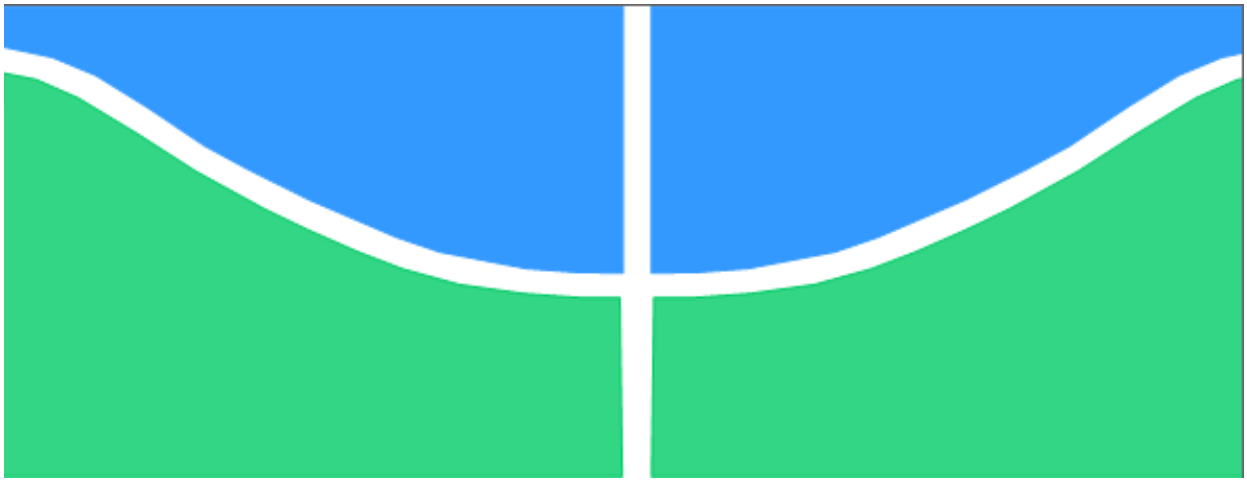

Universidade de Brasília - UnB
Faculdade UnB Gama - FGA
Curso de Engenharia de Software

Métodos Numéricos e Paradigmas de Programação para predição de valores no mercado de moedas.

Autores: Cleiton da Silva Gomes
Vanessa Barbosa Martins
Orientadores: Ricardo Matos Chaim
Milene Serrano

Brasília, DF
2014

RESUMO

O Mercado de Moedas ou Foreign Exchange (FOREX) é o mercado mais ativo e volumoso do mundo. O FOREX é todo eletrônico e abre na segunda-feira pela manhã, no horário da Nova Zelândia, e só fecha às 17 horas de sexta-feira, no horário dos Estados Unidos. Trata-se de um mercado dinâmico, pois as taxas de câmbio podem ser negociadas de qualquer lugar onde houver conexão com a internet. Uma prática bastante difundida é o uso de Experts para realizar operações de compra ou venda no Mercado de Moedas. Com capacidade de ler uma base histórica das cotações das moedas e baseados nesses dados, os Experts são softwares capazes de efetuar cálculos para prever uma cotação em relação a outra. Sendo assim, é possível ganhar ou perder dinheiro com a operação efetuada. Além disso, o Expert pode ser programado para percorrer diferentes instruções como proteção de lucro e realização de prejuizos, além de vários pares de moedas poderem ser utilizados para negociar, como EUR-USD (euro-dólar) e EUR-JPY (euro-yen). As diferentes instruções que um Expert pode utilizar serão programadas em diversas linguagens para explorar o contexto de Paradigmas de Programação e cada implementação será testada usando avaliações unitárias específicas para cada linguagem de programação.

Palavras-chave: FOREX, Expert, Paradigmas de Programação

1. Introdução

1.1 Justificativa

Operar no Mercado de Moedas de forma manual é extremamente perigoso, pois o mercado é muito volátil e o investidor pode perder todo seu capital construído durante muito tempo em apenas alguns minutos. Em diversas situações, o mercado varia as cotações em apenas um minuto, sendo que a mesma variação pode ser feita durante horas. Para contornar esse problema, a plataforma MetaTrader oferece as linguagens MQL4 (Paradigma Estruturado) e MQL5 (Paradigma Orientado a Objetos) para construir Experts que operem de forma automatizada.

A linguagem MQL4 foi espelhada na linguagem C e, evidentemente, também possui características próprias. Se um investidor resolve programar em MQL4 para automatizar estratégias e nunca programou em C, é bem provável que o mesmo terá problemas para automatizar as operações. A mesma analogia é válida para a linguagem MQL5, por sua vez espelhada no C++.

A plataforma MetaTrader não oferece nenhum suporte de ferramentas de teste unitário para as linguagens MQL4 e MQL5. Após implementar um Expert, não é possível implementar testes de unidade para verificar se as instruções programadas estão de acordo com o esperado. A única forma de verificar se o Expert está seguindo as estratégias programadas é usar uma conta real ou demo e operar durante um período específico de tempo.

Não existe nenhuma ferramenta que realize a análise estática de código fonte em MQL4 e MQL5. Portanto, não é possível obter o nível qualidade de código fonte dos Experts programados nessas linguagens.

O código da plataforma MetaTrader é fechado, isto é, não é possível que a comunidade de desenvolvedores colabore com o desenvolvimento da ferramenta.

Diante dos problemas expostos, será desenvolvida uma ferramenta com as melhores práticas da Engenharia de Software para investimento no Mercado de Moedas com código fonte aberto. Essa ferramenta será implementada em diversos Paradigmas de Programação, com os padrões de projetos necessários, testes unitários de todas as implementações e análise qualitativa de código fonte e a mesma substituirá um Expert (implementado em linguagem MQL4 ou MQL5) ou um conjunto de Experts, pois a ferramenta também terá a propriedade de controlar e monitorar um ou mais Experts.

O investidor terá maior segurança em suas operações e a ferramenta ofecerá os métodos numéricos que o mesmo deseje que o Expert use como critério de entrada no Mercado de Moedas. Ao selecionar e confirmar as configurações, o Expert uma vez ativo estará pronto para operar no Mercado de Moedas de forma automatizada. Também será possível desativá-lo, configurar um novo Expert e/ou acompanhar a rentabilidade em tempo real pelo computador ou celular. O único requisito é possuir conexão com a internet.

1.2 Objetivos

Objetivo Geral

Desenvolver uma ferramenta em diversos Paradigmas de Programação que opere no Mercado de Moedas de forma automatizada.

Objetivos Específicos

1) Descrever métodos numéricos de Correlação Linear, Mínimos Quadrados e Fibonacci e evidenciar principais utilidades e aplicações desses métodos no mercado de moedas.
2) Descrever Paradigmas de Programação Estruturado, Orientado a Objetos, Funcional, Lógico e Multiagente e evidenciar possíveis aplicações desses Paradigmas de Programação no contexto do mercado de moedas.
3) Implementar o mesmo Expert em linguagem MQL4 e em Multiparadigma de Programação.
4) Comparar resultados do Expert em linguagem MQL4 e em Multiparadigma de programação em termos de ganho monetário.
5) Levantar métricas de qualidade de código fonte.
6) Realizar análise estática do código fonte do Expert Multiparadigma.
7) Levantar ferramentas para teste unitário em linguagens de programação C, Java, Prolog e Haskell.
8) Realizar teste unitário do código fonte do Expert Multiparadigma.

2. Referêncial teórico

2.1 Métodos Numéricos

2.1.1 Método Mínimos Quadrados

O método de Mínimos Quadrados determina o valor mais provável de quantidades não conhecidas em que a soma dos quadrados das diferenças entre valores observados e computados é mínimo (INÁCIO, 2010).

Usa-se o método de Mínimos Quadrados para determinar a melhor linha de ajuste que passa mais perto de todos os dados coletados, no intuito de obter a melhor linha de ajuste, de forma que minimize as distâncias entre cada ponto de consumo (DIAS,1985, p. 46).

A aplicação do método de Mínimos Quadrados é dado para deduzir a melhor estimativa de mensurações de n medições idênticas (em condições de “repetitividade”) e não idênticas (em condições de “reprodutividade”). O peso estatístico de um resultado é definido (VUOLO, 1996, pág. 149).

O desvio vertical do ponto (Xi, Yi) da reta Y = B0 = B1Xi é a altura do ponto menos altura da reta. A soma dos desvios quadrados verticais dos pontos (X1, Y1), ..., (Xn,Yn) à reta é, portanto, $f(B0,B1) = \Sigma [Yi - (B0 + B1Xi]^2$; $0 <= i < \infty$. As estimativas pontuais de C0 e C1, representadas por K0 e K1 e denominadas estimativa de Mínimos Quadrados, é aquela que minimizam f(B0, B1). Em suma, K0 e K1 são tais que f(K0,K1) <= f(B0,B1) para qualquer B0 e B1. A reta de Regressão Estimativa ou de Mínimos Quadrados é, por conseguinte, a reta cuja equação é Y = K0 + K1X (DEVORE, 2006, pág. 441).

2.1.2 Método de Correlação Linear

Em estudos quem envolvem duas ou mais variáveis, é comum o interesse em conhecer o relacionamento entre elas, além das estatísticas descritivas normalmente calculadas. A medida que mostra o grau de relacionamento entre as variáveis é chamada de Coeficiente de Correlação ou Correlação Linear ou Correlação Linear de Pearson. A Correlação Linear também é conhecida como medida de associação, interdependência, intercorrelação ou relação entre as variáveis (LIRA, 2004).

O Coeficiente de Correlação linear de Pearson (r) é uma estatística utilizada para medir força, intensidade ou grau de relação linear entre duas variáveis aleatórias (FERREIRA, 2009, pág. 664).

Segundo Lopes (2005, pág. 134), a Correlação Linear indica a relação entre duas variáveis. De modo a interpretar o coeficiente de correlação (r), podem ser utilizados os seguintes critérios para classificar os resultados obtidos:

• 0 a 0,50: fraca correlação;
• De 0,51 a 0,84: moderada correlação;
• A partir de 0,85: forte correlação.

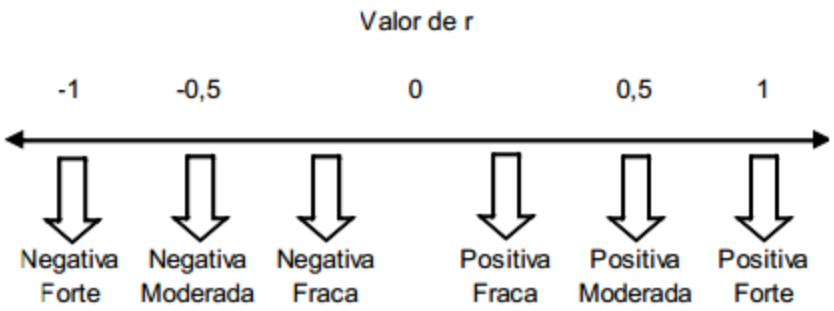

Figura X: Classificação da Correlação Linear
Fonte: Lopes (2005, pág. 134).

Segundo Regra (2010), a Correlação Linear revela o grau de associação entre duas variáveis aleatórias. A dependência de duas variáveis X e Y é dada pelo Coeficiente de Correlação Amostral, conhecido também por coeficiente r-de-Pearson. Designa-se, normalmente, por r e é determinado de acordo com a figura X.

$$r = \frac{Cov(X,Y)}{s_X . s_Y}$$

$$Cov(X,Y) = \frac{\sum_{i=1}^{n} XY}{n} - \overline{X}\,\overline{Y} \ ,$$

$$s_X = \sqrt{\frac{\sum_{i=1}^{k} f_i \left(X_i - \overline{X}\right)^2}{n}} \ \text{e} \ \ s_Y = \sqrt{\frac{\sum_{i=1}^{k} f_i \left(Y_i - \overline{Y}\right)^2}{n}}$$

Figura X: Determinação da Correlação Linear
Fonte: Regra (2010).

Segundo Viale (2009, pág. 8), as propriedades mais importantes do Coeficiente de Correlação são:

1. O intervalo de variação vai de -1 a +1.
2. O coeficiente é uma medida adimensional, isto é, é independente das unidades de medida das variáveis X e Y.
3. Quanto mais próximo de +1 for “r”, maior o grau de relacionamento linear positivo entre X e Y, ou seja, se X varia em uma direção, Y variará no mesmo sentido.
4. Quanto mais próximo de -1 for “r”, maior o grau de relacionamento linear negativo entre X e Y, isto é, se X varia em um sentido Y variará na direção inversa.
5. Quanto mais próximo de zero estiver “r”, menor será o relacionamento linear entre X e Y. Um valor igual a zero indicará ausência apenas de relacionamento linear.

A análise da Correlação Linear fornece um número, indicando como duas variáveis variam conjuntamente e mede a intensidade e a direção da relação linear ou não-linear entre duas variáveis. Fornece também um indicador que atende à necessidade de estabelecer a existência ou não de uma relação entre essas variáveis sem que, para isso, seja preciso o ajuste de uma função matemática. Em suma, o grau de variação conjunta entre X e Y é igual ao de Y e X (LIRA, 2004).

2.2 Paradigmas de Programação

A palavra paradigma significa aquilo que pode ser utilizado como padrão, de forma que é um modelo a ser seguido (FERREIRA,1986). Segundo Kurt Normark (2013), professor da Universidade de Aalborg na Dinamarca, paradigma de programação é um padrão que serve como uma escola de pensamentos para a programação de computadores.

Paradigmas se diferem nos conceitos e abstrações usadas para representar os elementos de um programa como objetos, funções, variáveis e restrições. Também divergem nas etapas que compõem um cálculo como atribuição, avaliação, continuações e os fluxos de dados. Algumas linguagens são projetadas para suportar um paradigma específico, enquanto que outras linguagens de programação suportam múltiplos paradigmas como C++, Object Pascal e Python (PAQUET; MOKHOV, 2010).

Uma linguagem de programação multi-paradigma é uma linguagem de programação que suporta mais de um paradigma de programação. O objetivo tais linguagens é permitir que programadores usem a melhor ferramenta para o trabalho, admitindo que nenhum paradigma resolve todos os problemas da maneira mais fácil ou mais eficiente (PAQUET; MOKHOV, 2010).

2.2.1 Paradigma Procedural

Programação procedural é um paradigma de programação, derivado da programação estruturada, com base no conceito da chamada de procedimento. Procedimentos, também conhecidos como rotinas, sub-rotinas, métodos ou funções contêm uma série de passos computacionais a serem realizados. Qualquer procedimento pode ser chamado a qualquer momento durante a execução de um programa, inclusive por outros procedimentos ou a si mesmo (PAQUET; MOKHOV, 2010).

A linguagem de programação procedural fornece ao programador um meio de definir com precisão cada passo na execução de uma tarefa e é muitas vezes uma escolha melhor em situações que envolvem complexidade moderada ou que requerem significativa facilidade de manutenção (PAQUET; MOKHOV, 2010).

As linguagens imperativas foram desenvolvidas em torno da arquitetura de computadores prevalentes na época, chamada de arquitetura von Neumann, criada pelo matemático húngaro John von Neumann (SEBESTA, 2012, pág. 18).

Em um computador de von Neumann, ambos os dados e programas são armazenados na mesma memória. A unidade central de processamento (CPU), que executa as instruções, é separada da memória. Portanto, as instruções e os dados devem ser transmitidos da memória para a CPU. Os resultados das operações na CPU deve ser devolvidos à memória (SEBESTA, 2012, pág. 18).

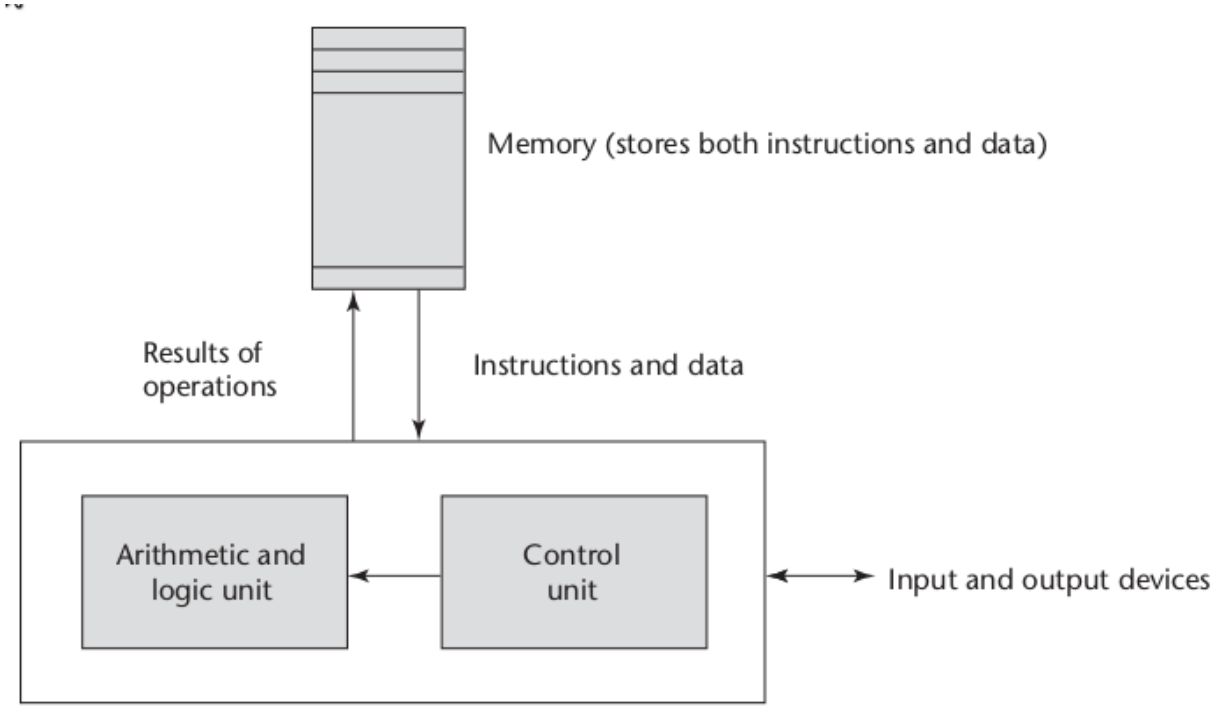

Figura X: Arquitetura de von Neumann
Fonte: Sebesta, (2012, pág. 19).

Por causa da arquitetura de von Neumann, os recursos centrais das linguagens imperativas são as variáveis, as quais modelam as células de memória, as instruções de atribuição, baseadas na operação de canalização (piping) e a forma interativa de repetição, o método mais eficiente dessa arquitetura. A iteração é rápida nos computadores de von Neumann porque as instruções são armazenadas em células adjacentes da memória. Esta eficiência desencoraja o uso de recursão para repetição, embora a recursão seja frequentemente mais natural (SEBESTA, 2012, pág. 18).

Linguagens imperativas contêm variáveis e valores inteiros, operações aritiméticas básicas, comandos de atribuição, sequenciamentos de comandos baseados em memórias, condições e comandos de ramificação. Também suportam determinadas características comuns que surgiram com a evolução do paradigma: estruturas de controle, entrada/saídas, manipulação de exceções e erros, abstração procedural, expressões e atribuição, suporte de biblioteca para estruturas de dados (TUCKER; NOONAN, 2009).

Fortran (FORmula TRANslation) foi a primeira linguagem de alto nível a ganhar ampla aceitação, sendo esta uma linguagem imperativa. Projetada para aplicações científicas, conta com notação algébrica, tipos, subprogramas e entrada/saída formatada. Foi implementada em 1956 por John Backus na IBMespecificamente para a máquina IBM 704. A execução eficiente foi uma grande preocupação, consequentemente, sua estrutura e comandos têm muito em comum com linguagens de montagem (BROOKSHEAR, 2013, pág 458).

Outra linguagem de programação é o C. Segundo Dennis M. Ritchie, um dos criadores da linguagem, ela se tornou uma linguagem dominante na década de 90. Seu sucesso deve ao sucesso do Unix, um sistema operacional implementado em C. Além disso a linguagem é simples e pequena, traduzível com simples e pequenos compiladores. Seus tipos e operações são bem fundamentados naquelas fornecidas por máquinas reais, e para pessoas que usam o computador para trabalhar, aprender a linguagem para gerar programas em tempo e espaço eficientes não é difícil. Ao mesmo tempo a linguagem é suficientemente abstrata dos detalhes da máquina de modo que a portabilidade de programa pode ser alcançada (RITCHIE, 1996).

2.2.2 Paradigma Orientado a Objetos

Um objeto é a representação de uma "coisa" (alguém ou algo), e esta representação é expressa com a ajuda de uma linguagem de programação. A coisa pode ser um objeto real, ou algum conceito mais complicado. Tomando um objeto comum, como um gato, por exemplo, você pode ver que ele tem certas características (cor, nome, peso) e pode executar algumas ações (miar, dormir, esconder, fugir). As características do objeto são chamados de propriedades em OO (Orientação a Objetos) e as ações são chamadas de métodos (STEFANOV, 2008).

A programação orientada a objetos fornece um modelo no qual um programa é uma coleção de objetos que interagem entre si, passando mensagens que transformam

seu estado. Neste sentido, a passagem de mensagens permite que os objetos dados se tornem ativos em vez de passivos (TUCKER; NOONAN, 2009).

Classes servem como modelos a partir dos quais os objetos podem ser criados. Elas tem precisamente as mesmas variáveis e operações de instâncias dos objeto, mas sua interpretação é diferente: Onde um objeto representa variáveis reais, variáveis de classe são em potencial, instanciadas apenas quando um objeto é criado (WEGNER,1990).

Uma nova classe (designada por subclasse derivada) pode ser derivada de outra classe (designada por superclasse) por um mecanismo chamado de herança. A classe derivada herda todas as características da classe base: a sua estrutura e comportamento (resposta a mensagens). Além disso, a classe derivada pode conter estado adicional (variáveis de instância), e pode apresentar um comportamento adicional (novos métodos para responder novas mensagens). Significativamente, a classe de derivados também podem substituir o comportamento correspondente a alguns dos métodos da classe base: não seria um método diferente para responder à mesma mensagem. Além disso, o mecanismo de herança é permitido mesmo sem acesso ao código-fonte da classe base. Herança dá a OO seu benefício chefe sobre outros paradigmas de programação - a reutilização de código relativamente fácil sem a necessidade de alterar o código fonte existente (LEAVENS, 2014).

2.2.3 Paradigma Funcional

O centro da programação funcional é a idéia de uma função. A linguagem de programação funcional dá um modelo simples de programação: uma valor, o resultado é calculado com base em outros valores, as entradas da função. Por causa de sua fundação simples, uma linguagem funcional dá uma visão mais clara das idéias centrais da computação moderna, incluindo abstração, polimorfismo e sobrecarga (THOMPSON,1999, pág. 16).

A programação funcional exige que as funções sejam de primeira classe , o que significa que elas são tratados como quaisquer outros valores e podem ser passados como argumentos para outras funções ou ser retornado como um resultado de uma função. Sendo de primeira classe também significa que é possível definir e manipular funções dentro de outras funções (Desconhecido, 2013).

Uma linguagem de programação puramente funcional não usa avariáveis ou instruções de atribuição. Isso libera o programador de preocupar-se com as células da memória do compilador no qual o programa é executado. Sem variáveis, construções interativas não são possíveis, porque elas são controladas por variáveis. A repetição deve ser feita por meio de recursão, não por meio de laços (SEBESTA, 2008).

A primeira linguagem de programação funcional foi inventada para oferecer recursos de linguagem para processamento de listas, cuja necessidade surgiu a partir das primeiras aplicações na área da inteligência artificial (TUCKER; NOONAN, 2012).

2.2.4 Paradigma Lógico

Referências

BROOKSHEAR, J. Glenn. **Ciência da Computação: Uma Visão Abrangente**. 3 ed. São Paulo: Bookman, 2003.

Desconhecido. **Functional Programming.** Dísponivel em <http://www.haskell.org/haskellwiki/Functional_programming>. Acesso em: 24 de ago. 2014.

FERREIRA, Aurélio B. de Hollanda. **Novo Dicionário da Língua Portuguesa**. 2. ed. Rio de Janeiro: Nova Fronteira, 1986.

LEAVENS, Gary T. **Major programming paradigms.** Dísponivel em <http://www.eecs.ucf.edu/~leavens/ComS541Fall97/hw-pages/paradigms/major.html#object>. Acesso em: 18 de ago. 2014.

NORMAK, Kurt. **Programming Paradigms**. Dísponivel em <http://people.cs.aau.dk/~normark/prog3-03/html/notes/paradigms_themes-paradigms.html#paradigms_the-word_title_1>. Acesso em: 08 de ago. 2014.

PAQUET, Joey; MOKHOV, Serguei A. **Comparative Studies of Programming Languages**. Montreal, ago. 2010.

RITCHIE, Dennis M. **O Desenvolvimento da Linguagem C**.Dísponivel em <http://cm.bell-labs.com/cm/cs/who/dmr/chistPT.html>. Acesso em: 20 de ago. 2014.

SEBESTA, Robert W. **Concepts of programming languages**. 10 ed. New York: Pearson, 2012.

STEFANOV, Stoyan. **Object-Oriented JavaScript**. Olton: Packt Publishing, 2008.

THOMPSON, Simon. **Haskell: The Craft of Functional Programming**. 2 ed. New York: Addison-Wesley, 1999.

TUCKER, Allen B.; NOONAN Robert E. **Linguagens de programação : Princípios e paradigmas**. 2 ed. São Paulo: McGran-Hill, 2009.

WEGNER, Peter. **Concepts and paradigms of object-oriented programming.** http://dl.acm.org/citation.cfm?id=383004

DEVORE, Jay L. **Probabilidade e Estatística para Engenharia e Ciências**, 6.ed. São Paulo: Pioneira Thomson Learning, 2006.

DIAS, Marco A. P. **Administração de materiais:** uma abordagem logística. 2.ed. Sao Paulo:
Atlas, 1985. 523p.

FERREIRA, D. F. **Estatística básica**. 2. ed. Lavras: UFLA, 2009.

INÁCIO, José Francisco Secorun. **Análise do Estimador de Estado por Mínimos Quadrados Ponderados**. Trabalho de Conclusão de Curso, UFRS - Universidade Federal do Rio Grande do Sul, 2010.

LIRA, Sachiko Araki. **Análise Correlação: Abordagem Teórica e de Construção dos Coeficientes com Aplicações**. Dissertação Pós-Graduação em Métodos Numéricos, Universidade Federal do Paraná, 2004.

REGRA, Carlos Manoel. **Teste de Mestrado em Estatística Computacional**. Universidade Aberta, 2010.

ROCHA, Ricardo Alexandre. **Algumas Evidências Computacionais da Infinitude dos Números Primos de Fibonacci**. Trabalho de Conclusão de Curso em Ciência da Computação, Universidade Federal do Rio Grande do Norte, 2008.

SOUSA, Tiago Alves. **Um Passeio na Sequência de Fibonacci**. Trabalho de Conclusão de Curso em Matemática, Universidade Estadual da Paraíba, 2012.

VIALI, Lorí. M. **Estatística Básica**. Departamento de Estatística, Instituto de Matemática da UFRGS (Universidade Federal do Rio Grande do Sul). Disponível em <http://www.pucrs.br/famat/viali>. Acesso em: 08 de ago. 2014.

VUOLO, José Henrique. **Fundamentos da Teoria de Erros**. 2.ed. Blucher, 1996.