Análise de linguagens de programação para a resolução do processo de Gram-Schmidt

Mathaus C. Huber

[1]Universidade Federal de Pelotas (UFPEL) – Discente do Curso Superior de Ciência da Computação
R. Gomes Carneiro, 1 - Centro – 96075-630 – Pelotas – RS – Brazil

mchuber@inf.ufpel.edu.br

Abstract. *This article describes the research project on programming languages developed in the subject of Programming Language Concepts, given to the fifth semester of the Computer Science course, at the Federal University of Pelotas, which refers to the analysis of the best efficiency of programming languages. programming for solving the Gram-Schmidt process. In which, we will use some mathematical languages, since the Gram-Schmidt process is a method for orthonormalization of a set of vectors in a space with an inner product.*

Resumo. *Este artigo descreve o projeto de pesquisa em linguagens de programação desenvolvida na disciplina de Conceitos de Linguagem de Programação, conferida ao quinto semestre do curso de Ciência da Computação, da Universidade Federal de Pelotas, no qual se refere a análise da melhor eficiência de linguagens de programação para a resolução do processo de Gram-Schmidt. No qual, utilizaremos algumas linguagens matemáticas, visto que o processo de Gram-Schmidt é um método para ortonormalização de um conjunto de vetores em um espaço com produto interno.*

1. Introdução

Antes de entrarmos no conceito da eficiência de linguagens de programação para a solução do processo de Gram-Schmidt, iremos elucidar um pouco sobre o problema proposto neste projeto de pesquisa, de forma a melhor compreensão das técnicas utilizadas posteriormente. Logo, este método consiste, em um algoritmo para obter uma base ortogonal (ou ortonormal) a partir de uma base qualquer. De maneira mais geral, o método permite transformar um conjunto de vetores linearmente independentes em um conjunto ortogonal que gera o mesmo espaço vetorial.

Partindo do princípio que não existe melhor ou pior linguagem, e sim, aquela que nos proporciorna uma maior gama de recursos e flexibilidade para solucionar determinado problema. Para fins de comparação, usaremos alguns critérios para definir a melhor eficiência, entre as linguagens escolhidas, para a resolução do processo de Gram-Schmidt. Previamente, definimos as linguagens testes levando em consideração as mais usadas no nicho da matemática e álgebra linear, sendo elas: C, Java, Python, Javascript, Fortran e C++.

Um dos critérios definidos foi a legibilidade, pois, considero um dos critérios mais importantes de linguagens de programação, visto que, é facilidade com a qual os programas podem ser lidos e entendidos, portanto, a sintaxe de uma linguagem de programação tem bastante impacto na legibilidade. Algumas linguagens, a exemplo de C, consideram muitos recursos de escrita, e a multiplicidade de recursos pode ser um problema para a legibilidade, a exemplo disso, é possível incrementar uma variável de 4 formas diferentes na linguagem. O contrário, porém, também é um problema. Imagine uma linguagem com poucos recursos de escrita, como Assembly, por exemplo. Ela aceita comandos primitivos, que devem ser executados um a um. A parte da sobrecarga de operadores, como já visto em aula, deve ser levada em conta também, pois, linguagens que permitem a sobrecarga de operadores e de métodos também podem comprometer a legibilidade se não forem usadas de forma intuitiva.

É levado em conta, também, a ortogonalidade, pois, a falta de ortogonalidade leva a exceções às regras de linguagem. A ortogonalidade é fortemente relacionada à simplicidade: quanto mais ortogonal o projeto de uma linguagem, menor é o número necessário de exceções às regras da linguagem. Menos exceções significam um maior grau de regularidade no projeto, o que torna a linguagem mais fácil de aprender, ler e entender.

A facilidade de escrita é o critério que diz quão fácil é escrever uma solução usando determinada linguagem de programação. O primeiro fator é o contexto do nosso problema. Não dá para comparar uma linguagem que não foi projetada para determinada função com uma que foi e medir a facilidade de escrita das duas, por isso, neste projeto de pesquisa, visamos a comparação de linguagens projetadas aproximadamente na mesma área.

O suporte para a abstração é um fator fundamental no projeto de uma linguagem de programação. O grau de abstração permitido por uma linguagem de programação e a naturalidade de sua expressão são importantes para sua facilidade de escrita. Visando um exemplo, no escopo do projeto, se formos implementar uma arvore binária que armazena dados inteiros em seus nós. Tal árvore poderia ser implementada em uma linguagem que não oferece suporte a ponteiros e gerenciamento de memória dinâmica usando um monte (heap), como na linguagem Fortran. Em C++ e Java, essas árvores podem ser implementadas utilizando uma abstração de um nó de árvore na forma de uma simples classe com dois ponteiros (ou referências) e um inteiro.

A confiabilidade de uma linguagem também deve ser levada em conta, e a mesma está associada aos testes executados por ela os limites impostos no projeto da linguagem. Um exemplo, que se optarmos por usar a linguagem Java teríamos uma maior confiabilidade, pois, ela não permite ao programador acessar ou usar ponteiros como em C++. Ao invés de permitir o acesso a posições de memória, ela permite apenas a referência. Outro exemplo é a checagem de algumas condições antes de gerar o novo programa com o código, o que nos dá a linguagem Java nos oferece mais fatores a fa-

vor, por um exemplo, se usarmos a linguagem Javascript que é conhecida por não dar pouquíssimos erros de verificação. Isso acaba sendo um problema porque, quando o programa gera um resultado inesperado, é mais difícil encontrar o erro com o programa rodando errado do que ser avisado de um erro semântico por exemplo e saber exatamente o que corrigir.

2. Objetivo

O objetivo dessa pesquisa é identificar qual a melhor linguagem de programação para resolver o processo de Gram-Schmidt, de forma que a mesma tenha mais recursos disponíveis ao programador, a exemplo de bibliotecas matemáticas específicas, ofereça maior legibilidade e supra a necessidade de verificar se essas fórmulas produzem uma sequência ortogonal.

3. Metas

- Encontrar fundamentos teóricos na literatura indicando uma linguagem de programação mais adequada para a implementação do processo de Gram-Schmidt, e que seja bem avaliada ao seguir os critérios de avaliação definidos por Sebesta (2011);
- Implementar o algoritmo do processo de Gram-Schimdt nas diferentes linguagens definidas anteriormente para teste, de forma a comparar o uso dos critérios definidos no livro do Sebesta, e especificados na introdução do projeto de pesquisa.
- Fazer uma comparação e análise estatística dos resultados gerados por cada um dos algoritmos, definição do tempo empregado em cada um deles, de forma a definir a complexidade de implementação nas linguagens selecionadas.

4. Resultados esperados

Serão obtidos as análises das ferramentas disponíveis em cada uma das linguagens teste, para a solução do processo de Gram-Schimdt, além disso, a facilidade de escrita do algoritmo, tempo empregado em tratamento de erros, e resultado final gerado pelo algoritmo e comparação dos erros de arredondamento.

5. Metodologia

A metodologia da pesquisa começará pela **Revisão Bibliográfica**, onde será feita um maior aprofundamento das pesquisas já feitas, explorando melhor as capacidades de algumas linguagens de programação na implementação das estruturas usadas na resolução do processo de Gram-Schimdt. As linguagens que serão testadas são algumas das mais populares dentro da categoria de linguagens de programação que possuem mais suporte à matemática, como C, Java, Python, Javascript, Fortran e C++, além de ficar aberto o estudo de outras linguagens identificadas como interessante durante o processo de pesquisa. Essas linguagens também serão avaliadas quanto aos três critérios de avaliação definidos por Sebesta (2011), sendo elas Legibilidade, Facilidade de Escrita e Confiabilidade.

Essa é uma etapa que se estenderá durante quase todo processo de pesquisa com o intuito de continuar a identificar abordagens ainda não identificadas.

A segunda fase da metodologia é uma **Fase Exploratória**, onde baseado nos conhecimentos adquiridos na revisão bibliográfica, é iniciado a perquirição de técnicas objetivas para embasar a tese do projeto. De forma que escolhemos quais linguagens e ferramentas serão usadas para testar e implementar o algoritmo do processo de Gram-Schmidt.

A próxima fase é referente a **Implementação**, onde será utilizada as técnicas descobertas na fase exploratória para o desenvolvimento do algoritmo. Com a análise prévia feita para o projeto de pesquisa já é possível identificar as linguagens C, Java, Python, Fortran, Javascript e C++ como desejáveis de serem testadas e analisadas na pesquisa.

Quase em paralelo à implementação, teremos a etapa de **Coleta e Análise Preliminar dos Resultados**, onde através dos resultados do código fonte, gerado pelo algoritmo do processo de Gram-Schmidt, teremos uma análise estatística na qual poderemos prever, através das técnicas adquiridas no processo de Revisão Bibliográfica e Fase Exploratória, qual a linguagem de programação possui mais recursos e é mais eficaz para a solução do problema proposto.

Nos últimos dois meses da etapa de pesquisa, abrange o período de **Ajustes Finais e Reflexão**, onde teremos os resultados fornecidos pela Coleta e Análise do que foi implementado até então. Com esses resultados, consideraremos necessário uma reflexão crítica, de qual linguagem ofereceu maior suporte ao desenvolvedor, quais se mostraram ineficientes na solução do processo de Gram-Schimdt, e se, de fato, os resultados da pesquisa em linguagens para o problema se mostrou vantajosa.

6. Cronograma

	1	2	3	4	5	6	7	8	9	10	11	12
Revisão Bibliográfica	X	X	X	X	X	X	X	X	X	X		
Fase Exploratória			X	X								
Implementação					X	X	X	X	X			
Coleta e Análise Preliminar dos Resultados						X	X	X	X	X		
Ajustes Finais e Reflexão											X	X

7. Impactos Esperados

Em geral, os impactos que esperamos nesse projetos de pesquisa, é uma reflexão crítica dos resultados obtidos na performance de linguagens de programação, escolhidas de forma adequada, para serem testadas com os critérios estabelecidos por Sebesta no livro Conceitos de Linguagens de Programação, na qual, o problema proposto, é a solução

do processo de Gram-Schimdt. Lembrando que o custo desse algoritmo é assintoticamente $2nk^2$ operações de ponto flutuante, nas quais n é a dimensionalidade dos vetores, podendo ter um custo computacional maior ou menor em cada uma das linguagens teste, sendo considerado também, o custo total para implementação um dos critérios a serem refletidos.

8. Referências Bibliográficas

8.1. Bibliografia Referenciada

SEBESTA, R. W. **Concepts of Programming Languages**, 9th edition. Pearson Education, Inc, 2011.

MELO, Ana Cristina Vieira de; SILVA, Flávio Soares Corrêa da. **Princípios de linguagens de programação** 2003.

SUSSMAN, G. **Structure and Interpretation of Computer Programs**. Inc, 1985.

BARANAUSKAS, M. C. C. **Procedimento, função, objeto ou lógica? Linguagens de programação vistas pelos seus paradigmas. Computadores e Conhecimento: Repensando a Educação.** Campinas, SP, Gráfica Central da Unicamp, 1993.

8.2. Bibliografia para Consulta

Bau III, David; Trefethen, Lloyd N. **Numerical linear algebra, Philadelphia: Society for Industrial and Applied Mathematics**. 1997

Golub, Gene H.; Van Loan, Charles F. **Matrix Computations**, 3rd ed. , Johns Hopkins. 1996

Greub, Werner **Linear Algebra** 4th ed. , Springer. 1975

Soliverez, C. E.; Gagliano, E. **Orthonormalization on the plane: a geometric approach**, Mex. J. Phys. 1985

Cheney, Ward; Kincaid, David. **Linear Algebra: Theory and Applications.** 2009

Anton, Howard; **Álgebra Linear com Aplicações** - 10a ed. Bookman, 2012

Prasolov, Viktor Vasil'evich; **Problems and Theorems in Linear Algebra.** AMS, 1994;