FUNDAÇÃO DE ENSINO “EURÍPIDES SOARES DA ROCHA”
CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA – UNIVEM
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

EDMILTON OSÉIAS DA CUNHA

IMPLEMENTAÇÃO DE UM SOFTWARE PARA FAZER COMPARAÇÃO DE RESULTADOS DO CÁLCULO DE ÁREAS USANDO SOMA DE RIEMANN

MARÍLIA
2008

EDMILTON OSÉIAS DA CUNHA

IMPLEMENTAÇÃO DE UM SOFTWARE PARA FAZER COMPARAÇÃO DE RESULTADOS DO CÁLCULO DE ÁREAS USANDO SOMA DE RIEMANN

Trabalho de Curso apresentado ao Curso de Bacharelado em Ciência da Computação da Fundação de Ensino "Eurípides Soares da Rocha", mantenedora do Centro Universitário Eurípides de Marília – UNIVEM, como requisito parcial para obtenção do grau de Bacharelado em Ciência da Computação.

Orientadora:
Profa. Dra. JULIANA DE OLIVEIRA.

MARÍLIA
2008

CUNHA, Edmilton Oséias da

Implementação de um software para fazer comparação de resultados do cálculo de áreas usando Soma de Riemann / Edmilton Oséias da Cunha; orientadora: Juliana de Oliveira. Marília, SP: [s.n.], 2008.

50 f.

Trabalho de Curso (Graduação em Bacharelado em Ciência da Computação) - Fundação de Ensino "Eurípides Soares da Rocha", mantenedora do Centro Universitário Eurípides de Marília – UNIVEM.

1. Uml 2. Soma de Riemann.

CDD: 005.1003

CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

TRABALHO DE CONCLUSÃO DE CURSO – AVALIAÇÃO FINAL

Edmilton Oséias da Cunha

IMPLEMENTAÇÃO DE UM SOFTWARE PARA FAZER COMPARAÇÃO DE RESULTADOS DO CÁLCULO DE ÁREAS USANDO SOMA DE RIEMANN

Banca examinadora da monografia apresentada ao Curso de Bacharelado em Ciência da Computação do UNIVEM/F.E.E.S.R., para obtenção do Título de Bacharel em Ciência da Computação.

Nota: 9,0 (Nove)

Orientador: Juliana de Oliveira — Juliana de Oliveira

1º. Examinador: Maria Christina Amendola Almeida

2º. Examinador: Paulo Augusto Nardi

Marília, 11 de novembro de 2008.

AGRADECIMENTOS

À Juliana de Oliveira e ao Paulo Augusto Nardi pelo direcionamento deste trabalho e a todos aqueles que me auxiliaram nesta etapa da minha vida e à força divina que me inspirou energias para a conclusão deste ciclo.

E por fim, àqueles que me trouxeram à vida, àqueles que em companhia de mim cresceram e àquela que a compartilhará comigo para todo o sempre.

CUNHA, Edmilton Oséias da. **Implementação de um software para fazer comparação de resultados do cálculo de áreas usando Soma de Riemann.** 2008. 50f. Trabalho de Curso (Bacharelado em Ciência da Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino "Eurípides Soares da Rocha", Marília, 2008.

RESUMO

Este trabalho tem por objetivo apresentar uma visão abrangente da Soma de Riemann, modelar em UML e implementar em Java um software para calcular áreas em quaisquer curvas para uma ou mais funções predeterminadas. Para gerar os gráficos do software foi usada a API JFreeChart. Por fim foi feito o estudo de caso da modelagem do software e o teste de resultados do mesmo com o software matemático Maple com a finalidade de verificar a correspondência dos resultados.

Palavras-Chave: UML, Soma de Riemann.

CUNHA, Edmilton Oséias da. **Implementação de um software para fazer comparação de resultados do cálculo de áreas usando Soma de Riemann.** 2008. 50f. Trabalho de Curso (Bacharelado em Ciência da Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino "Eurípides Soares da Rocha", Marília, 2008.

ABSTRACT

This paper presents a comprehensive view of Riemann's Sum, to model in UML and implement in Java a software to calculate areas in any curves for one or more predetermined functions. It was used JFreeChart API to generate the graphics of the software. Finally, was done the case study of modeling software and the test of the results obtained with this software using the Maple mathematical software in order to verify the correlation of results.

Keywords: UML, Riemann's Sum.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

UML – Unified Modeling Language (Linguagem de Modelagem Unificada)

LISTA DE TABELAS

SUMÁRIO

INTRODUÇÃO

O cálculo de áreas é um conteúdo importante e faz parte de algumas disciplinas dos cursos superiores na área de Exatas. É simples quando precisamos obter a área de um retângulo ou de um triângulo, porém fica complexo quando é uma área de uma região com lados curvos (STEWART, 2006, p.367).

Estudaremos a Soma de Riemann que tem o objetivo de diminuir esta complexidade, pois particiona a área em retângulos e soma a área de todos estes retângulos. Abordaremos a Soma de Riemann nos casos da aproximação pelos pontos à esquerda, à direita, médio e randômico.

Este trabalho tem por objetivo modelar um software nos diagramas Casos de Uso, Classes, Seqüência e Componentes da UML (Linguagem de Modelagem Unificada) e implementá-lo na linguagem Java para calcular áreas em quaisquer curvas para uma ou mais funções predeterminadas. Para gerar os gráficos é usada a API JFreeChart.

Por fim é usado o software matemático Maple para fazer comparações de resultados obtidos com o software desenvolvido e, com isto saber a precisão de cálculo deste e a credibilidade em usá-lo.

Este trabalho está dividido nos seguintes capítulos:

O primeiro capítulo – Cálculo de áreas usando a Soma de Riemann – descreve quem foi Riemann e sua importante contribuição para a matemática com a Soma de Riemann. É feita uma descrição de como calcular a Soma de Riemann.

O segundo capítulo – Diagramas UML – apresenta os diagramas Casos de Uso, Classes, Seqüência e Componentes da UML que serão usados no desenvolvimento do software.

O terceiro capítulo – Desenvolvimento do software para calcular áreas por meio da Soma de Riemann – mostra a modelagem do software com a UML. Neste capítulo também é apresentada as funcionalidades do software e realizada as comparações com o Maple.

Por fim são apresentadas a Conclusão, as Referências usadas para a realização do trabalho e um apêndice com os métodos da Soma de Riemann e da geração de gráfico combinado.

CAPÍTULO 1 – CÁLCULO DE ÁREAS USANDO A SOMA DE RIEMANN

1.1 Introdução

GEORG FRIEDRICH BERNHARD RIEMANN nasceu em 17 de setembro de 1826 em Breselenz/Dannenberg na Alemanha. Em 1842 ingressou na escola de Johanneum na cidade de Lüneburg e mostrou suas habilidades em matemática e física. Seu pai era vigário e isto influenciou Riemann a estudar teologia e filosofia na Universidade de Göttingen. Porém, após realizar algumas conferências matemáticas, Riemann conseguiu mudar a opinião de seu pai e, em 1846 se transferiu para o curso de matemática. Em 1851 recebeu seu Ph. D sob a orientação do matemático Karl Friedrich Gauss e começou a lecionar em Göttingen. Riemann faleceu de tuberculose em 20 de julho de 1866 com 39 anos de idade (ANTON, 2002, p.410; HAFTENDORN, 2008; STEWART, 2006, p.379).

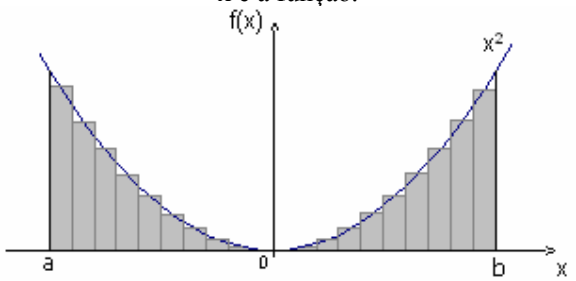
Riemann deixou importantes trabalhos tais como a teoria das funções complexas e física-matemática, a definição da Integral Definida, a Soma de Riemann e o conceito de espaço amplo e fundamento da geometria que, anos mais tarde, foram usados para desenvolver a teoria da relatividade geral de Albert Einstein (ANTON, 2002, p.410; STEWART, 2006, p.379).

A seguir está a descrição da Soma de Riemann e seus casos mais usuais para a obtenção do cálculo de área.

1.2 Soma de Riemann

A teoria descrita a seguir foi baseada em Anton (2002), Finney, Giordano e Weir (2005), Guidorizzi (2007) e Stewart (2006).

Figura 1.2.1: Gráfico demonstrando os retângulos para o cálculo da área entre as retas verticais, o eixo x e a função.

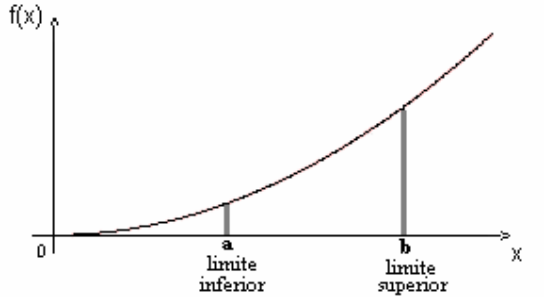


A figura 1.2.1 ilustra a definição da Soma de Riemann, que é a soma das áreas dos retângulos aproximantes entre a função e o eixo x e entre duas retas verticais.

Para calcularmos uma área usando a Soma de Riemann, podemos seguir o seguinte algoritmo:

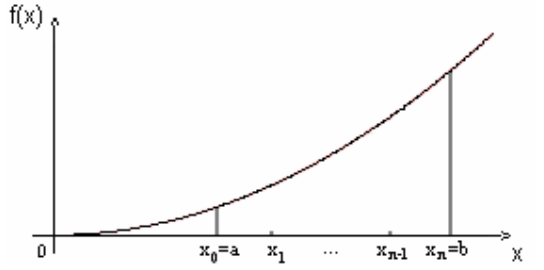
Temos uma função f(x) contínua num intervalo fechado [a, b], onde a e b são os limites inferior e superior respectivamente, como mostra a figura 1.2.2.

Figura 1.2.2: Limite inferior (x = a) e limite superior (x = b).



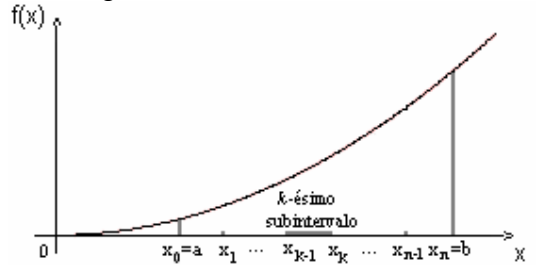
O intervalo [a, b] é particionado em n subintervalos de comprimentos $\Delta x = (b - a) / n$, onde a partição é representada por um conjunto finito $P= \{x_0, x_1, x_2, ..., x_n\}$, seguindo a condição $a = x_0 < x_1 < x_2 < ... < x_n = b$, como na figura 1.2.3.

Figura 1.2.3: Intervalo [a, b] particionado.



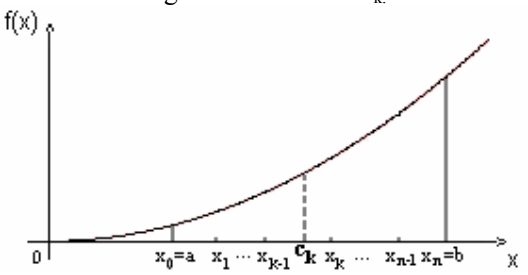
Os n subintervalos são: $[x_0, x_1]$, $[x_1, x_2]$, ..., $[x_{k-1}, x_k]$, ..., $[x_{n-1}, x_n]$, onde $[x_{k-1}, x_k]$ é o *k*-ésimo subintervalo de P, ilustrado na figura 1.2.4. O comprimento deste subintervalo é representado por $\Delta x_k = x_k - x_{k-1}$.

Figura 1.2.4: *k*-ésimo subintervalo.



Escolhemos um ponto amostral em cada subintervalo, denotando por c_k o ponto do *k*-ésimo subintervalo como pode ser verificado na figura 1.2.5.

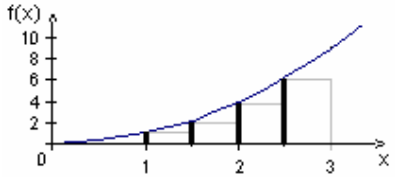
Figura 1.2.5: Ponto c_k.



Construímos retângulos com base no eixo x para cada subintervalo, atingindo a curva em $(c_k, f(c_k))$. O valor de c_k mais usual pode ser o ponto:

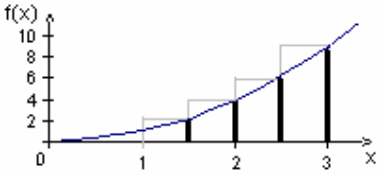
- mais à esquerda, chamado de aproximação pelo extremo esquerdo, onde $c_k = x_{k-1}$ (figura 1.2.6).

Figura 1.2.6: Aproximação pelo extremo esquerdo.



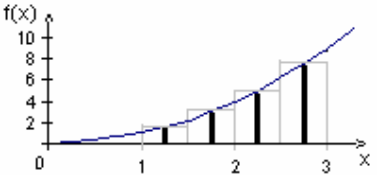
- mais à direita, chamado de aproximação pelo extremo direito, onde $c_k = x_k$ (figura 1.2.7).

Figura 1.2.7: Aproximação pelo extremo direito.



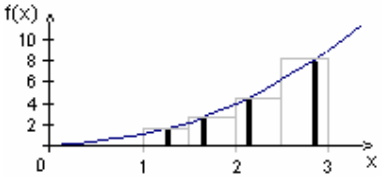
- aproximação pelo ponto médio, onde $c_k = \Delta x_k / 2$ (figura 1.2.8).

Figura 1.2.8: Aproximação pelo ponto médio.



- aproximação pelo ponto aleatório ou ponto randômico, onde c_k está entre x_{k-1} e x_k (figura 1.2.9).

Figura 1.2.9: Aproximação pelo ponto randômico.



Calculamos a área de um retângulo com $f(c_k) . \Delta x_k$ para cada subintervalo;

Se $f(x) \geq 0$, obtemos a área aproximada da região somando o produto de todos os subintervalos. A fórmula para calcular a área total é definida por

$$A = \sum_{k=1}^{n} f(c_k) . \Delta x_k$$

Por exemplo, para calcularmos a área destacada da figura 1.2.10, com $f(x) = x^2$, a = 1, b = 3 e n = 10, onde n é o número de subintervalos e, usando a aproximação pelo ponto médio, temos:

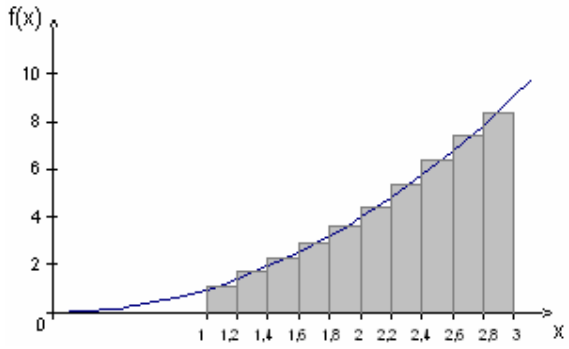
$A = \Delta x_k . [f(1,1) + f(1,3) + f(1,5) + f(1,7) + f(1,9) + f(2,1) + f(2,3) + f(2,5) + f(2,7) + f(2,9)]$

Como $\Delta x_k = (3 – 1) / 10 = 1/5$, então:

$A = 1/5 . (1,21 + 1,69 + 2,25 + 2,89 + 3,61 + 4,41 + 5,29 + 6,25 + 7,29 + 8,41)$

$A = 8,66$

Figura 1.2.10: Área acima do eixo x.



Se $f(x) < 0$, a fórmula é representada em módulo:

$$A = \sum_{k=1}^{n} \left| f(c_k) . \Delta x_k \right|$$

Ao calcularmos a área da figura 1.2.11, com $f(x) = -(x^2)$, a = 1, b = 3 e n = 5, usando a aproximação pelo ponto médio, temos:

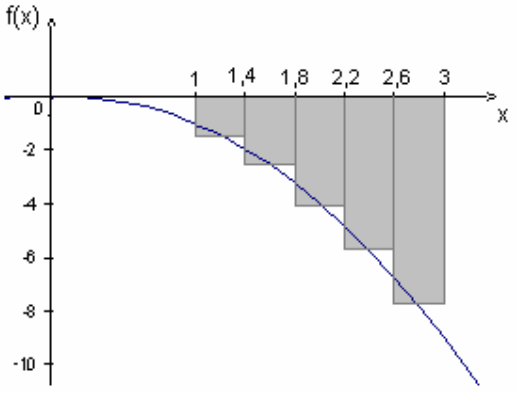
$A = \left| \Delta x_k . [f(1,2) + f(1,6) + f(2) + f(2,4) + f(2,8)] \right|$

Como $\Delta x_k = (3 - 1) / 5 = 2/5$, então:

$A = \left| 2/5 \, . \, [(-1,44) + (-2,56) + (-4) + (-5,76) + (-7,84)] \right|$

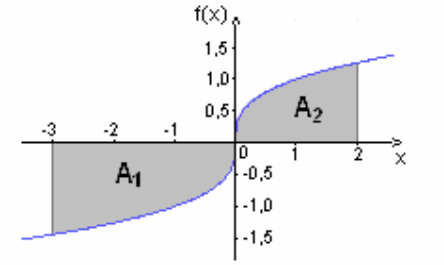
$A = \left| -8,64 \right| = 8,64$

Figura 1.2.11: Área abaixo do eixo x.



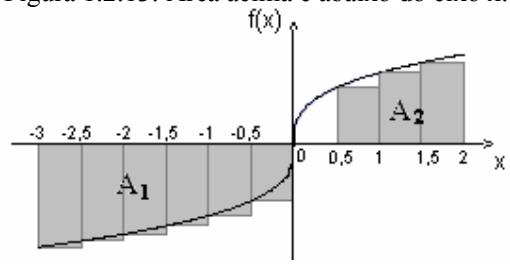
Podemos ter as duas condições apresentadas acima numa mesma situação, como mostra a figura 1.2.12, com a função x^3.

Figura 1.2.12: $f(x) \geq 0$ (A_2) e $f(x) < 0$ (A_1).



Vamos resolver este caso com os valores de a = -3, b = 2, n = 10 e usando a aproximação pelo extremo esquerdo, como exibe a figura 1.2.13.

Figura 1.2.13: Área acima e abaixo do eixo x.



$A_1 = \left| \Delta x_k . [f(-3) + f(-2,5) + f(-2) + f(-1,5) + f(-1) + f(-0,5)] \right|$

$A_1 = \left| \Delta x_k . [(-1,44) + (-1,36) + (-1,26) + (-1,14) + (-1) + (-0,79)] \right|$

$A_1 = \left| \Delta x_k . (-6,99) \right|$

$A_2 = \Delta x_k . [f(0) + f(0,5) + f(1) + f(1,5)]$

$A_2 = \Delta x_k . [0 + 0,79 + 1 + 1,14]$

$A_2 = \Delta x_k . 2,93$

$A_{TOTAL} = A_1 + A_2$

Como $\Delta x_k = [2 – (–3)] / 10 = 1/2$, então:

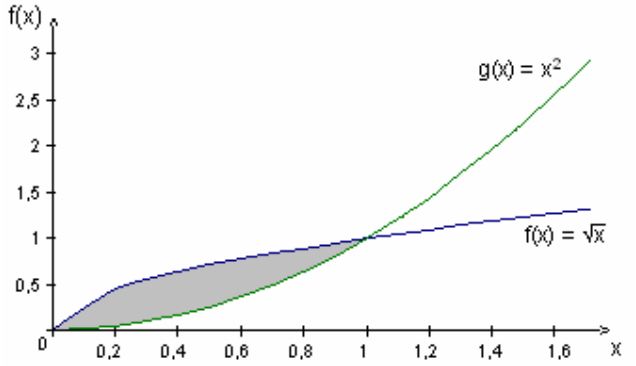
$A_{TOTAL} = \left| 1/2 . (-6,99) \right| + 1/2 . 2,93$

$A_{TOTAL} = 3,495 + 1,465$

$A_{TOTAL} = 4,96$

Outra situação é quando temos duas funções, como exibe a figura 1.2.14. Para calcularmos a área destacada, subtraímos a área sob a função g(x) da área sob a função f(x).

Figura 1.2.14: Área = f(x) – g(x).



Por exemplo, podemos calcular as áreas separadamente. Assumindo os valores de a = 0, b = 1, n = 5 e usando a aproximação pelo ponto médio, temos:

-Área sob f(x) - (figura 1.2.15).

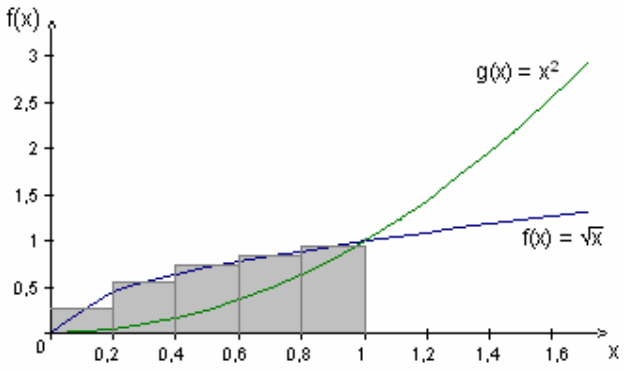
$A = \Delta x_k . [f(0,1) + f(0,3) + f(0,5) + f(0,7) + f(0,9)]$

Como $\Delta x_k = (1 – 0) / 5 = 1/5$, então:

$A = 1/5 . (0,01 + 0,09 + 0,25 + 0,49 + 0,81)$

$A \approx 0,333$

Figura 1.2.15: Área sob f(x).



-Área sob g(x) - (figura 1.2.16).

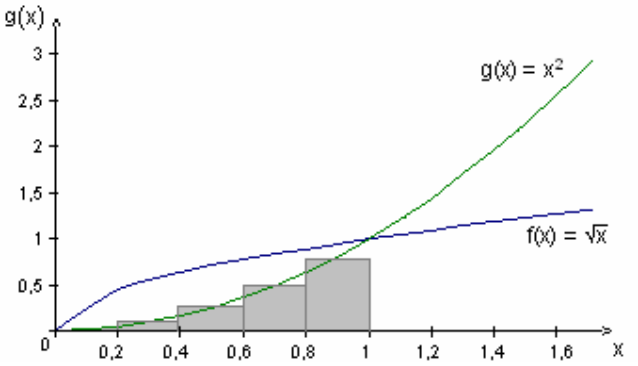
$A = \Delta x_k \, . \, [f(0,1) + f(0,3) + f(0,5) + f(0,7) + f(0,9)]$

Como $\Delta x_k = (1 - 0) / 5 = 1/5$, então:

$A = 1/5 \, . \, (0,316228 + 0,547723 + 0,707107 + 0,83666 + 0,948683)$

$A \approx 0,67128$

Figura 1.2.16: Área sob g(x).



A área destacada da figura 1.2.14 é aproximadamente: $0,67128 - 0,333 \approx 0,338$.

Para obtermos uma maior precisão quanto à área a ser calculada, temos de aumentar o valor de n e, com isto o subintervalo Δx_k diminui para todo k de um até n (FINNEY; GIORDANO; WEIR, 2005, p.348). A tabela 1.2.1 mostra alguns resultados da aproximação pelos extremos esquerdo e direito e pelos pontos médio e randômico da área destacada da figura 1.2.17 sob a curva $f(x) = x^2$, a = 1 e b = 3, com vários valores de n.

Figura 1.2.17: $f(x) = x^2$, a = 1 e b = 3.

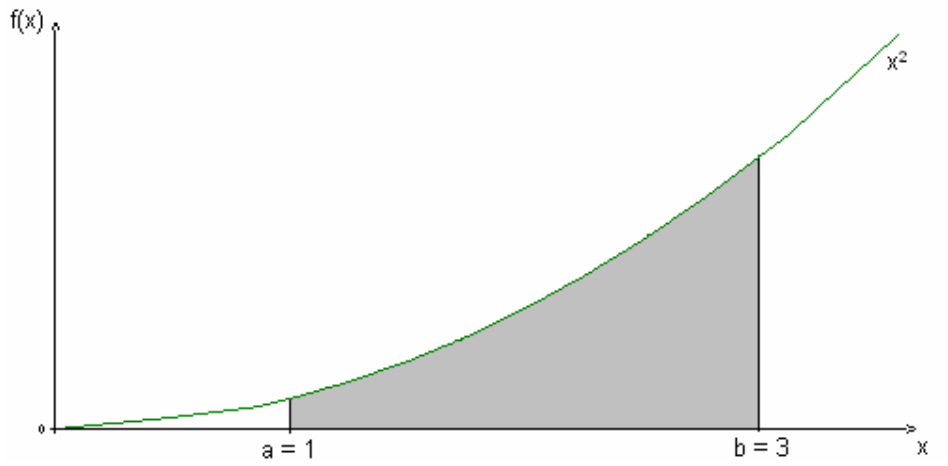


Tabela 1.2.1: Resultados das aproximações de $f(x) = x^2$.

n	Aproximação pelo extremo esquerdo	Aproximação pelo extremo direito	Aproximação pelo ponto médio	Aproximação pelo ponto randômico
1	2	18	8,0	2,54444419714888
2	5	13	8,5	6,17432187632961
3	6,1481481481481469	11,481481481481478	8,5925925925925908	7,97220406859670
4	6,75	10,75	8,625	8,31584418019507
5	7,12	10,32	8,64	8,47090933220337
6	7,3703703703703688	10,037037037037034	8,6481481481481463	9,02351030716335
7	7,5510204081632645	9,8367346938775499	8,6530612244897950	8,47334738003041
8	7,6875	9,6875	8,65625	8,76905133466857
9	7,7942386831275704	9,5720164609053477	8,6584362139917677	8,58166699425583
10	7,88	9,48	8,66	8,53695953599780
20	8,27	9,07	8,665	8,68490310676083
50	8,5072	8,8272	8,6664	8,68604758883257
100	8,5868	8,7468	8,6666	8,67245881814847
200	8,6267	8,7067	8,66665	8,66624873183436
500	8,650672	8,682672	8,666664	8,66637013664396
1000	8,658668	8,674668	8,666666	8,66657632166363
10000	8,66586668	8,66746668	8,66666666	8,66666336763343

Para sabermos se o cálculo da área usando a Soma de Riemann está correto, podemos comparar o resultado desta com o da área calculada pela integral definida. A integral definida

de f de a até b é o limite da soma de todos os retângulos que representa a área total entre f(x) e [a, b] sendo denotada por:

$$A = \lim_{n \to \infty} \sum_{k=1}^{n} f(c_k).\Delta x_k = \int_a^b f(x)\,dx$$

Pela integral definida conseguimos o resultado contínuo de a até b sem quebras (FINNEY; GIORDANO; WEIR, 2005, p.349). Para o cálculo da área sob a curva de $f(x) = x^2$, a = 1 e b = 3 da figura 1.2.17, temos:

$$A = \int_a^b x^2\,dx = \left[\frac{x^3}{3}\right]_1^3 = \frac{27}{3} - \frac{1}{3} = \frac{26}{3} \approx 8{,}66666$$

Portanto, ao compararmos o resultado da integral definida com os da tabela 1.2.1 podemos concluir que, quanto maior o valor de n, o resultado da área se torna mais precisa.

CAPÍTULO 2 – DIAGRAMAS UML

2.1 Introdução

De acordo com Booch, Jacobson e Rumbaugh (2005), Guedes (2006) e Fowler (2005), a UML é uma linguagem padronizada de modelagem de sistemas de softwares que surgiu da junção das metodologias de modelagem de software orientado a objetos de Grady Booch, James Rumbaugh e Ivar Jacobson e padronizada desde 1997 pela OMG (Grupo de Gerenciamento de Objetos) como linguagem de desenvolvimento de software orientado a objetos.

Para desenvolvermos um sistema, podemos seguir um processo delineado de análise de requisitos e modelar um projeto para atender estes requisitos. Na análise determinamos o que o sistema deve fazer, identificando e documentando de forma clara e não-ambígua o que de fato é necessário e, no projeto, como o sistema deve fazer para solucionar os requisitos da análise (DEITEL; DEITEL, 2005, p.16; FOWLER, 2005, p.48). A documentação gerada na análise denomina-se documento de requisitos, podendo ser a conseqüência de entrevistas com possíveis usuários do sistema e especialistas envolvidos com o mesmo (DEITEL; DEITEL, 2005, p.45). Podemos definir requisitos como a descrição das necessidades para um produto (LARMAN, 2002, p.60).

O objetivo da UML é representar graficamente e documentar todas as etapas do desenvolvimento do sistema com paradigmas de orientação a objetos antes do início da implementação, além de apoiar na manutenção do sistema (GUEDES, 2006).

A UML é dividida nos seguintes diagramas: Classes, Objetos, Implantação, Estrutura Composta, Componentes, Pacotes, Casos de Uso, Atividades, Máquina de Estados, Seqüência, Comunicação, Geral de Interação e Tempo (GUEDES, 2006, p.35; BOOCH; JACOBSON; RUMBAUGH, 2005, p.26); que Guedes (2006, p.25) cita que “o objetivo disto é fornecer múltiplas visões do sistema a ser modelado, analisando-o e modelando-o sob diversos aspectos, procurando-se assim atingir a completitude da modelagem” e também que “a utilização de diversos diagramas permite que falhas sejam descobertas, diminuindo a possibilidade da ocorrência de erros futuros”.

Na modelagem do software são usados os diagramas de Casos de Uso, Classes, Seqüência e Componentes descritos a seguir.

2.2 Diagrama de Casos de Uso

O Diagrama de Casos de Uso tem o objetivo de informar o usuário das interações do mesmo com o sistema de software a ser desenvolvido, sem fornecer os detalhes de implementação. Este diagrama geralmente é usado no começo da modelagem do sistema na fase de análise de requisitos (GUEDES, 2006, p.45) e é importante para modelar os comportamentos do sistema (BOOCH; JACOBSON; RUMBAUGH, 2005, p.241). Comportamento do sistema descreve o que um sistema faz, isto é, como este muda quando seus objetos se interagem (LARMAN, 2002, p.143; DEITEL; DEITEL, 2005, p.46).

O Diagrama de Casos de Uso pode ser aplicado à modelagem de negócio descrevendo a estrutura e a dinâmica da empresa (BOOCH; JACOBSON; RUMBAUGH, 2005, p.448).

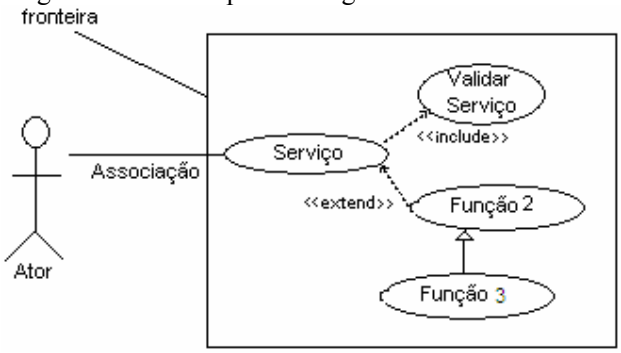
Segundo Booch, Jacobson e Rumbaugh (2005), Guedes (2006) e Larman (2002), neste diagrama o usuário é descrito graficamente como um boneco (ator) que é quem e de alguma forma interage com o sistema sem fazer parte do mesmo. Um ator pode também representar um hardware ou um outro software que interaja com o sistema. Esta interação é denominada de associação e é descrita por meio de uma reta que conecta o ator a uma elipse designada caso de uso. O caso de uso representa serviços ou funções que o sistema deve executar para cada ator. Além da interação entre ator e caso de uso, podemos ter o relacionamento entre atores e entre casos de uso. O relacionamento entre casos de uso pode ser denominado especialização/generalização, inclusão (<<include>>) e extensão (<<extend>>). A especialização/generalização ocorre com dois ou mais casos de uso e dar-se-á quando há características em comum com pequenas diferenças entre si. A especialização/generalização é representada por uma reta com uma seta fechada em sua extremidade apontada para o caso de uso geral. A representação da extensão e da inclusão é feita por meio de retas tracejadas com uma seta aberta na extremidade. A extensão e inclusão são estereótipos da UML. Estereótipo destaca a função de um determinado item de um diagrama (GUEDES, 2006). Para separarmos o que está externo do que está interno no sistema, envolvemos este num retângulo. Este retângulo é denominado fronteira do sistema (GUEDES, 2006, p.55).

O Diagrama de Casos de Uso pode vir acompanhado de sua especificação (documentação). Esta especificação descreve o(s) ator(es) envolvido(s), o caso de uso, um resumo deste, as etapas realizadas pelo ator e pelo sistema para que o caso de uso execute sua função, quais restrições e validações devem possuir. Contudo, a UML é flexível quanto a

estes itens, podendo ser alterados para satisfazer as necessidades de documentação (LARMAN, 2002, p.68; GUEDES, 2006, p.47).

Como mostra a figura 2.2.1, o ator solicita um serviço ao sistema. O relacionamento inclusão obriga o caso de uso Serviço executar o caso de uso Validar Serviço. A extensão não obriga a execução do caso de uso, por exemplo, o caso de uso Função2 pode ou não ser executado. Há também a especialização/generalização entre a Função2 e a Função3. Neste caso, a Função3 possui todas as características da Função2 além das suas e, dizemos que a Função2 é uma generalização da Função3 e esta é uma especialização daquela (DEITEL; DEITEL, 2005).

Figura 2.2.1: Exemplo de Diagrama de Casos de Uso.



O Diagrama de Casos de Uso pode ser usado e alterado pelo desenvolvedor do sistema durante todo o processo de desenvolvimento do sistema (GUEDES, 2006, p.45).

2.3 Diagrama de Classe

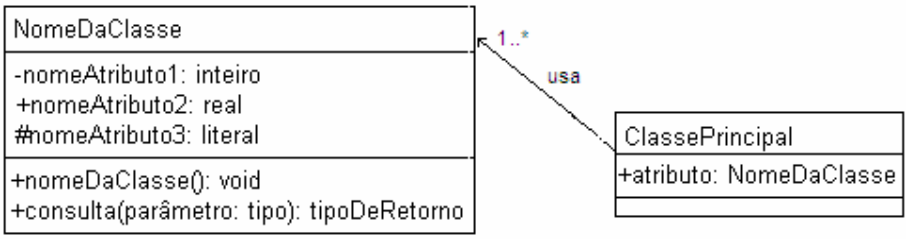
Segundo Deitel e Deitel (2005), Guedes (2006) e Fowler (2005), o Diagrama de Classes tem por objetivo definir a navegabilidade (relação) entre as classes, mostrar os nomes dos atributos com seus tipos e os métodos. Classe é um conjunto de objetos com os mesmos atributos (características) e métodos. O objeto é um item da classe e difere-se de outros objetos pelos conteúdos de seus atributos. Os métodos são serviços ou operações que os objetos podem executar. Quando é necessário preservar os objetos de uma classe permanentemente, esta classe é chamada de classe persistente (GUEDES, 2006, p.69).

Podemos gerar um Diagrama de Classes analisando os substantivos do documento de requisitos e identificando as classes necessárias para compor este diagrama. Listamos os substantivos e separamos dentre eles, os atributos das classes, aqueles que não fazem parte do

sistema e aqueles que compreendem o sistema, provavelmente sendo estes as classes necessárias para implementar o sistema (DEITEL; DEITEL, 205, p.76).

Este diagrama estrutura todo o sistema a ser desenvolvido (GUEDES, 2006, p.27). A figura 2.3.1 mostra um exemplo de Diagrama de Classes.

Figura 2.3.1: Exemplo de Diagrama de Classes.



Como mostrado nesta figura, cada classe é formada por até três divisões (GUEDES, 2006, p.38). A primeira divisão contém o nome da classe. Na segunda divisão são definidos os atributos com os nomes e os tipos de dados. A última divisão mostra os métodos da classe, com seus possíveis parâmetros e o tipo de retorno.

Como pode ser observado na figura 2.3.1, os símbolos antes dos nomes dos atributos e dos métodos definem a visibilidade (nível de acesso) de outras classes quanto a estes atributos e métodos (GUEDES, 2006, p.40-41). Os níveis de acesso podem ser:

privado: representado geralmente por – (menos), e determina que apenas a classe proprietária do atributo ou método poderá usá-lo;

protegido: representado geralmente por # (sustenido), e determina a utilização de um atributo ou método de uma classe por ela ou por suas subclasses;

público: representado geralmente por + (mais), e garante que todas as classes tenham acesso ao atributo ou método da classe.

As classes se relacionam entre si por meio de retas, que significam compartilhamento de informações para a execução de um processo (GUEDES, 2006, p.72). Estes relacionamentos podem ser:

associações: especifica a ligação que ocorre entre as classes. Esta ligação pode ser entre duas classes (associação binária), em mais de duas classes (associação ternária ou N-ária) ou vinculada à própria classe (associação unária ou reflexiva). Estas associações podem possuir nomes para mostrar o tipo de vínculo entre as classes. Podem também possuir multiplicidade, que determina o nível de dependência de uma classe com as outras envolvidas e qual destas classes fornece informações para as outras (GUEDES, 2006; FOWLER, 2005). Na figura 2.3.1 temos a multiplicidade entre as classes ClassePrincipal e NomeDaClasse. Neste

exemplo podemos dizer que há pelo menos um objeto NomeDaClasse envolvido no relacionamento com a ClassePrincipal (GUEDES, 2006). A tabela 2.3.1 mostra alguns exemplos de multiplicidade (GUEDES, 2006):

Tabela 2.3.1: Exemplos de Multiplicidade.

Multiplicidade	**Significado**
0..1	Os objetos das classes associadas podem ter no mínimo zero e no máximo um objeto envolvido no relacionamento.
1..1	Apenas um objeto de uma classe se relaciona com os objetos de outra classe.
0..*	No mínimo zero e no máximo muitos objetos participando do relacionamento.
*	Especifica muitos objetos envolvidos no relacionamento.
1..*	No mínimo um e no máximo muitos objetos fazendo parte do relacionamento.

Há também a associação de agregação e de composição. A agregação indica uma relação todo/parte entre os objetos relacionados, isto é, as informações de um objeto (objeto-todo) precisam ser complementadas pelas informações contidas em um ou mais objetos de outra classe (objeto-parte). As duas partes relacionadas podem existir independentemente. (GUEDES, 2006; DEITEL; DEITEL, 2005). A agregação é representada por uma reta com um losango na extremidade da classe do objeto todo. A composição indica um relacionamento todo/parte mais forte que a agregação, especificando que o objeto-parte pertence exclusivamente a um único objeto-todo. É representado por um losango preenchido na extremidade da classe do objeto todo (GUEDES, 2006);

especialização/generalização: é muito similar ao relacionamento de mesmo nome no Diagrama Caso de Uso. O objetivo é identificar classes-mãe (gerais) e classes-filhas (especializadas). Tem a mesma representação do Diagrama Caso de Uso (GUEDES, 2006);

dependência: indica uma dependência de uma classe pela outra, isto é, a classe dependente sofre alterações quando é alterada a classe de que ela depende (GUEDES, 2006; FOWLER, 2004).

O Diagrama de Classes possui três estereótipos predefinidos (GUEDES, 2006):

<<entity>>: informa que a classe possui informações transmitidas ou geradas por meio do sistema. Estas informações podem ter um período de vida longo ou apenas serem armazenadas na memória;

<<boundary>>: especifica a classe que faz a comunicação entre os atores externos e o sistema. Em geral é definido como uma interface para o sistema;

<<control>>: faz a ligação entre a classe de estereótipo <<boundary>> e as outras classes do sistema. Este tipo interpreta os eventos dos objetos <<boundary>> e os retransmitem para os objetos das classes do sistema.

Além destes três estereótipos, a UML permite a criação de outros estereótipos (GUEDES, 2006).

2.4 Diagrama de Seqüência

Segundo Guedes (2006), o Diagrama de Seqüência demonstra a comunicação entre os objetos, as mensagens trocadas por estes, os métodos chamados e a ordem temporal em que os eventos ocorrem. Um evento é um serviço solicitado ao sistema por um ator ou por outro sistema.

Os Diagramas de Seqüência, de Comunicação e de Tempo são diagramas de Interação. Este diagrama descreve uma interação entre um conjunto de objetos contendo mensagens que podem ser trocadas entre eles. A diferença entre aqueles diagramas é que o de Seqüência enfatiza a ordenação temporal das mensagens, o de Comunicação destaca a organização estrutural dos objetos que enviam e recebem mensagens e o de Tempo salienta os tempos reais para as trocas de mensagens (BOOCH; JACOBSON; RUMBAUGH, 2005, p.27).

Para gerarmos um Diagrama de Seqüência, analisamos um caso de uso específico e determinamos seus processos, além de dependermos dos métodos e objetos das classes do Diagrama de Classe (GUEDES, 2006, p.104).

De acordo com Booch, Jacobson e Rumbaugh (2005) e Guedes (2006), o Diagrama de Seqüência tem os seguintes componentes:

linha de vida: tempo em que o objeto fez parte do processo. É representado por uma reta vertical tracejada que parte do objeto;

ator: o mesmo representado no diagrama caso de uso, porém contendo uma linha de vida;

objeto: instância da classe envolvida no processo. É representado por um retângulo contendo a classe do objeto e podendo conter o nome deste objeto, sendo separados por dois pontos e sublinhados;

foco de controle ou ativação: indica o período em que o objeto está executando uma ação. É representado na linha de vida de um objeto por um retângulo alto e estreito;

mensagens ou estímulos: especifica a ocorrência de eventos que podem forçar a chamada de um método. É representada por uma reta saindo de um ator/objeto e apontando com uma seta para o ator/objeto que recebe a ativação;

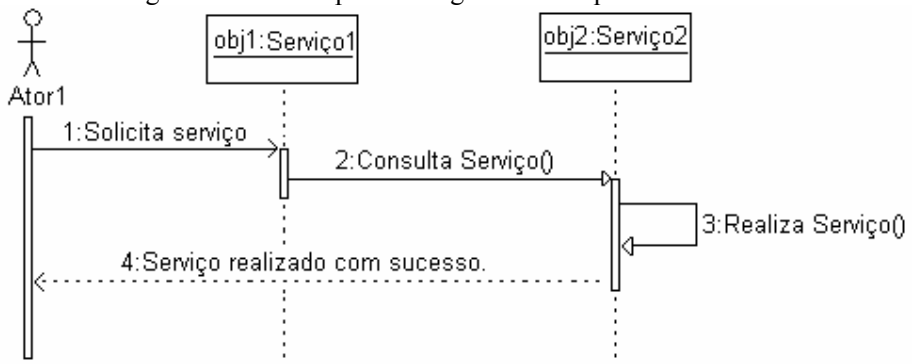
mensagem de retorno: mensagem de resposta de um método para o objeto ou ator que chamou este método. É representada por uma linha tracejada contendo uma seta apontada para o objeto ou ator que recebe a mensagem;

auto-chamadas ou auto-delegações: mensagem disparada e recebida por um mesmo objeto;

condições ou condições de guarda: expressão booleana colocada entre colchetes antes do evento de ativação para indicar que uma mensagem só poderá ser enviada se a expressão for satisfeita. Também é conhecida como condição de proteção.

A figura 2.4.1 exemplifica um Diagrama de Seqüência.

Figura 2.4.1: Exemplo de Diagrama de Seqüência.



Na figura acima, temos um ator designado por Ator1, um objeto chamado obj1 da classe Serviço1 e outro chamado obj2 da classe Serviço2. O Ator1 solicita um serviço, representado por meio de uma reta com uma seta aberta, forçando que o obj1 dispare o método Consulta Serviço() para o obj2, representado por meio de uma reta com uma seta fechada. O obj2 dispara o método Realiza Serviço() em si próprio e em seguida retorna uma mensagem ao Ator1, representado por meio de uma reta tracejada com uma seta aberta.

2.5 Diagrama de Componentes

Um componente pode ser definido como cada arquivo que faz parte do sistema (GUEDES, 2006, p.168). O componente é a parte física do sistema e pode ser substituída ou alterada (BOOCH; JACOBSON; RUMBAUGH, 2005, p.342).

De acordo com Booch, Jacobson e Rumbaugh (2005) e Guedes (2006), um componente pode conter os seguintes estereótipos:

<<executable>>: refere-se a um arquivo executável;

<<library>>: especifica uma biblioteca de funções e sub-rotinas que podem ser compartilhadas pelos componentes;

<<table>>: representa uma tabela de banco de dados;

<<document>>: refere-se a arquivos-texto;

<<file>>: refere-se aos arquivos de código-fonte.

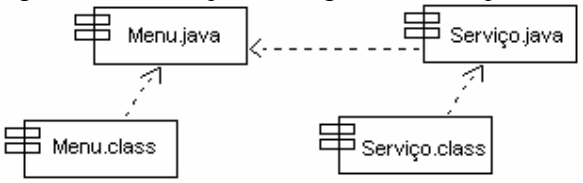
Um componente é representado por dois retângulos pequenos sobressaindo à esquerda de outro retângulo maior (GUEDES, 2006, p.169).

O Diagrama de Componentes modela o aspecto físico do sistema mostrando como os componentes são organizados e a relação entre os mesmos (BOOCH; JACOBSON; RUMBAUGH, 2005, p.387; GUEDES, 2006, p.168).

Um componente pode conectar-se a outro por meio de um relacionamento de dependência, significando que o componente utiliza serviços de outro. O relacionamento de dependência é representado por uma reta tracejada contendo uma seta que aponta de que outro componente o componente em questão depende (GUEDES, 2006).

A figura 2.5.1 exemplifica um Diagrama de Componentes.

Figura 2.5.1: Exemplo de Diagrama de Componentes.

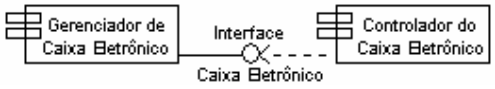


Na figura acima, temos os componentes Menu.class e Serviço.class que dependem respectivamente dos componentes Menu.java e Serviço.java. O componente Serviço.java também tem um relacionamento de dependência com o componente Menu.java.

Outro conceito importante do Diagrama de Componentes é a Interface, que é definida como um serviço realizado por uma classe ou componente. Usualmente um

componente implementa (realiza) ou depende de uma interface (GUEDES, 2006). A Interface não tem implementação e é representada por um círculo como mostra a figura 2.5.2.

Figura 2.5.2: Exemplo de Interface.



Neste exemplo o componente Gerenciador de Caixa Eletrônico implementa a interface Caixa Eletrônico por meio do relacionamento de realização, fazendo com que o componente de interface herde o comportamento daquele. O componente Controlador do Caixa Eletrônico está associado à interface pelo relacionamento de dependência.

CAPÍTULO 3 – DESENVOLVIMENTO DO SOFTWARE PARA CALCULAR ÁREAS POR MEIO DA SOMA DE RIEMANN

3.1 Introdução

Este capítulo demonstra o estudo de caso da modelagem do software com a UML, as funcionalidades deste software, alguns recursos da API JFreeChart e a última parte descreve sobre o Maple e as comparações dos resultados obtidos.

3.2 Modelagem do software com a UML

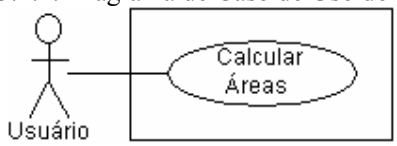
Para iniciarmos a modelagem do sistema, podemos analisar o seguinte Documento de Requisitos:

Desenvolver um sistema para efetuar o cálculo de áreas usando a Soma de Riemann. Vamos considerar os seguintes requisitos:

- o cálculo deve ser feito para os casos da aproximação pelos pontos à esquerda, à direita, médio e randômico;
- gerar os gráficos para os casos apontados anteriormente;
- a interface do sistema deve oferecer a opção para o usuário escolher a função;
- o usuário deve informar os seguintes parâmetros de entrada: limites inferior e superior e quantidade de subintervalos;
- o usuário sempre poderá escolher outra função ou entrar com novos parâmetros;
- o sistema deve exibir uma mensagem apropriada quando falhar ou quando algum parâmetro de entrada estiver incorreto;
- para gerar os gráficos é usada uma ou mais funções;
- para realizar a Soma de Riemann é usada uma ou mais funções;
- as funções são predeterminadas;
- o sistema deve mostrar para o usuário os gráficos e os resultados dos quatro casos da Soma de Riemann.

Ao analisarmos a descrição do problema acima, podemos determinar que o usuário é quem faz a interação com o sistema enquanto a função deste é calcular áreas usando a Soma de Riemann. Portanto, podemos definir o usuário como o Ator e calcular áreas como o Caso de Uso. O Diagrama de Caso de Uso é exibido na figura 3.2.1.

Figura 3.2.1: Diagrama de Caso de Uso do sistema.



Nesta figura temos o Ator Usuário conectado ao Caso de Uso Calcular Áreas que está no interior da fronteira do sistema. A tabela 3.2.1 descreve a documentação do Caso de Uso Calcular Áreas.

Tabela 3.2.1: Documentação do Caso de Uso Calcular Áreas.

Nome do caso de uso	Calcular Áreas
Ator	Usuário
Resumo	Este caso de uso descreve as etapas percorridas pelo Usuário para calcular áreas usando a Soma de Riemann.
Ações do Ator (Fluxo Normal)	**Respostas do Sistema (Fluxo Normal)**
1. Selecionar uma função.	
2. Entrar com os limites inferior e superior e a quantidade de subintervalos.	
3. Solicitar o cálculo de áreas.	
	4. Validar os dados de entrada.
	5. Exibir os resultados das áreas e os gráficos.
Restrições/Validações	
	1. Não é necessário selecionar uma função. A primeira função da lista é selecionada por default (padrão).
	2. O limite inferior não deve ser maior que o limite superior.
	3. A quantidade de subintervalos deve ser maior que zero.
	4. Os limites têm de satisfazer o intervalo da função. Ex: na função $\sqrt{x}$ nenhum limite pode ser negativo.

Nosso próximo passo é identificar as possíveis classes do sistema. Vamos separar os substantivos do Documento de Requisitos e analisá-los conforme a seguir:

cálculo de áreas: serviço que um objeto pode executar. Podemos defini-lo como um método;

Soma de Riemann: podemos abstrair que este substantivo tem o mesmo sentido que o anterior, ou seja, é o método pelo qual usaremos para realizar o cálculo de áreas;

Riemann: podemos determinar esta como uma classe do sistema e que nela sejam feitas as operações para calcular áreas, além de armazenar informações para a geração de gráficos. Riemann usa funções para realizar seus cálculos;

pontos à esquerda, à direita, médio e randômico: vamos calcular estes usando a Soma de Riemann. Podemos determiná-los como atributos de Riemann;

gráfico: podemos defini-lo como classe e, deste modo, separamos a geração de gráficos de outras classes. Para gerar os gráficos são usadas funções e resultados de cálculos efetuados com a Soma de Riemann;

interface do sistema: meio de comunicação entre o Usuário e o sistema. Podemos determiná-lo como uma classe do tipo <<boundary>>;

usuário: é quem interage com o sistema e, por não fazer parte deste, não precisamos modelá-lo como classe;

função: armazena a lista de funções predeterminadas e é usada pela Soma de Riemann e pelo gráfico. Podemos defini-la como classe para não precisarmos implementá-la em Soma de Riemann e em gráfico;

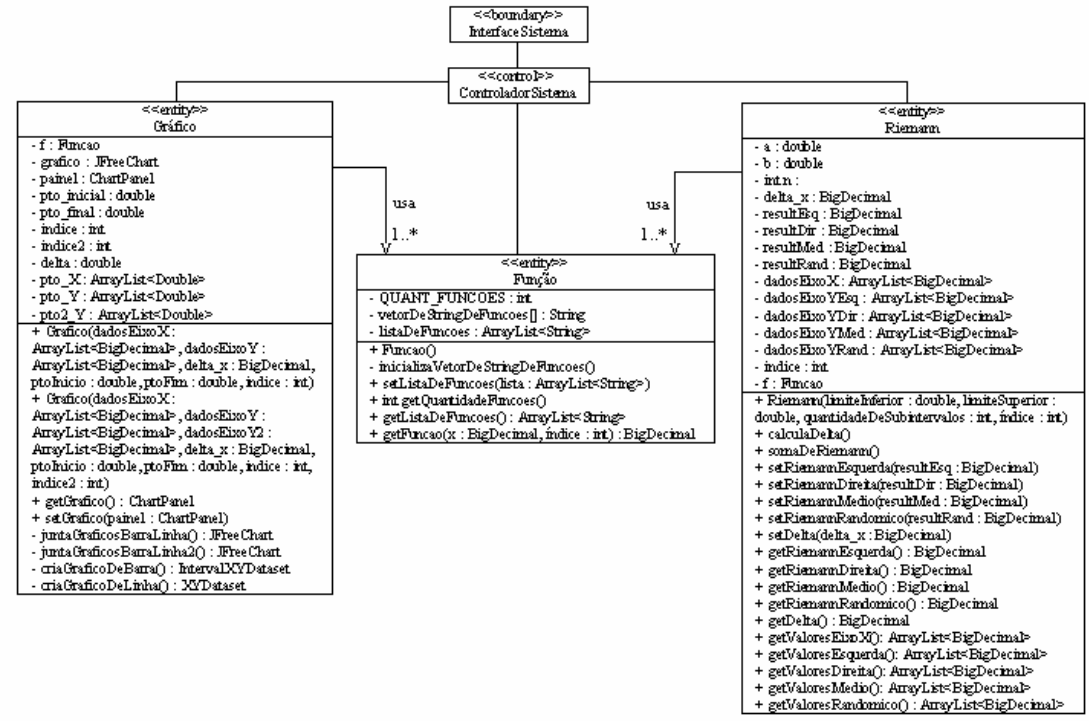
parâmetros de entrada: valores fornecidos pelo Usuário para calcular a Soma de Riemann. Também são usados para formar os gráficos. Podemos defini-los como atributos de Riemann e de gráficos;

mensagem: podemos tratá-la com a classe do tipo <<control>>. Não é necessário defini-la como atributo.

Definimos Função, Gráfico e Riemann como as classes do nosso sistema. Além destas, vamos criar as classes InterfaceSistema e ControladorSistema com os estereótipos do tipo <<boundary>> e <<control>> respectivamente.

Uma possível solução para o nosso Diagrama de Classes é descrita na figura 3.2.2. Além dos atributos e métodos especificados acima, colocamos outros que podem ser necessários para a construção do Diagrama de Classes e, conseqüentemente para a implementação do sistema.

Figura 3.2.2: Diagrama de Classes do sistema.



Neste diagrama temos:

InterfaceSistema: esta classe representa a comunicação entre o Usuário e o sistema. Por meio da InterfaceSistema o Usuário solicita o cálculo de áreas e recebe os resultados e gráficos.

ControladorSistema: classe de controle responsável por traduzir a solicitação feita pelo Usuário e retransmiti-la às outras classes do sistema. Ela também é encarregada de informar o Usuário de possíveis erros de entrada ou falhas no sistema.

Riemann: nesta classe são efetuados os cálculos de áreas usando a Soma de Riemann. Ela também é responsável pelo armazenamento dos resultados dos cálculos para formar o gráfico de barra. Esta classe tem uma associação binária com a classe Função, e esta possui a multiplicidade 1..* em sua extremidade. Podemos interpretar que Riemann usa no mínimo uma e no máximo muitas funções, entretanto, uma função só pode ser usada por um objeto Riemann.

Seus atributos são: a (armazena o limite inferior); b (armazena o limite superior); n (armazena a quantidade de subintervalos); delta_x (armazena o valor de delta); resultEsq, resultDir, resultMed e resultRand (armazenam os resultados da Soma de Riemann); dadosEixoX, dadosEixoYEsq, dadosEixoYDir, dadosEixoYMed e dadosEixoYRand

(armazenam os resultados da Soma de Riemann de toda iteração); índice (função corrente); f (objeto da classe Função).

Seus métodos são: Riemann (método construtor que permite gerar uma nova instância desta classe. Recebe por parâmetro os limites inferior e superior, a quantidade de subintervalos e a função corrente); calculaDelta (calcula o valor de delta [(b-a)/n]); somaDeRiemann (calcula a Soma de Riemann para os quatro casos); setRiemannEsquerda, setRiemannDireita, setRiemannMedio, setRiemannRandomico e setDelta (substituem os resultados anteriores pelos passados por parâmetro); getRiemannEsquerda, getRiemannDireita, getRiemannMedio e getRiemannRandomico (retornam os resultados da Soma de Riemann); getDelta (retorna o resultado de delta_x); getValoresEixoX, getValoresEsquerda, getValoresDireita, getValoresMedio e getValoresRandomico (retornam listas com os pontos para os eixos).

Gráfico: esta classe é responsável pela formação dos gráficos. Ela tem uma associação binária com a classe Função, sendo que esta tem a multiplicidade 1..* em sua extremidade. Podemos entender que um gráfico usa no mínimo uma e no máximo muitas funções, no entanto, uma função só pode ser usada por um gráfico.

Seus atributos são: f (objeto da classe Função); gráfico (armazena o gráfico completo); painel (área onde é gerado o gráfico); pto_inicial e pto_final (pontos inicial e final para gerar o gráfico de linha); índice e indice2 (funções correntes); delta (estabelece a largura de cada barra); pto_X, pto_Y e pto2_Y (pontos para gerar o gráfico de barras).

Seus métodos são: Grafico (método construtor que permite gerar uma nova instância desta classe. Recebem por parâmetro a lista de pontos e o delta_x para gerar o gráfico de barras, os pontos inicial e final para gerar o gráfico de linha e a função. O segundo construtor difere-se do primeiro por receber duas listas de pontos e duas funções); getGrafico (retorna o gráfico); setGrafico (substitui o gráfico anterior pelo passado por parâmetro); juntaGraficosBarraLinha (junta um gráfico de barra com um de linha); juntaGraficosBarraLinha2 (junta dois gráficos de barra com dois de linha); criaGraficoDeBarra (cria o gráfico de barra); criaGraficoDeLinha (cria o gráfico de linha).

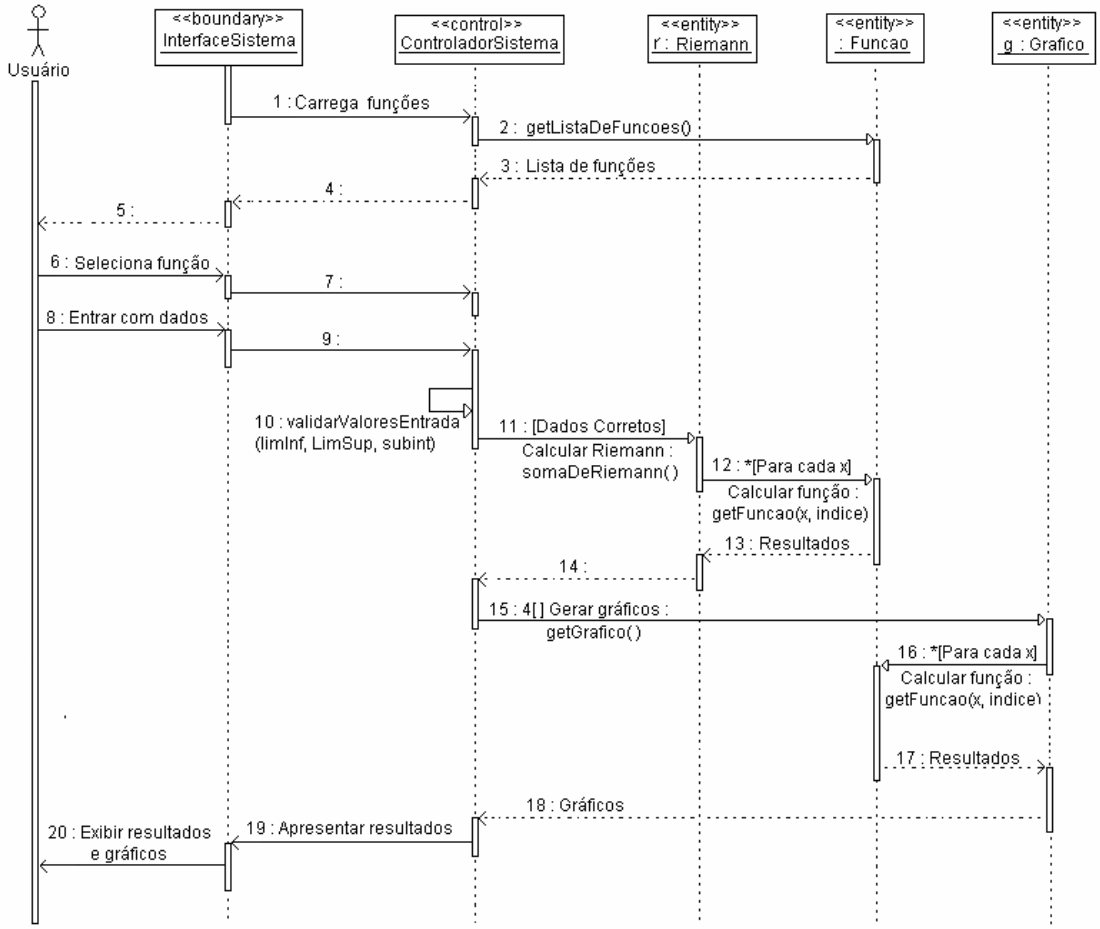
Função: esta classe armazena as funções do sistema e realiza cálculos da função corrente. Além disto, ela é usada pelas Classes Gráfico e Riemann. Esta para a realização do cálculo de áreas e aquela para a geração dos gráficos.

Seus atributos são: QUANT_FUNCOES (armazena a quantidade de funções); vetorDeStringDeFuncoes[] e listaDeFuncoes (armazenam todas as funções).

Seus métodos são: Função (método construtor que permite gerar uma nova instância desta classe); inicializaVetorDeStringDeFuncoes (inicializa a lista com todas as funções); setListaDeFuncoes (substitui a lista anterior de funções pela lista passada por parâmetro); getQuantidadeFuncoes (retorna a quantidade de funções); getListaDeFuncoes (retorna a lista de funções); getFuncao (retorna o cálculo efetuado para um valor x de entrada de uma determinada função da lista).

O nosso próximo passo é modelar o Diagrama de Seqüência. A figura 3.2.3 expõe este diagrama para o fluxo normal explicitado na documentação do Caso de Uso Calcular Áreas.

Figura 3.2.3: Diagrama de Seqüência do sistema (Fluxo Normal).



Este diagrama apresenta os objetos InterfaceSistema e ControladorSistema com os tipos <<boundary>> e <<control>> respectivamente. O primeiro representa a interface do sistema por meio da qual o Usuário entra com os dados e recebe os resultados e os gráficos. O

segundo tem como função interpretar os dados da entrada feita pelo Usuário e repassá-la aos objetos internos do sistema.

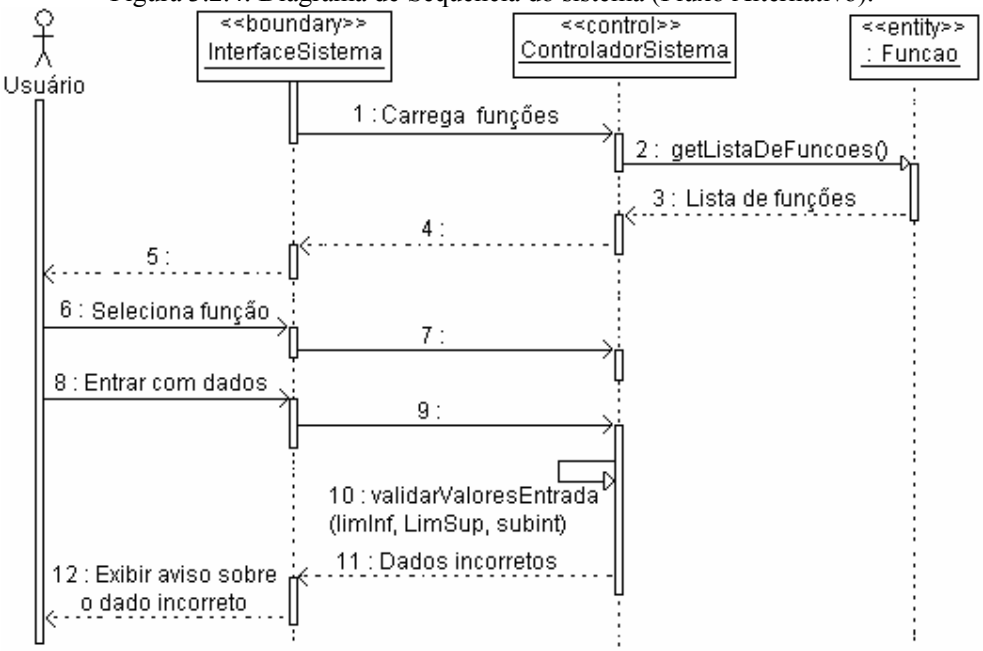
O processo modelado começa quando o usuário inicializa o sistema. Esta ação faz o sistema responder disparando o método getListaDeFuncoes() num objeto da Classe Função e retornar a lista de funções ao Usuário. O Usuário pode escolher uma função, porém se assim não fizer, a primeira função da lista é selecionada por default.

O Usuário entra com os dados (limites e subintervalos) e a solicitação para fazer os cálculos. Este ato faz o sistema disparar o método validarValoresEntrada(limInf, limSup, subint) no ControladorSistema. Se os dados estiverem corretos, é disparado o método somaDeRiemann() no objeto r da Classe Riemann. Para executar este último método é disparado o método getFuncao(x, índice) na Classe Função para cada subintervalo de uma determinada função.

Após o retorno dos resultados calculados acima para o objeto da Classe Riemann, o sistema dispara quatro vezes o método getGrafico() no objeto g da Classe Gráfico, fazendo este disparar o método getFuncao(x, índice) na Classe Função para cada ponto definido de uma determinada função e, assim gerar os gráficos e retornar ao ControladorSistema que, juntamente com os resultados da Soma de Riemann, retorna à InterfaceSistema e este exibe ao Usuário.

A figura 3.2.4 mostra um fluxo alternativo de um processo.

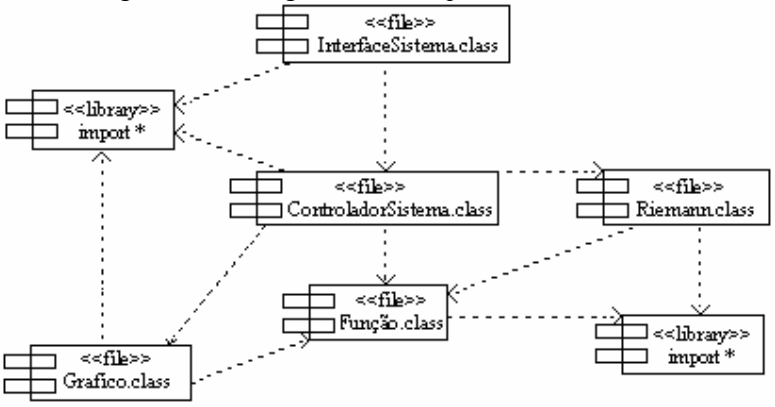
Figura 3.2.4: Diagrama de Seqüência do sistema (Fluxo Alternativo).



Este fluxo alternativo pode acontecer se algum dado de entrada informado pelo Usuário estiver incorreto ou for omitido. O sistema avisa o Usuário sobre o erro e este pode corrigi-lo e executar novamente o processo.

Por último, modelamos o Diagrama de Componentes que é apresentado na figura 3.2.5.

Figura 3.2.5: Diagrama de Componentes do sistema.



Neste diagrama repetimos o componente import * para deixá-lo mais inteligível e não precisarmos interceptar linhas. Este componente tem o estereótipo <<library>> e são as bibliotecas usadas por todos os outros componentes. Estes outros possuem o estereótipo <<file>> e a extensão .class que contém a versão compilada dos componentes .java (código-fonte) (DEITEL; DEITEL, 2005, p.10). Os componentes Riemann.class, Gráfico.class e ControladorSistema.class dependem do componente Função.class para calcular áreas, gerar os gráficos e usar a lista de funções respectivamente. O ControladorSistema.class depende ainda dos componentes Riemann.class e Gráfico.class para receber os resultados destes e retransmiti-los ao componente InterfaceSistema.class.

3.3 Funcionalidades do software

A figura 3.3.1 mostra a interface do software que tem as seguintes funcionalidades:

1 – opção para escolher a função. Caso não seja escolhida, a função x é selecionada por default.

2 – campo de entrada para o limite inferior. Este campo é obrigatório e seu valor não pode ser maior que o valor do limite superior.

3 – campo de entrada para o limite superior. Este campo é obrigatório e seu valor não pode ser menor que o valor do limite inferior.

4 – campo de entrada para a quantidade de subintervalos. Este campo é obrigatório e seu valor tem de ser maior que zero.

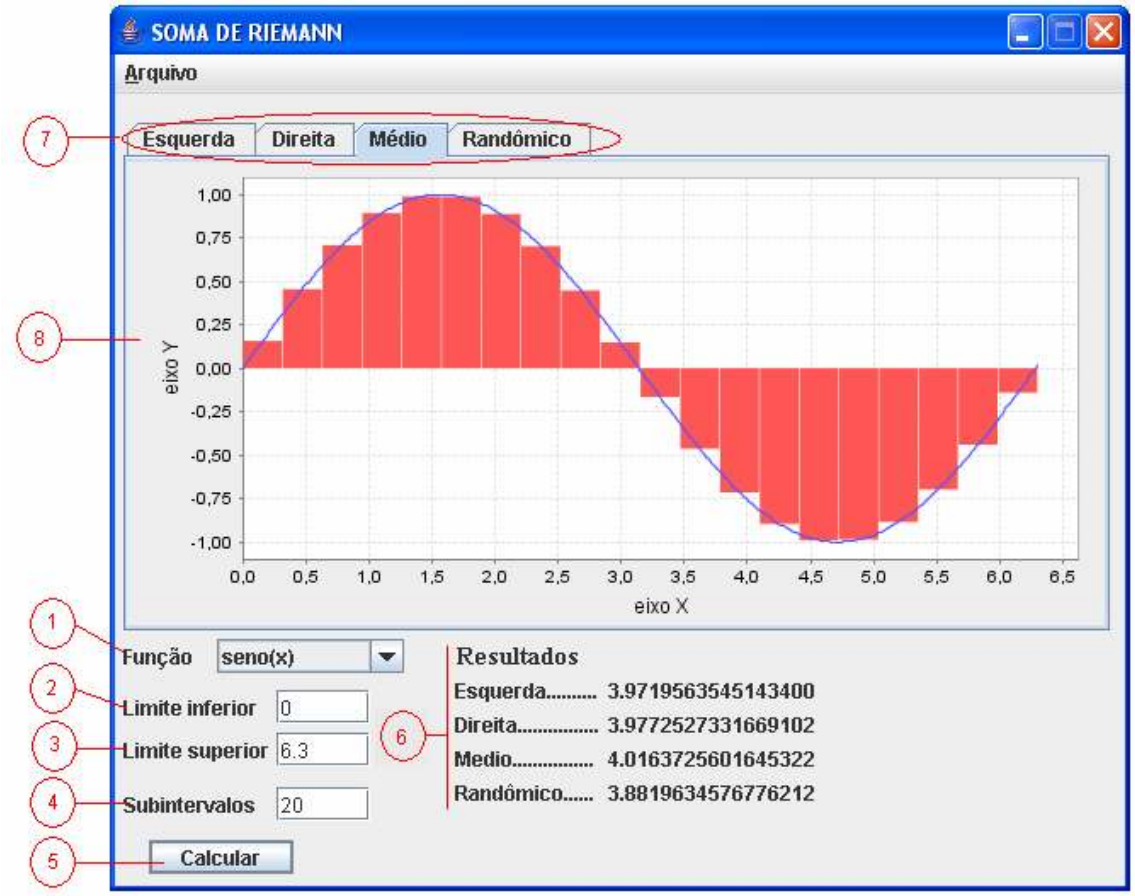
5 – o botão calcular ativa as operações e gera a saída dos resultados e dos gráficos.

6 – área onde são colocados os resultados.

7 – abas de acesso que definem o gráfico visível.

8 – área onde os gráficos são gerados.

Figura 3.3.1: Funcionalidades do Software.



3.4 JFreeChart

JFreeChart é uma biblioteca desenvolvida em Java para a geração de diversos tipos de gráficos, tais como gráficos de pizza, barra, linha, combinados, entre outros. Esta biblioteca possibilita que os gráficos produzidos sejam utilizados em aplicações desktop, applets, servlets, JSP, além de serem exportados para arquivos de imagens, gráficos vetoriais e de formatos de arquivo (OBJECT REFINERY LIMITED, 2008).

JFreeChart foi usada no software para a geração de gráficos combinados (barra e linha). O software gera um gráfico para cada caso da Soma de Riemann e são apresentados em camadas onde apenas uma fica visível por vez, sendo estas acessadas por meio de abas.

3.5 Comparações de resultados com o Maple

Para verificarmos a integridade dos resultados gerados pelo software, usaremos o Maple que, segundo Mariani (2005), é um sistema de computação algébrica e simbólica que tem o propósito de solucionar uma gama de problemas matemáticos, sendo usado em disciplinas como Cálculo, Equações Diferenciais, Álgebra Linear e Geometria Analítica.

O Maple possui diversas funções, dentre as quais, para a resolução da Soma de Riemann nos casos das aproximações dos pontos à esquerda, à direita e médio e para resolução da Integral. Estas funções, que são mostradas na tabela 3.5.1 (MARIANI, 2005; ANDRADE, 2004), serão usadas para fazer os testes de comparações de resultados com o software desenvolvido.

Tabela 3.5.1: Funções do Maple.

	função	**entradas**
Esquerda	leftsum(f(x), x= a..b, n)	**f(x):** função **a:** limite inferior **b:** limite superior **n:** quantidade de subintervalos
Direita	rightsum(f(x), x= a..b, n)	
Médio	middlesum(f(x), x= a..b, n)	
Integral	int(f(x), x= a..b)	

Vamos analisar alguns exemplos para calcularmos áreas usando o Maple:

Exemplo1:

```
f := x -> x;     # função de entrada
                 x -> x
evalf[17](leftsum(f(x), x = 0..3, 10));   # aproximação a esquerda
                 4.0500000000000000   # resultado
evalf[17](rightsum(f(x), x = 0..3, 10));   # aproximação a direita
                 4.9500000000000000   # resultado
evalf[17](middlesum(f(x), x = 0..3, 10));   # aproximação pelo pto médio
                 4.5000000000000000   # resultado
evalf[17](int(f(x), x = 0..3));   # integral
                 4.5000000000000000   # resultado
```

A função evalf() faz a aproximação decimal e pode determinar a quantidade de dígitos decimais entre colchetes.

Exemplo 2:

```
> f := x -> x;     # função de entrada
                    x -> x

> evalf[17](leftsum(f(x), x = -3..3, 10));      # aproximação a esquerda
                    -1.8000000000000000      # resultado

> evalf[17](rightsum(f(x), x = -3..3, 10));     # aproximação a direita
                     1.8000000000000000      # resultado

> evalf[17](middlesum(f(x), x = -3..3, 10));    # aproximação pelo pto médio
                     0.      # resultado

> evalf[17](int(f(x), x = -3..3));   # integral
                     0.      # resultado
```

Ao compararmos os dados de entrada do exemplo 1 com os do exemplo 2, e os resultados obtidos, podemos notar que a resposta correta seria se os resultados deste fossem o dobro daquele por pegarmos a mesma área acima (entre 0 (zero) e 3) e abaixo (entre -3 e 0 (zero)) do eixo x para a mesma função. Portanto, para obtermos o resultado esperado, colocamos a função em módulo – usando a função abs() – como podemos verificar a seguir.

```
> f := x -> x;     # função de entrada
                    x -> x

> evalf[17](leftsum(abs(f(x)), x = -3..3, 10));   # aproximação a esquerda
                     9.0000000000000000      # resultado

> evalf[17](rightsum(abs(f(x)), x = -3..3, 10));    # aproximação a direita
                     9.0000000000000000      # resultado

> evalf[17](middlesum(abs(f(x)), x = -3..3, 10));   # aproximação pelo pto médio
                     9.0000000000000000      # resultado

> int(abs(f(x)), x = -3..3);    # integral
                     9     # resultado
```

Exemplo 3:

```
> f := x -> sin(x);     # função de entrada
                    x -> sin(x)

> evalf[17](leftsum(abs(f(x)), x = 0..6.28, 20));    #aproximação a esquerda
                     3.9670729866940288      # resultado
```

```
evalf[17](rightsum(abs(f(x)), x = 0..6.28, 20));  #aproximação a direita
                3.9680731714570741      # resultado

evalf[17](middlesum(abs(f(x)), x = 0..6.28, 20)); #aproximação pelo pto médio
                4.0164724045072413      # resultado

int(abs(f(x)), x = 0..6.28);  #integral
                3.999992390      # resultado
```

No exemplo 3 colocamos a função seno(x) em módulo, assim como no exemplo anterior.

Para a realização das comparações dos resultados obtidos, vamos fazer testes com as mesmas entradas (função, limites inferior e superior e quantidade de subintervalos) para o Maple e para o software.

Teste 1

f(x) = x

limite inferior = 0

limite superior = 3

subintervalos = 10

Saídas	Maple	Software
Esquerda	4.05	4.05
Direita	4.95	4.95
Médio	4.5	4.5
Integral	4.5	

Teste 2

f(x) = x

limite inferior = -3

limite superior = 3

subintervalos = 10

Saídas	Maple	Software
Esquerda	9	9
Direita	9	9
Médio	9	9
Integral	9	

Teste 3

$f(x) = x^2$

limite inferior = -3

limite superior = 3

subintervalos = 100

Saídas	Maple	Software
Esquerda	18.0036	18.0036
Direita	18.0036	18.0036
Médio	17.9982	17.9982
Integral	18	

Teste 4

$f(x) = x^3$

limite inferior = -1

limite superior = 1

subintervalos = 100

Saídas	Maple	Software
Esquerda	0.5002	0.5002
Direita	0.5002	0.5002
Médio	0.4999	0.4999
Integral	0.5	

Teste 5

$f(x) = \sqrt{x}$

limite inferior = 0

limite superior = 5

subintervalos = 50

Saídas	Maple	Software
Esquerda	7.3353689245397908	7.3353689245397908
Direita	7.5589757222897698	7.5589757222897698
Médio	7.4553922186600689	7.4553922186600689
Integral	7.4535599249992989	

Teste 6

f(x) = sen(x)

limite inferior = 0

limite superior = 6.28

subintervalos = 20

Saídas	Maple	Software
Esquerda	3.9670729866940287	3.9670729866940286
Direita	3.9680731714570741	3.9680731714570739
Médio	4.0164724045072412	4.016472404507241
Integral	3.999992390	

Teste 7

f(x) = cos(x)

limite inferior = -4.71

limite superior = 4.71

subintervalos = 20

Saídas	Maple	Software
Esquerda	5.9879156702489561	5.9879156702489562
Direita	5.9879156702489561	5.9879156702489562
Médio	6.0056569836687872	6.0056569836687872
Integral	5.999994293	

Teste 8

f(x) = |x|

limite inferior = -15

limite superior = 15

subintervalos = 200

Saídas	Maple	Software
Esquerda	225	225
Direita	225	225
Médio	225	225
Integral	225	

Teste 9

f(x) = $\sqrt{x}$ e x^2

limite inferior = 0

limite superior = 1

subintervalos = 50

Saídas	Maple	Software
Esquerda	0.33269534221241990	0.3326953422124199
Direita	0.33269534221241990	0.3326953422124199
Médio	0.33353055199387560	0.3335305519938756
Integral	0.33333333333333334	

Os resultados dos testes não foram arredondados e nem truncados. No Maple foram geradas 20 casas decimais para os resultados e usadas as 17 primeiras casas para os testes. Para fazer os cálculos do software foi usada a classe BigDecimal, por manipular números com uma sucessão arbitrariamente longa de dígitos e ser usada em cálculos que requerem precisão em seus resultados. Porém, esta classe não faz sobrecarga dos operadores matemáticos, sendo necessário o uso dos métodos da mesma (BLOCH, 2008; DEITEL; DEITEL, 2005, p.134).

Ao verificarmos os resultados dos testes efetuados podemos concluir que, com exceção dos testes com o sen(x) e cos(x), eles se equivalem. Na linguagem Java estes métodos pertencem à classe Math e retornam um valor do tipo double. Segundo Deitel e Deitel (2005, p.134), o uso de números de ponto flutuante (double ou float) pode causar erro de precisão e resultar em valores incorretos.

A tabela 3.5.2 mostra a comparação para as funções sen(x) e cos(x) efetuadas com o Maple e com a função de Java.

Tabela 3.5.2: Diferença de resultados das funções sen(x) e cos(x).

entrada	**Maple**	**Software**
sen(1)	0.84147098480789650665	0.8414709848078965
cos(1)	0.54030230586813971740	0.5403023058681398

Podemos notar que o Maple trabalha com um retorno maior de casas decimais para estas funções e, portanto, isto pode ser a causa da diferença nos resultados. Esta diferença também pode acontecer com as funções $\sqrt{x}$ e $\sqrt[3]{x}$ por também usarem ponto flutuante.

CONCLUSÕES

Neste trabalho, além do conhecimento teórico obtido com a modelagem e implementação do software para calcular áreas usando a Soma de Riemann para uma ou mais funções predeterminadas, outras contribuições importantes foram o estudo de casos da Soma de Riemann, estudo da API JFreeChart para gerar os gráficos e do Maple para realizar as comparações dos resultados obtidos.

Durante o projeto, surgiram algumas dificuldades. Primeiramente quanto à geração de gráficos combinados (barra e linha), por nos parecer limitada a disponibilidade de materiais de pesquisa para este tipo de gráfico, sendo até então a Internet único meio para tal. Para a solução deste problema foram realizados testes com a utilização de exemplos de códigos encontrados até a obtenção do gráfico apropriado.

Houve ainda uma impossibilidade na implementação de um método regular para o cálculo da Integral para funções impróprias ($1/x$, $1/x^2$) e trigonométricas (sen(x), cos(x)). Para outros tipos de funções (x, x^2, x^3, 2x, $\sqrt{x}$, $\sqrt[3]{x}$) do software, a função integral desenvolvida correspondeu aos resultados dos testes feitos com o Maple para quaisquer valores de entrada.

Num trabalho futuro, pode-se usar a biblioteca Class MathEvaluator que dispõe um mecanismo para avaliar expressões algébricas (CLASS, 2008) e, com isso generalizar a entrada do usuário para resolver a Soma de Riemann para quaisquer funções. Um outro fator importante é implementar um método regular para calcular a Integral.

REFERÊNCIAS

ANDRADE, Lenimar Nunes de. **Introdução à Computação Algébrica com o Maple**. Rio de Janeiro: Sociedade Brasileira de Matemática, 2004.

ANTON, Howard. **Cálculo:** um novo horizonte. 6. ed. Porto Alegre: Bookman, 2002.

BLOCH, Joshua. **Effective Java:** The Java series from the source. 2. ed. Boston: Addison-Wesley Professional, 2008.

BOOCH, Grady; JACOBSON, Ivar; RUMBAUGH, James. **UML:** guia do usuário. Rio de Janeiro: Elsevier, 2005.

CLASS MathEvaluator. Disponível em: < http://www.jaxfront.com/member/java-core-api/com/jaxfront/core/rule/MathEvaluator.html >. Acesso em 27 de outubro de 2008.

DEITEL, Harvey. M.; DEITEL, Paul. J. **Java:** como programar. 6. ed. São Paulo: Pearson Prentice Hall, 2005.

FINNEY, Ross L.; GIORDANO, Frank R.; WEIR, Maurice D. **Cálculo George B. Thomas**. 10. ed. São Paulo: Pearson, 2005.

FOWLER, Martin. **UML essencial:** um breve guia para a linguagem-padrão de modelagem de objetos. 3. ed. Porto Alegre: Bookman, 2005.

GUEDES, Gillianes T. A. **UML:** uma abordagem prática. 2. ed. São Paulo: Novatec Editora, 2006.

GUIDORIZZI, Hamilton Luiz. **Um curso de cálculo**. 5. ed. Rio de Janeiro: Livros Técnicos e Científicos, 2007.

HAFTENDORN, Dörte. **Bernhard Riemann.** Disponível em: <http://rzserv2.fhlueneburg.de/u1/gym03/englpage/chronik/riemann/riemann.htm>. Acesso em 20 de março de 2008.

LARMAN, Craig. **Utilizando UML e padrões:** uma introdução à análise e ao projeto orientados a objetos. Porto Alegre: Bookman, 2002.

MARIANI, Viviana Cocco. **Maple:** fundamentos e aplicações. Rio de Janeiro: LTC – Livros Técnicos e Científicos, 2005.

OBJECT REFINERY LIMITED. **JFree.** Disponível em: <http://www.jfree.org>. Acesso em 15 de março de 2008.

STEWART, James. **Cálculo**. 4. ed. São Paulo: Pioneira Thomson Learning, 2006.

APÊNDICE A – Métodos da Soma de Riemann e da geração de gráficos

A seguir são apresentados os métodos para fazer a Soma de Riemann nos quatro casos e para gerar o gráfico combinado (barra e linha).

Para calcular a Soma de Riemann, foi desenvolvido o seguinte método:

```
1   //método para calcular a Soma de Riemann nos quatro casos
2   public void somaDeRiemann(){
3       //inicializa os resultados
4       resultEsq = BigDecimal.ZERO; //resultEsq = 0;
5       resultDir = BigDecimal.ZERO;
6       resultMed = BigDecimal.ZERO;
7       resultRand = BigDecimal.ZERO;
8       BigDecimal ci = BigDecimal.valueOf(a); //ci = a
9
10      for(int contador=0;contador<n;contador++){
11          //esquerda -> |f(ci)|
12          resultEsq = resultEsq.add(funcao(ci).abs());
13
14          //direita -> |f(delta_x+ci)|
15          resultDir = resultDir.add(funcao(delta_x.add(ci)).abs());
16
17          //médio -> |f(ci+(delta_x+ci))/2)|
18          resultMed = resultMed.add(funcao((ci.add(delta_x.add(ci))).
            divide(BigDecimal.valueOf(2)).abs()));
19
20          //randômico -> |f(ci+delta_x . random())|
21          resultRand = resultRand.add(funcao(ci.add(delta_x.multiply
            (BigDecimal.valueOf(Math.random()))))).abs());
22
23          ci = ci.add(delta_x); //atualiza o valor de ci -> ci = ci + delta_x
24      }
25      resultEsq = resultEsq.multiply(delta_x);  //resultEsq = resultEsq . delta_x
26      resultDir = resultDir.multiply(delta_x);
27      resultMed = resultMed.multiply(delta_x);
28      resultRand = resultRand.multiply(delta_x);
29  }
```

Para gerar o gráfico combinado de exemplo da figura A.1, foi implementado os seguintes métodos:

```
1   //método para juntar os gráficos de barra e linha
2   private JFreeChart juntaGraficosBarraLinha() {
3       //ValueAxis: exibe dados de valor
4       //NumberAxis(nome do eixo): cria um eixo de números
5       ValueAxis eixoX = new NumberAxis("x");
6       ValueAxis eixoY = new NumberAxis("y");
7
8       //IntervalXYDataset: permite definir dados para valores de x, y ou ambos(x,y).
```

```
9           IntervalXYDataset grafBarra = criaGraficoDeBarra();
10
11          //XYItemRenderer: faz a representação visual de um único item (x, y).
12          //XYBarRenderer(margem de porcentagem para aparar da largura de cada
            barra): cria um renderer.
13          XYItemRenderer renderer1 = new XYBarRenderer(0.02);
14
15          //cria um plot(para o gráfico completo) com o gráfico, eixos e rendereriza.
16          XYPlot plot = new XYPlot(grafBarra, eixoX, eixoY, renderer1);
17
18          //adiciona o segundo gráfico e renderiza...
19          //XYDataset: acessa dados na forma (x, y).
20          XYDataset grafLinha2 = criaGraficoDeLinha();
21
22          //StandardXYItemRenderer: cria um render
23          XYItemRenderer renderer2 = new StandardXYItemRenderer();
24
25          //setDataset(posição, gráfico) - coloca o gráfico na posição do plot
26          plot.setDataset(2, grafLinha2);
27          //setRenderer(posição, render) - coloca a renderização na posição do plot
28          plot.setRenderer(2, renderer2);
29
30          //ordem de retribuição dos gráficos
31          plot.setDatasetRenderingOrder(DatasetRenderingOrder.FORWARD);
32
33          //JFreeChart(legenda, título, gráfico, criar legenda): cria o gráfico completo.
34          return new JFreeChart(null, null, plot, true);
35  }

36  //método para criar o gráfico de barra
37  private IntervalXYDataset criaGraficoDeBarra() {
38          // XYSeries(nome): cria série para valores (x, y)
39          XYSeries xy = new XYSeries("barra");
40
41          //adiciona dados na série
42          xy.add(.1, .1);
43          xy.add(.3, .3);
44          xy.add(.5, .5);
45          xy.add(.7, .7);
46          xy.add(.9, .9);
47
48          //cria coleção de dados com séries (x, y)
49          XYSeriesCollection colecaoDadosGrafBarras = new XYSeriesCollection(xy);
50          //setIntervalWidth(double) - estabelece a largura das barras
51          colecaoDadosGrafBarras.setIntervalWidth(.2);
52          return colecaoDadosGrafBarras;
53  }

54  //método para criar o gráfico de linha
55  private XYDataset criaGraficoDeLinha() {
```

```
56              Function2D funcao = new Function2D() {
57                      public double getValue(double x) {
58                              return x;
59                      }
60              };
61              //parâmetros: função, valor inicial, valor final, quantidade de pontos, nome
62              return DatasetUtilities.sampleFunction2D(funcao, 0, 1, 100, "linha");
63      }
```

Figura A1: Exemplo de gráfico gerado.