Carlos Duarte do Nascimento

Uma Plataforma de Software para o Estudo Interativo de Métodos e Algoritmos Econométricos

Projeto e desenvolvimento de uma plataforma de software que permita o cadastro e a manipulação de procedimentos utilizados na econometria e implementações de metodos numéricos relacionados; Teste de um exemplo usando métodos numéricos em Econometria.

Orientador:
Professor Doutor Cicely Moitinho Amaral

Universidade de São Paulo

São Paulo

2008

Resumo

Propõe uma plataforma de *software* cujo objetivo é servir de ao ensino de tópicos de Econometria. Trabalha um problema de Econometria apresentando métodos numéricos para a sua resolução. Descreve a experiência de implementar a plataforma e o problema proposto, comentando os resultados.

Sumário

1 Introdução

1.1 Motivação

Dentre os objetivos de um curso de Econometria de nivel superior está a capacitação do aluno para compreender, construir e analisar modelos econométricos, utilizando-os como ferramentas para o planejamento estratético e a tomada de decisões relacionadas aos fenômenos estudados nestes modelos.

Neste sentido, é importante que o aluno tenha conhecimento tanto da teoria econômica envolvida quanto do instrumental matemático necessário para a modelagem dos fenômenos analisados. Esta meta só pode ser atingida se, aliada a uma sólida base teórica, o aluno puder interagir com estes modelos, realizando experiências e comparando resultados.

Tradicionalmente os cursos expoem o aluno a um pacote de software especializado, tal como o *EViews ou o* SPSS[1]. Esta abordagem tem como principal vantagem a capacitação do aluno em uma ferramenta que ele possivelmente irá usar no seu dia-a-dia.

Não é por acaso que tais ferramentas são adotadas universalmente por econometristas profissionais: elas automatizam muitos dos processos algébricos (regressões, aproximações numéricas) e estatísticos (testes de hipóteses, cálculo de estimadores qualitativos) envolvidos na modelagem econométrica. Entretanto, é exatamente este poder de simplificação que cria barreiras para o uso didático das

[1]Uma lista completa e atualizada de pacotes do gênero está disponível em [12]

mesmas: por mais importante que seja a visão geral do processo, a formação do aluno fica prejudicada se ele não é exposto aos detalhes computacionais do processo.

No extremo oposto, há professores que preferem o uso de um ambiente de programação tradicional, levando o aluno a construir suas próprias ferramentas e apresentar seus resultados na forma de "exercícios-programa". Esta abordagem permite maior aprofundamento, mas há a questão do tempo (a programação de um algoritmo de regressão simples, por exemplo, pode consumir boa parte do tempo de um curso básico de séries de tempo, por exemplo) e do interesse do aluno (que não pretende necessariamente se tornar um programador, mas apenas compreender a lógica do processo).

Estas duas abordagens nos levaram a experimentar uma alternativa intermediária: um sistema no qual um professor poderia mesclar textos teóricos com implementações de algoritmos para a resolução de problemas econométricos pontuais. O aluno poderia controlar seu nível de interação, podendo executar os algoritmos com dados pré-cadastrados, alterar estes dados e comparar os resultados, ou até mesmo criar seus próprios dados e sugerir mudanças nos algoritmos (caso tenha o conhecimento de programação de computadores para tanto).

Esta abordagem não é particularmente inédita: [.. falar um pouco do EconJava aqui? ...]

1.2 Proposta Funcional

O objetivo do sistema é facilitar o estudo de conceitos e técnicas de econometria. Suas principais áreas funcionais[2] seriam:

[2]Tais áreas estão propositalmente definidas de forma superficial, com o objetivo de não prjeudicar o caráter experimental do trabalho. Um produto de software mais objetivo deveria seguir a linha proposta por Jacobson[9].

1.2.1 Usuários

O sistema irá identificar os usuários no sistema, qualificando-os entre aqueles que irão introduzir o material a ser trabalhado (doravante denominados *professores*) e aqueles que irão trabalhar este material (que chamaremos de *alunos*).

1.2.2 Aulas

A introdução do material didático no sistema é feita pelo professor. Para fins de organização, cada tópico abordado será tratado como uma *aula*, dividida em um ou mais *passos*, que serão apresentados sequencialmente ao aluno quando ele selecionar uma determinada aula.

Cada passo da aula consiste em: , e consiste em:

- Parte teórica: Conteúdo textual/HTML simples, explicando o que será executado;
- Parte prática: Abaixo do texto teórico, o aluno encontrará campos onde poderá entrar com dados e executar um código previamente preparado pelo professor, obtendo resultados imediatamente. Esta execução pode ser feita repetidas vezes, variando os dados ou mesmo observando múltiplas iterações de um determinado algoritmo, até que o aluno decida avançar para o passo seguinte.

Desta forma, o professor pode programar possibilidades experimentais para o aluno, tornando o experimento tão amplo quanto deseje. É necessário codificar apenas a parte matemática/econométrica (muitas vezes já praticamente pronta na literatura) - sendo que a plataforma irá cuidar dos detalhes relacionados a compilação, fluxo e apresentação ao aluno.

Para o aluno tudo fica simplificado também - não há exposição ao código, apenas ao texto e à ferramenta para experimentar. Uma possibilidade é salvar os

dados de entrada/saída para uso futuro (ou para avaliação do professor). Também é interessante que o aluno possa carregar dados pré-existentes disponibilizados pelo professor (por exemplo, dados reais aplicados em um modelo).

1.3 Histórico de Trabalho

O trabalho se iniciou com a definição do escopo funcional da plataforma, através de pesquisa de trabalhos já existentes (incluindo o já mencionado trabalho do Prof. Cicely). Com base neste, os detalhes da arquitetura de software foram traçados.

Foi feita alguma pesquisa para implementar a compilação dinâmica em Java (de forma multiplataforma e baseada em API oficial), e para a escolha dos frameworks utilizados no código da plataforma, e deu-se início à implementação de fato.

Em paralelo, foi selecionado um problema de econometria com soluções numéricas conhecidas, e estas foram implementadas isoladamente da plataforma. Desta forma, passou-se a contar com uma base para testar a plataforma (convertendo este código para o cadastro de uma aula interativa).

Finalmente, este teste foi executado, e as conclusões foram registradas neste documento.

2 Mecanismos de Arquitetura

2.1 Introdução

Este capítulo tem por objetivo selecionar tecnologias (linguagens, ferramentas e frameworks) e metodologias (padrões de projeto) que otimizem uma plataforma de software para a implementação de métodos e algoritmos econométricos.

As escolhas definidas aqui foram norteadas por dois pré requisitos: que todas as metodologias tenham referencial acadêmico e implementações bem-sucedidas; e que todas as ferramentas utilizadas sejam gratuitas e de código livre, garantindo que a aplicação possa ser ampliada e modificada por qualquer entidade interessada, sem que hajam restrições de qualquer natureza.

2.2 Linguagem de Programação

É necessário fazer duas escolhas neste tópico: a da linguagem a ser utilizada para construir o aplicativo, e a da linguagem através da qual os algoritmos serão descritos para a mesma.

Para o aplicativo, é preciso ter uma linguagem que ofereça performance, escalabilidade, suporte ao desenvolvimento para a web e facilidade para trabalhar com bancos de dados relacionais (ferramenta indispensável para o volume de dados gerado pelo uso em larga escala, vide adiante). Também é importante que a linguagem não seja excessivamente obscura ou limitada a um nicho de mercado, já

que um dos objetivos é propor uma plataforma que possa ser ampliada e melhorada por quem tenha interesse.

Já o algoritmo exige uma linguagem que possa ser compilada ou interpretada dinamicamente, que tenha respaldo acadêmico (para aproveitar o código e, mais importante, o conhecimento pré-existente no corpo docente) e, preferencialmente, que já possua um ambiente de compilação/runtime livre (a implementação de tal ambiente não as complexidades inerentes a este tipo de implementação fogem ao escopo deste trabalho).

Com o uso dos mecanismos de arquitetura apropriados é possível trabalhar ambas as demandas através de uma linguagem única. Esta abordagem simplifica o projeto e reduz a barreira de entrada para novos desenvolvedores, e foi considerada antes de tentar trabalhar com uma linguagem de aplicação separada da linguagem de algoritmos.

Embora existam linguagens de domínio específico para algoritmos matemáticos, faz sentido considerar primeiro linguagens de uso geral (ainda no sentido de simplificar e popularizar a plataforma). Estudos como o de Prechelt[15] abordam aspectos como tempo de desenvolvimento e performance para o uso geral, mas é necessário levar em conta as necessidades particulares desta aplicação. Isto nos levou a considerar:

2.2.1 C/C++

A melhor opção em termos de performance. Por ter acesso aos mais diferentes tipos de bilbiotecas nos diversos sistemas operacionais existentes, também não apresentaria problemas para trabalhar com banco de dados ou com web. No entanto, a programação nesta dupla é bastante sujeita a erros, e muitas vezes é preciso escolher entre a flexibilidade do C++ e a performance do C.

Além disso, é necessário um esforço extra para garantir a compabilidade entre diferentes plataformas. Tais fatores elevariam o tempo do projeto e colocariam

uma barreira à entrada de novos desenvolvedores – isso sem falar que dificilmente a programação dos algoritmos seria feita de forma didaticamente viável nela.

2.2.2 Pascal

Tem a seu favor um excelente balanço entre performance e facilidade de programação (por ser mais fortemente tipada do que C e executar muitas das verificações de erros comuns em tempo de execução), além de possuir extensões de orientação a objeto e implementações livres (como o Free Pascal). No entanto as universidades já não tem mais incluído esta linguagem em seus currículos (Java é o substituto mais comum), o que se reflete em reduzida disponibilidade de programadores.

2.2.3 Fortran

Um dos pontos fortes é a vasta quantidade de algoritmos matemáticos e estatísticos disponíveis na literatura já codificados nesta linguagem. No entanto, a linguagem oferece poucas facilidades para a programação na web, e a mão-de-obra disponível é muito limitada ao meio científico/acadêmico.

2.2.4 Java (opção final)

O equilíbrio entre as demandas funcionais e não-funcionais levou à escolha da linguagem Java para a primeira implementação. Estudos como o de Bull, Smith, Pottage e Freeman[3] mostram que apresenta performance comparável a C, aliada a uma ampla gama de bibliotecas/frameworks que tornarão a implementação mais simples e expansível.

Uma vantagem adicional é a possibilidade que a linguagem oferece (a partir da Versão 6) de que o compilador seja invocado através de APIs de alto nível, permitindo a execução interativa dos algoritmos econométricos de forma independente

de sistema operacional ou plataforma.

Caso esta escolha não seja a mais apropriada no futuro, uma possibilidade que estará disponível são linguagens dinâmicas como Ruby, Python ou LISP, ou, alternativamente, como linguagens estáticas com suporte a reflexão, como C# ou Objective-C – observando-se o mapeamento dos mecanismos de arquitetura para recursos equivalentes em cada uma delas.

2.3 Ambiente de Desenvolvimento, Compilação e Publicação

É importante que o processo de compilação e publicação (*deploy*) da aplicação sejam completamente automatizados, de forma que qualquer pessoa possa facilmente descarregar o código existente de um repositório, testá-lo e implementar novas características.

A grande quantidade de frameworks envolvidos no processo gera uma dificuldade adicional: o gerenciamento de dependências. Tento isto em vista, usaremos o Maven[1] não apenas para executar a compilação e publicação, mas também para recuperar automaticamente todas as bibliotecas (.JAR) de frameworks utilizados a partir da Internet.

O código-fonte não irá exigir nenhum IDE[2] em particular. No entanto, o Eclipse (`http://eclipse.org`) será utilizado como ferramenta base, sempre tomando o cuidado de não tornar a ferramenta dependente dele.

[1] `http://maven.apache.org`

[2] *Integrated Development Environment* - Ambiente de Desenvolvimento Integrado

2.4 Arquitetura

Sob a perspectiva da arquitetura geral, o sistema pode ser visto como uma coleção interativa de CRUDs[3]. As exceções ficam por conta da execução interativa de algoritmos e da importação de dados), sendo, portanto, razoável trabalhar com a separação em três camadas descrita por Eckerson[5]:

- Uma camada de interface (front-end), utilizando MVC e outros princípios detalhados a seguir.
- Uma camada intermediária de "fachada", agrupando as operações de alto nível. Esta será implementada parcialmente através do mecanismo de MVC e (onde o reuso justificar) em classes de apoio através de objetos simples[4];
- Uma camada de operações (back-end) tais como: armazenamento de dados, execução de algoritmos e conversão de formatos, cuja implementação é detalhada adiante;

DESENHAR

Figura 1: Diagrama da Arquitetura

As melhores práticas de desenvolvimento de software muitas vezes demandam a implementação de diversos padrões de projeto[5]. Felizmente, a plataforma Java conta com diversos frameworks que implementam tais padrões de projeto, economizando esforço e tornando o código mais enxuto e focado no problema educacional.

O restante desta sessão é dedicado a descrever e justificar algumas destas práticas, definindo (quando aplicável) os frameworks selecionados para a implementação das mesmas.

[3] *Create-Read-Update-Delete*, acrônimo para módulos que efetuam estas quatro operações básicas sobre algum tipo de entidade -uma referência ao termo (contextualizada num ambiente OO) se encontra em Kilov[13])

[4] *POJO*, isto é, Plain Old Java Objects

[5] Dados empíricos mostram[2] a relação entre este tipo de padrão e a qualidade do software

2.4.1 Mapeamento Objeto-Relacional

Ao longo das últimas décadas, os sistemas gerenciadores de banco de dados relacionais (RDBMS) simplificaram o armazenamento de dados através da introdução da abordagem relacional de representação dos mesmos, implementada em pacotes de software de baixo custo, de forma isolada da aplicação principal, permitindo ao desenvolvedor concentrar-se no domínio específico do problema computacional a ser resolvido[6].

Além disso, a popularização dos RDBMS permitiu o uso de recursos computacionais relativamente limitados (tais como microcomputadores) para a execução de tarefas de manipulação de dados anteriormente restritas a sistemas de grande porte (e elevado custo de operação e manutenção), razão pela qual o uso de um RDBMS é indicado em qualquer sistema no qual a manipulação indireta dos dados não represente impacto na performance.

Tal característica, aliada à importância que os dados representam para as organizações (chegando, em muitos casos, a ser mais valiosos que os aplicativos ou os meios físicos nos quais eles são armazenados e processados), fez com que muitos dos sistemas desenvolvidos entre as décadas de 80 e 90 tivessem o modelo relacional de banco de dados como base do seu projeto – o software que manipularia estes dados era pensado de forma secundária, quase que conseqüência direta do desenho do banco.

Com a introdução das técnicas de desenvolvimento de software orientado a objeto – outro artefato que aumentou o nível de abstração com o qual os projetistas de software lidam com o domínio dos problemas (e, portanto, a sua produtividade) – surgiu uma nova abordagem: proponentes destas técnicas defendem que o sistema deve ser modelado sob o ponto de vista de suas classes – o armazenamento em meio não-volátil dos objetos destas classes (agora denominado *persistência*) passa a ser visto apenas como uma capacidade adicional das mesmas, e o banco de dados

[6]Para uma perspectiva histórica, consulte Codd[4]

torna-se um mero armazém de objetos[7].

A plataforma proposta neste trabalho segue esta nova abordagem, para a qual se coloca um problema: como representar a riqueza gramatical dos elementos da orientação a objeto (tais como herança, polimorfismo e navegabilidade) dentro do sistema de modelagem relacional dos RDBMS? Esta questão se divide em dois aspectos: o da busca da metodologia para mapear estas características e o da forma de implementá-la (evitando a redundância de código).

Este problema não é novo[8], tampouco exclusivo desta aplicação. A técnica para resolve-lo é denominada mapeamento objeto-relacional, e na plataforma Java existem diversos frameworks de código livre que a implementam[9]. Neste caso uma análise mais aprofundada pôde ser dispensada, visto que o Hibernate[10] é o padrão de facto adotado pela comunidade Java para sistemas com características de persistência convencionais, tais como este.

O Hibernate é um framework de código totalmente livre, que gera automaticamente e em tempo de execução as declarações SQL necessárias para persistir e recuperar objetos no RDBMS. O código SQL gerado é geralmente muito otimizado, e é possível customizar qualquer declaração que não seja aceitável. Além disso, a Versão 3 permite o uso de annotations, isto é, do desenho do mapeamento sobre o próprio código. Esta característica torna os arquivos de mapeamento (que usualmente demandam muito tempo e acrescentam um passo extra na compreensão do código) dispensáveis, o que, por si só, já justifica o seu uso.

Uma outra técnica para implementar o conceito de persistência é a chamada prevalência - nela, o estado dos objetos é armazenado em sistemas de arquivo tradicionais, num misto de serialização e de aualizações de estado (balanceando segurança dos dados e performance). Tal abordagem pode ser implementada usando

[7]Uma descrição isenta de plataforma deste tipo de mapeamento é dada por Ambler[1]

[8]Kilov novamente

[9]Uma lista ampla pode ser encontrada em `http://java-source.net/open-source/persistence`

[10]`http://www.hibernate.org`

um framework como Prevayler[11] para adicionar a capacidade de prevalência às classes de domínio. Embora esta técnica simplifique o deploy por dispensar um SGBD, a quantidade de memória RAM necessária para manter todo o grafo de objetos em memória tornaria inviável o uso da aplicação em larga escala.

2.4.2 Apresentação e MVC (Model / View / Controller)

Devido à sua natureza de interação com o usuário, a camada de apresentação é uma das mais sujeitas a alterações. Além disso, seu fluxo pode se tornar bastante complexo, o que favorece a duplicação de código desnecessária.

O padrão de projeto Model/View/Controller (MVC) tem se demonstrado útil na redução destes problemas. Nele, a camada de apresentação é segregada em dois tipos de componentes: view (composta pelas diversas interfaces do sistema, e desprovida de qualquer código que não esteja relacionado à interação com o usuário e à pré-validação dos dados introduzidos por ele) e model (código que responde a ações imperativas do usuário, tais como submeter um formulário de dados ou solicitar uma funcionalidade).

View e Model operam de forma totalmente independente: componentes de model respondem às solicitações utilizando as camadas inferiores e retornando algum tipo de status (ex.: "sucesso", "operação inválida", etc.), e componentes de view apresentam dados anexados a eles e retornam os dados novos ou alterações feitas pelos usuários.

A conexão entre eles é feita pelo controller: um componente que, para cada solicitação da view, dispara um ou mais componentes do model, e, conforme o resultado, apresenta uma nova view. Todo o fluxo é mantido neste componente (no código ou em um arquivo de configuração), desacoplando o código e oferecendo uma visão de alto nível que torna fácil identificar componentes reutilizáveis e/ou o impacto de quaisquer mudanças[12].

[11] `http://www.prevayler.org`

[12] Tygre descreve sua primeira implementação de MVC na interface gráfica original da Xerox,

Embora seja possível adotar a filosofia MVC através do desenvolvimento direto, muito trabalho pode ser popupado através do uso de um framework MVC que, dentre outras coisas, implemente um controller configurável e auxilie na passagem de dados entre model e view (tarefa que se torna complexa à medida em que se considera a generalidade do HTML no tocante a formulários de dados, e o desejo de usar técnicas como AJAX para aumentar a usabilidade da aplicação). De início, foram considerados os frameworks mais utilizados atualmente, a saber:

- Struts: Um dos frameworks mais tradicionais em Java, tem como vantagens uma biblioteca de apresentação bastante rica e evolução constante. Sua maior limitação é a grande quantidade de código/configuração necessários para definir o fluxo da aplicação;
- Spring MVC: É parte do framework Spring, o que facilitaria a sua integração. Mas também sofre do mal de exigir muita configuração, sem apresentar maiores atrativos que compensem o fato;

Tendo em vista que as alternativas padrão não atendem às necessidades do projeto, pesquisamos frameworks com menor base de usuários, tais como VRaptor, Jaffa e Stripes. Este último, por contar com excelente documentação e muitos exemplos na web, foi escolhido para a implementação.

Finalmente, é importante salientar que existe uma outra alternativa para otimizar o desenvolvimento da camada de apresentação: o uso de arquiteturas baseadas em componentes, tais como Wicket, WebWorks e JSF (os dois primeiros são frameworks, o último é uma especificação para este tipo de arquitetura, definida sob supervisão da empresa que desenvolve o Java e seguida por diversos frameworks), sobretudo pela agilidade que oferecem na prototipação e na criação de interfaces. No entanto, aplicações desenvolvidas sob este tipo de arquitetura não possuem

mas a consolidação do modelo em sistemas web é atribuída a Gamma / Helm / Johnson / Vlissides[8]

a “tolerância a mudanças” que o MVC proporciona (e estas mudanças seguramente ocorrerão à medida em que a plataforma for expandida), razão pela qual descartamos tal tipo de solução.

2.4.3 Inversão de Controle / Injeção de Dependências

Tendo em perspectiva que a aplicação resultante deste projeto será uma base para o desenvolvimento de outros sistemas, é importante que a mesma seja de fácil compreensão e manutenção.

Um dos grandes obstáculos para a manutenção de projetos de software é o acoplamento excessivo entre os seus diferentes módulos e camadas. Ainda que se use (e usamos) boas práticas de separação das mesmas (tais como a arquitetura Model-View-Controller e o modelo de três camadas), se estas camadas apresentarem excessiva dependência cruzada, pequenas alterações irão demandar grandes esforços de codificação e teste.

A inversão de controle[13] (IoC) é uma técnica de projeto que aborda o problema do acoplamento subvertendo a maneira tradicional com que um módulo do sistema solicita funcionalidade a outro módulo (daí o nome). O princípio fundamental é que um módulo que dependa de outro para executar a sua funcionalidade não o chama explicitamente – ao invés disso ele manifesta esta dependência de alguma forma, e o ambiente operacional cuida de oferecer o componente que melhor ofereça o tipo de serviço necessário.

Isso faz com que o módulo se concentre na sua própria funcionalidade – ao invés de misturar este código com o código que cuidará da iteração com a dependência. Há um ligeiro aumento na quantidade de código devido à necessidade de formalizar a dependência através de suas características (e não através da chamada direta do módulo que satisfaz a dependência) – o exemplo canônico de Fowler mostra essa

[13]Fowler[6] define a IoC como uma característica comum em frameworks, mas uma vez identificada, esta passou a ser usada como técnica de projeto

diferença. Mas isto é largamente compensado pelo desacoplamento obtido, e a clareza do código não é prejudicada.

Existem várias formas de implementar o princípio de IoC, sendo que a Injeção de Dependências é bastante popular por reduzir a quantidade de código envolvida no processo[14]. Nela, o módulo que oferece a funcionalidade apresenta uma interface não apenas para as tradicionais chamadas, mas também para as dependências delas (ex.: conexões de banco de dados, canais de saída, etc.). O módulo que solicita a funcionalidade o faz como na programação tradicional (chamando o método exposto), mas um framework media estas chamadas e fornece as dependências necessárias de forma apropriada. O Spring[15] tem se tornado um padrão no mundo Java para esta tarefa, a exemplo do que o Hibernate faz no mapeamento objeto-relacional.

[14] Fowler descreve esta implementação na forma de um padrão de projeto[7]
[15] `http://www.springsource.org/`

3 Implementação

[... modelo de dados e/ou de classes ...] [... casos de uso? telas? ...] [... limitações: ex.: não usamos spring, não salvamos/recuperamos dados ...]

4 Métodos e Modelos

Seria difícil avaliar a eficácia da plataforma de software sem a implementação de um exemplo prático, o que nos leva a trabalhar nela um modelo econométrico de interesse e um instrumental matemático associado. Este capítulo aborda os detalhes matemáticos destas escolhas.

[Não sei se vai ser exatamente séries de tempo, mas o trecho abaixo é só pra dar uma idéia do "clima"do texto]

4.1 Modelo Econométrico: Séries de Tempo

Séries de Tempo consistem em conjuntos de observações discretas de um fenômeno feitas ao longo de um período de tempo. Elas diferem de outros conjuntos de dados por apresentar dependências entre observações vizinhas, relações sazonais e outras caraterísticas únicas que podem ser aproveitadas na sua análise.

Tal análise permite a criação de modelos que permitem a elaboração de previsões, que são do maior interesse para pesquisadores e para o público em geral. Exemplos econométricos característicos incluem a previsão do comportamento de ações em bolsas de valores[16], ou a determinação do impacto de fatores qualitativos em vendas de produtos.

[... completar com a descrição matemática de uma série de tempo? ...]

4.2 Métodos Numéricos

Judge, Griffiths, Hill e Lütkepohl[11] apresentam de forma sucinta e didática o uso de métodos numéricos de otimização para a estimação de parâmetros em modelos estatísticos no geral (tendo em vista os modelos econométricos apresentados ao longo do livro).

Para o nosso exemplo, a parte de interesse são os métodos de gradiente, aplicados a problemas sem restrições. A proposta dos autores (pp. 951 – 954) é trabalhar métodos iterativos, nos quais se busca uma seqüência de parâmetros $\theta_1, \theta_2, ..., \theta_n$ que minimizam uma função-objetivo conveniente $H(\theta)$.

Isto significa que, para cada iteração n, temos uma matriz de direção P_n e tamanho de passo t_n, que multiplicados pelo gradiente da função H (γ_n) permitem determinar o parâmetro para o passo seguinte, isto é:

$$\theta_{n+1} = \theta_n - t_n P_n \gamma_n$$

A abordagem adotada é a de usar métodos de gradiente para que [... continuar ...]

5 Experimentos e Resultados

6 Conclusões

APÊNDICE A – Apêndice I - Listagens de Software

A.1 Configuração do Ambiente de Trabalho

Os seguintes passos permitem a configuração de um ambiente para trabalhar com o código desenvolvido:

- Instalar o Java Development Kit (JDK) da Sun (`http://java.sun.org/jdk`) ou equivalente, versão 1.6 ou superior;
- Instalar o Eclipse (`http://www.eclipse.org`), versão 3.4 ou superior;
- Instalar o Maven (ex.: sudo apt-get install maven2);
- Adicionar ao Eclipse o suporte a Maven (`http://m2eclipse.codehaus.org/`);
- Adicionar ao Eclipse o componente Web Tools Platform (WTP, `http://www.eclipse.org/webtools/`);
- Baixar o código fonte e abrir;
- Instalar o Tomcat 6.0 ou superior (`http://tomcat.apache.org`);

Referências

[1] Scott W. Ambler. Mapping objects to relational databases: O/r mapping in detail. http://www.agiledata.org/essays/mappingObjects.html.

[2] Kent Beck, Ron Crocker, Gerard Meszaros, John Vlissides, James O. Coplien, Lutz Dominick, and Francis Paulisch. Industrial experience in design patterns. In *Proceedings of the 18th International Conference on Software Engineering: March 25-29, 1996 Berlin, Germany*. IEEE Computer Society, 1996.

[3] J. M. Bull, L. A. Smith, L. Pottage, and R. Freeman. Benchmarking java against c and fortran for scientific applications. In *In ACM 2001 Java Grande/ISCOPE Conf*, pages 97–105. ACM Press, 2001.

[4] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, 1970.

[5] Wayne W. Eckerson. Three tier client/server architecture: Achieving scalability, performance, and efficiency in client server applications. *Open Information Systems*, 10(1), 1995.

[6] Martin Fowler. Inversion of control. http://martinfowler.com/bliki/InversionOfControl.html.

[7] Martin Fowler. Inversion of control containers and the dependency injection pattern. http://martinfowler.com/articles/injection.html.

[8] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. Addison-Wesley Publishing Company, New York, NY, 1995.

[9] Ivar Jacobson. *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2004.

[10] David Janzen and Hossein Saiedian. Test-driven development: Concepts, taxonomy, and future direction. *Computer*, 38(9):43–50, 2005.

[11] George G. Judge, William E. Griffiths, R. Carter Hilla, Helmut Lütkepohl, and Tsoung-Chao Lee. *The Theory and Practice of Econometrics (Wiley Series in Probability and Statistics)*. Wiley, 1985.

[12] John Kane. Econometric software. `http://www.oswego.edu/~economic/econsoftware.htm`.

[13] Haim Kilov. From semantic to object-oriented data modeling. In Ng et al. [14], pages 385–393.

[14] Peter A. Ng, C. V. Ramamoorthy, Laurence C. Seifert, and Raymond T. Yeh, editors. *Proceedings of the First International Conference on Systems Integration, Morristown, NJ, USA, April 1990*. IEEE Computer Society, 1990.

[15] Lutz Prechelt. An empirical comparison of seven programming languages. *Computer*, 33(10):23–29, 2000.

[16] Wikipedia. Time series. `http://en.wikipedia.org/w/index.php?title=Time_series&oldid=232223965`.