la eficiencia de las aplicaciones obtenidas. El lenguaje permitió utilizar además un modelo simplificado de componentes de software, haciendo uso como formato binario bibliotecas compartidas (Shared Object ó Dynamic Link Libraries) mediante un cargador multiplataforma.

2. Reglas de Resolución de Integrales Definidas aplicadas

Se utilizaron dos reglas para resolver integrales: Regrla del Trapecio Compuesta y la Regla de Simpson 1/3 compuesta.

2.1 Regla del Trapecio Compuesta

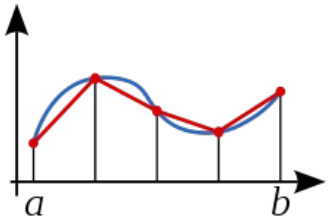


Ilustración de la regla del trapecio compuesta: La función f(x) (en azul) es aproximada por la función lineal (en rojo).

La regla del trapecio compuesta o regla de los trapecios es una forma de aproximar una integral definida utilizando n trapecios. En la formulación de este método se supone que f es continua y positiva en el intervalo [a,b]. De tal modo la integral definida:

$$\int_a^b f(x)\,dx$$

representa el área de la región delimitada por la gráfica de f y el eje x, desde x=a hasta x=b. Primero se divide el intervalo [a,b] en n subintervalos, cada uno de ancho

$$\Delta x = (b-a)/n$$

Después de realizar todo el proceso matemático se llega a la siguiente fórmula:

$$\int_a^b f(x)\,dx \sim \frac{h}{2}[f(a) + 2f(a+h) + 2f(a+2h) + \ldots + f(b)]$$

Donde $h = \frac{b-a}{n}$ y n es el número de divisiones.

La expresión anterior también se puede escribir como:

$$\int_a^b f(x)dx \sim \frac{b-a}{n}\left[\frac{f(a)+f(b)}{2} + \sum_{k=1}^{n-1} f\left(a + k\frac{b-a}{n}\right)\right]$$

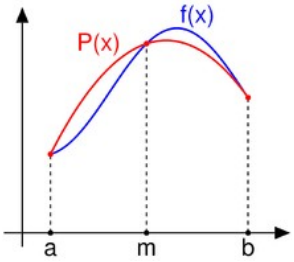
El error en esta aproximación se corresponde con :

$$-\frac{(b-a)^3}{12n^2} f^{(2)}(\xi)$$

Siendo ξ un número perteneciente al intervalo [a, b] y n el número de subintervalos.

Es importante aclarar que este método no presentará ningún error en el cálculo (resultado exacto) si la función sujeta a integración es lineal.

2.2 Regla de Simpson



La función f(x) (azul) es aproximada por una función cuadrática P(x) (rojo).

En análisis numérico, la regla o método de Simpson, nombrada así en honor a Thomas Simpson (y a veces llamada regla de Kepler), es un método de integración numérica para obtener el valor aproximado de integrales definidas; específicamente es la aproximación:

$$\int_a^b f(x)\,dx \approx \frac{b-a}{6}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right]$$

A comparación de la regla del trapecio, esté método de integración resulta ser más exacto, ya que se utilizan polinomios de segundo o tercer grado para su aproximación.

2.3 Regla de Simpson 1/3 Compuesta

La fórmula compuesta de Simpson o de segmentos múltiples consiste en dividir el intervalo $[a, b]$ en n subintervalos iguales (con n par), de manera que $h = (b-a)/n$ Entonces la regla compuesta viene dada por:

$$I \approx \frac{h}{3}[f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x_4) + \ldots + f(x_n)]$$

O en su forma general:

$$I \approx \frac{h}{3}\left[f(x_0) + 4\sum_{i=1,3,5,\ldots}^{n-1} f(x_i) + 2\sum_{j=2,4,6,\ldots}^{n-2} f(x_j) + f(x_n)\right]$$

Cálculo del error:

$$E_v = -\frac{b-a}{180}h^4 f^{(4)}(\xi)$$

ξ se debe encontrar dentro del intervalo [a, b].

Se debe notar que para el cálculo del error es necesario obtener la cuarta derivada de la función, pero, si la función fuera un polinomio de tercer grado esta derivada sería 0, entonces el error también sería del 0%, por lo tanto la regla de Simpsons obtiene la integral exacta para polinomios de tercer grado.

3. Concepto de componente de software

Hay un consenso general en la comunidad de que un componente es una unidad de software independiente que puede estar compuesta por otros componentes y que se utiliza para desarrollar un sistema de software. Aparte de esto, sin embargo, distintas personas han propuesto definiciones de un componente software. Councill y Heineman (Councill y Heineman, 2001) definen un componente como:

"Un elemento software que se ajusta a un modelo de componentes y que puede ser desplegado y compuesto de forma independiente sin modificación según un estándar de composición."

La visión de un componente como un proveedor de servicios resalta dos características críticas de un componente reutilizable:

1. El componente es una entidad ejecutable independiente. El código fuente no está disponible, por lo que el componente no tiene que ser compilado antes de que sea usado con otros componentes del sistema.
2. Los servicios ofertados por un componente están disponibles a través de una interfaz, y todas las interacciones son a través de esa interfaz. La interfaz del componente se expresa en términos de operaciones parametrizadas y su estado interno nunca se muestra.
3. Los componentes son entidades desplegarles. Es decir, no son compilados en un programa de aplicación, sino que se instalan directamente sobre una plataforma de ejecución. Los métodos y atributos definidos en sus interfaces pueden ser accedidos por otros componentes.
4. Los componentes no definen tipos. Una definición de clase define un tipo abstracto de datos y los objetos son instancias de ese tipo. Un componente es una instancia, no una plantilla que se utiliza para definir una instancia.
5. Las implementaciones de los componentes son opacas. Los componentes están, al menos en principio, completamente definidos por la especificación de su interfaz. La implementación está oculta para los usuarios de los componentes. Los componentes a menudo se entregan como unidades binarias de forma que el comprador no tiene acceso a la implementación.
6. Los componentes son independientes del lenguaje. Las clases de objetos tienen que

seguir las reglas de un lenguaje de programación orientado a objetos particular y, generalmente, sólo pueden interoperar con otras clases escritas en dicho lenguaje. Si bien los componentes se implementan normalmente utilizando lenguajes orientados a objetos tales como Java, pueden implementarse en lenguajes de programación no orientados a objetos.

7. Los componentes están estandarizados. A diferencia de las clases de objetos que pueden implementarse de cualquier forma, los componentes deben ajustarse a algún modelo de componentes que restringe su implementación.

3.1. Diagrama de Componentes

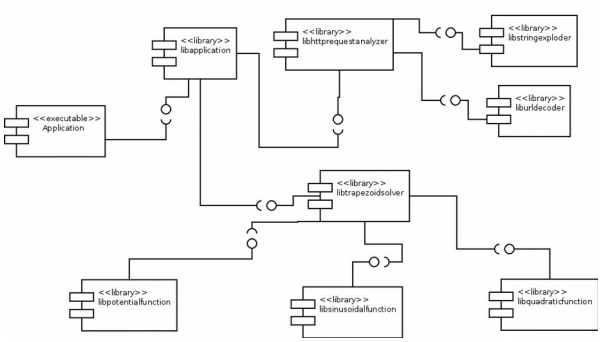


Diagrama de componentes sin paquetes del resolvedor simple de integral definida. (Ver en Apéndice página 5)

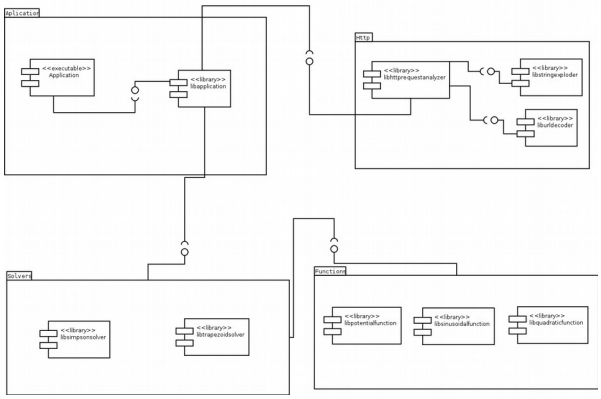


Diagrama de componentes con paquetes del resolvedor simple de integral definida. (Ver en Apéndice página 6)

4. Interfaces del sistema

Las interfaces son una especie de colección de métodos abstractos, presentados como una clase. En las interfaces se especifica qué se debe hacer pero no su implementación. Serán las clases que implementen estas interfaces las que describan la lógica del comportamiento de los métodos. A continuación se detalla el código de las principales interfaces usadas en el sistema:

```
/*Aplication.h*/

#ifndef APPLICATION_H
#define APPLICATION_H
#include <iostream>
using namespace std;

class Application
{
    public:
    virtual ~Application(){}
    virtual int run() = 0;
};

#endif // APPLICATION_H

/*Component*/
#ifndef COMPONENT_H
#define COMPONENT_H
#include <string>

class Component
{
  public:
  virtual ~Component(){}

  virtual bool
  implements(std::string
  interfaceName) = 0;

  virtual void* getInstance() = 0;
  virtual void release() = 0;
};

#endif // COMPONENT_H

/*Integral Solver*/
#ifndef INTEGRALSOLVER_H
#define INTEGRALSOLVER_H

#include <Compset/Function.h>
#include <iostream>
#include <unordered_map>
#include <list>
#include <string>

using namespace std;

class IntegralSolver
{
  public:
  virtual ~IntegralSolver(){}

  virtual list<string>
  getParameterList() = 0;

  virtual void
  setFunction(Component* function) =
  0;
```

```
  virtual double
  solve(unordered_map<string,
  string> parameters) = 0;
 };

#endif // INTEGRALSOLVER_H
```

/*Function.h*/

```
#ifndef FUNCTION_H
#define FUNCTION_H

#include <iostream>
#include <list>
#include <unordered_map>
#include <string>

using namespace std;

class Function
{
  public:
  virtual ~Function(){}

  virtual list<string>
  getParameterList() = 0;

  virtual double
  solve(unordered_map<string,
  double> parameters, double x) = 0;
};

#endif // FUNCTION_H
```

/*RequestAnalyzer.h*/

```
#ifndef REQUESTANALYZER_H
#define REQUESTANALYZER_H

#include <Compset/Component.h>

#include <unordered_map>
#include <string>
#include <iostream>

using namespace std;

class RequestAnalyzer
{
   public:
   virtual ~RequestAnalyzer() {}

   virtual void
   setExploder(Component*
   exploder_component) = 0;

   virtual void setDecoder(Component*
   decoder_component) = 0;

   virtual unordered_map<string,
   string> getRequestMap() = 0;
};
```

```
#endif // REQUESTANALYZER_H
```

5. Lenguajes de maquetación, estilo y lógica de interfaz

Para la el desarrollo de la interfaz gráfica se seleccionaron los siguientes lenguajes:

- Para la maquetación, HTML HyperText Markup Language («lenguaje de marcas de hipertexto»), hace referencia al lenguaje de marcado para la elaboración de páginas web.
- Para el diseño, CSS ó Hoja de estilo en cascada (siglas en inglés de cascading style sheets) que es un lenguaje usado para definir y crear la presentación de un documento estructurado escrito en HTML o XML2 (y por extensión en XHTML).
- Para la lógica de la interfaz, JavaScript (abreviado comúnmente "JS") es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos,3 basado en prototipos, imperativo, débilmente tipado y dinámico.

5. Bibliografía consultada

Stroustrup, Bjarne, *El Lenguaje de Programación C++* , 2da edición Addison Wesley, 7 Jacob Way, Massachusetts, 1993.

Sommerville, Ian, *Ingeniería de Software*, Septima Edición Addison Wesley, 7 Jacob Way, Massachusetts, 1993.

Regla del trapecio. (2015, 3 de agosto). *Wikipedia, La enciclopedia libre*. Fecha de consulta: 06:12, noviembre 26, 2015 desde https://es.wikipedia.org/wiki/Regla_del_trapecio

Regla de Simpson. (2015, 5 de noviembre). *Wikipedia, La enciclopedia libre*. Fecha de consulta: 06:09, noviembre 26, 2015 desde https://es.wikipedia.org/wiki/Regla_de_Simpson

HTML. (2015, 25 de noviembre). *Wikipedia, La enciclopedia libre*. Fecha de consulta: 06:20, noviembre 26, 2015 desde https://es.wikipedia.org/wiki/HTML

Hoja de estilos en cascada. (2015, 23 de noviembre). *Wikipedia, La enciclopedia libre*. Fecha de consulta: 06:20, noviembre 26, 2015 desde https://es.wikipedia.org/wiki/Hoja_de_estilos_en_cascada

JavaScript. (2015, 15 de noviembre).*Wikipedia, La enciclopedia libre*. Fecha de consulta: 06:20, noviembre 26, 2015 desde https://es.wikipedia.org/wiki/JavaScript