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 aplicar a ela o problema proposto, comentando os resultados.

Sumário

1 Introdução

1.1 Econometria: Teoria e Prática

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 e estatístico 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 é mantida por Kane[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).

1.2 Um Problema Econométrico

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

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 fins ilustrativos, eles resgatam um problema de [FALAR COM PROF. CICELY, é referenciado na pág. 956, mas não tenho a nota bibliográfica], que considera o modelo estatístico não-linear:

$$y_t = \theta_1^* + \theta_2^* x_{t2} + (\theta_2^*)^2 x_{t3} + e_t, t = 1, 2, ..., 20$$

$$y = f(\theta^*) + e$$

onde $y = (y_1, y_2, ..., y_{20})'$, $\theta^* = (\theta_1^*, \theta_2^*)'$ é o vetor de parâmetros, $e = (e_1, e_2, ..., e_{20})'$ e

$$f(\theta) = \begin{pmatrix} \theta 1 + \theta_2 x_{12} + \theta_2^2 x_{13} \\ \theta 1 + \theta_2 x_{22} + \theta_2^2 x_{23} \\ . \\ . \\ . \\ \theta 1 + \theta_2 x_{20,2} + \theta_2^2 x_{20,3} \end{pmatrix}$$

O objetivo é encontrar um estimador de mínimos quadrados para θ^*, e para tanto devemos minimizar a função:

$$H(\theta) = [y - f(\theta)]'[y - f(\theta)] \tag{1.1}$$

Este é um problema econométrico típico, já adaptado a um modelo matemático, que se presta bem a ilustrar a aplicação de uma plataforma como a proposta, razão pela qual nos pautamos sobre ele. Vamos utilizar os dados fornecidos pelos autores, apresentados na tabela 1, e estimar os parâmetros.

Os valores de x_{t2} e x_{t3} são números pseudo-aleatórios, gerados a partir de uma distribuição uniforme no intervalo unitário, e os y_t foram gerados tomando o modelo $\theta_1^* + \theta_2^* x_{t2} + \theta_2^* x_{t3}$, adicionando a ele um número pseudo-aleatório e considerando $\theta_1^* = \theta_2^* = 0$, isto é, adicionando um erro aleatório a $1 + x_{t2} + x_{t3}$.

t	y_t	x_{t1}	x_{t2}	x_{t3}
1	4.284	1,000	0.286	0.645
2	4.149	1,000	0.973	0.585
3	3.877	1,000	0.384	0.310
4	0.533	1,000	0.276	0.058
5	2.211	1,000	0.973	0.455
6	2.389	1,000	0.543	0.779
7	2.145	1,000	0.957	0.259
8	3.231	1,000	0.948	0.202
9	1.998	1,000	0.543	0.028
10	1.379	1,000	0.797	0.099
11	2.106	1,000	0.936	0.142
12	1.428	1,000	0.889	0.296
13	1.011	1,000	0.006	0.175
14	2.179	1,000	0.828	0.180
15	2.858	1,000	0.399	0.842
16	1.388	1,000	0.617	0.039
17	1.651	1,000	0.939	0.103
18	1.593	1,000	0.784	0.620
19	1.046	1,000	0.072	0.158
20	2.152	1,000	0.889	0.704

Tabela 1: Dados usados no exemplo

2 A Plataforma

2.1 Proposta Funcional

O objetivo do sistema é facilitar o estudo de conceitos e técnicas de econometria. Suas principais áreas funcionais[1] seriam:

2.1.1 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*).

2.1.2 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:

- Parte teórica: Conteúdo textual/HTML simples, explicando o que será exe-

[1] 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].

cutado;

- 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).

2.2 Modelagem

2.2.1 Diagrama de Classes de Entidade

As principais entidades são modeladas de acordo com o diagrama na Figura 1.

Passo é a entidade mais importante do sistema, representando cada tela com a qual o usuário interage, incluindo o código-fonte (algoritmo) que será experimentado pelo aluno. Uma **Aula** agrupa um conjunto de passos, permitindo a organização do conteúdo didático.

Cada variável que o professor determina que o aluno poderá inserir ou recupe-

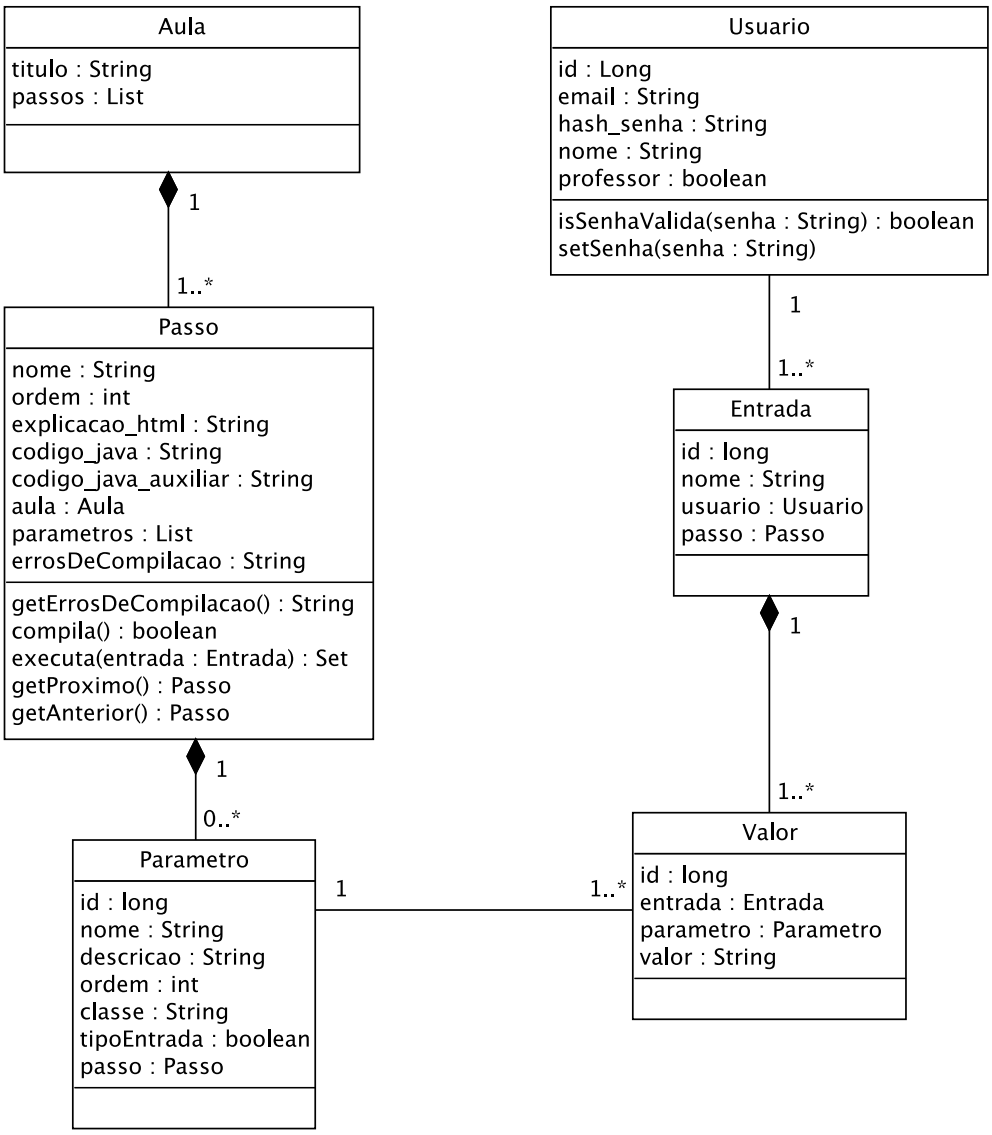

Figura 1: **Classes de Entidade do Sistema**

rar de um passo é um **Parametro**. Os dados efetivamente entrados pelo aluno são representados por objetos **Valor** e guardados coletivamente na forma de **Entrada**.

2.2.2 Casos de Uso

No geral, as iterações mais importantes do sistema são os casos de uso apresentados na Figura 2, cujos passos são descritos[2] em seguida.

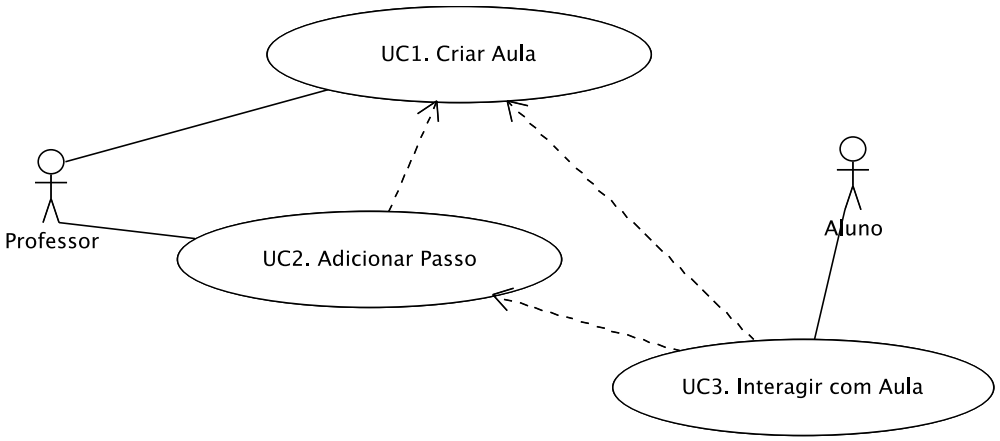

Figura 2: **Casos de Uso do Sistema**

2.2.2.1 UC1. Criar Aula

Após dividir o conteúdo didático em aula, o professor insere cada uma delas no sistema, atribuindo um nome e uma descrição breve.

2.2.2.2 UC2. Adicionar Passo

Cada aula é dividida pelo professor em um ou mais passos, que serão executados sequencialmente pelo aluno. Para cada passo adicionado, o professor dá

[2]Tradicionalmente, Casos de Uso são descritos através de vários campos-padrão (nome, atores, pré-requisitos, passos, etc.), seguindo as linhas gerais delineadas por Jacobson[9]. Tendo em vista o foco do trabalho, optamos por uma descrição mais sucinta, focada nos passos e em linguagem natural

um nome, cadastra um texto explicativo e um algoritmo que será executado como exercício ao final da leitura do texto, com parâmetros de entrada e saída também definidos pelo professor.

2.2.2.3 UC3. Interagir com Aula

Uma vez tendo o material preparado, o aluno pode entrar no sistema e visualizar a lista de todas as aulas. Ao selecionar uma delas, o primeiro passo tem seu texto explicativo exibido, e, ao final, o aluno pode entrar com os dados e verificar os resultados, como programado pelo professor.

Um projeto de software mais elaborado poderia documentar Para maior clareza, seguem algumas das páginas [3]

[3]Um projeto de software mais elaborado poderia documentar estas interações na forma de Casos de Uso UML[]

3 Arquitetura de Software

Uma vez definida a funcionalidade desejada, faz-se necessário selecionar tecnologias (linguagens, ferramentas e frameworks) e metodologias (padrões de projeto) que facilitem a criação da plataforma de software desejada de forma eficiente, e, acima de tudo, de forma expansível para acomodar necessidades futuras.

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.

3.1 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:

3.1.1 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.

3.1.2 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.

3.1.3 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.

3.1.4 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.

3.2 Ambiente de Desenvolvimento

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.

3.3 Compilação e Carregamento Dinâmico de Classes

Uma vez decidido que Java seria tanto a linguagem de desenvolvimento quanto a linguagem usada para definir os exercícios que seriam aplicados em cada passo

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

[2] *Integrated Development Environment* - Ambiente de Desenvolvimento Integrado - conjunto que agrega editor de textos, compilador e ferramenta de depuração em um único pacote, agilizando o desenvolvimento de software

de cada aula, foi preciso utilizar o recurso de compilação dinâmica, disponível a partir da versão 1.6 da linguagem, bem como o carregamento dinâmico de classes[3].

A abordagem para o uso do primeiro recurso foi a criação de uma nova classe para cada passo inserido no sistema, gerada a partir do identificador do passo relacionado a ela. Esta classe receberia um mapa nome-valor (*java.util.Map*) com os parâmetros de entrada e seus valores, e devolveria um outro mapa no mesmo formato com os parâmetros de saída (e eventuais parâmetros de entrada modificados - este último detalhe é importante para a execução de métodos iterativos como o que usaremos como exemplo).

Uma vez montada esta classe, a mesma é compilada dinamicamente sempre que necessário. Para executá-la, em princípio poderia ser feita uma chamada direta, mas isso gera um problema: a cada mudança é preciso descarregar a classe da memória e recarregá-la, e não é possível executar esta operação com classes carregadas automaticamente pelo *classloader* padrão do Java. A solução é instanciar um novo classloader a cada execução, colocando as classes compiladas num diretório visível apenas para ele.

Desta forma, quando não há mais referência ao classloader, ele se torna elegível para descarga na coleta de lixo seguinte - e o mesmo ocorre com as classes carregadas por ele. Ainda é possível solicitar à JVM a coleta de lixo (via *System.gc()*) - isto não garante a execução do procedimento, mas na maioria das implementações evita que o mesmo aconteça apenas após muitas alterações, gerando um benefício de performance. Em qualquer caso, o sistema reconhece as mudanças já na execução seguinte do passo da aula - que é o efeito desejado.

3.4 Testes Automatizados

É muito importante que o sistema mantenha os resultados consistentes, mesmo com a implementação de novas funcionalidades. Para tanto, a criação de testes

[3] A parte relevante do código-fonte para cada um destes recursos está disponível na Seção A.1

automáticos (unitários e funcionais) durante o processo de desenvolvimento (e não como um detalhe adicional) é desejável - de fato, autores como Janzen e Saiedian[10] defendem a idéia de que o desenvolvimento como um todo deve ser pensado com foco nos testes, alegando que esta abordagem (*Test-Driven Development*) teria impacto positivo na qualidade do código gerado e do produto final.

Sistema Java costumam usar o framework JUnit[4] para implementação de testes unitários automáticos, devido à sua natureza não-intrusiva (mantendo o código de testes separado do restante) e seu amplo leque de capacidades embutidas, mantendo o código de testes bastante sucinto. Mesmo testes funcionais automáticos podem ser feitos através dele (uma estratégia é expandi-lo através da extensão JFunc[5], desenhada para este fim).

3.5 Padrões de Projeto

Sob a perspectiva da arquitetura geral, o sistema pode ser visto como uma coleção interativa de CRUDs[6]. 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], ilustrada na Figura 3 e detalhada da seguinte forma:

- Uma camada de apresentação, que irá isolar a interface da aplicação das camadas inferiores, utilizando MVC e outros princípios detalhados a seguir.
- Uma camada intermediária de lógica de negócio, que irá interfacear com as *actions* MVC e delegar as operações às classes de apoio (*helpers*) através de objetos simples[7], a bibliotecas matemáticas como a Jama e às classes de

[4] `http://junit.org`

[5] `http://jfunc.sourceforge.net`

[6] *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]

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

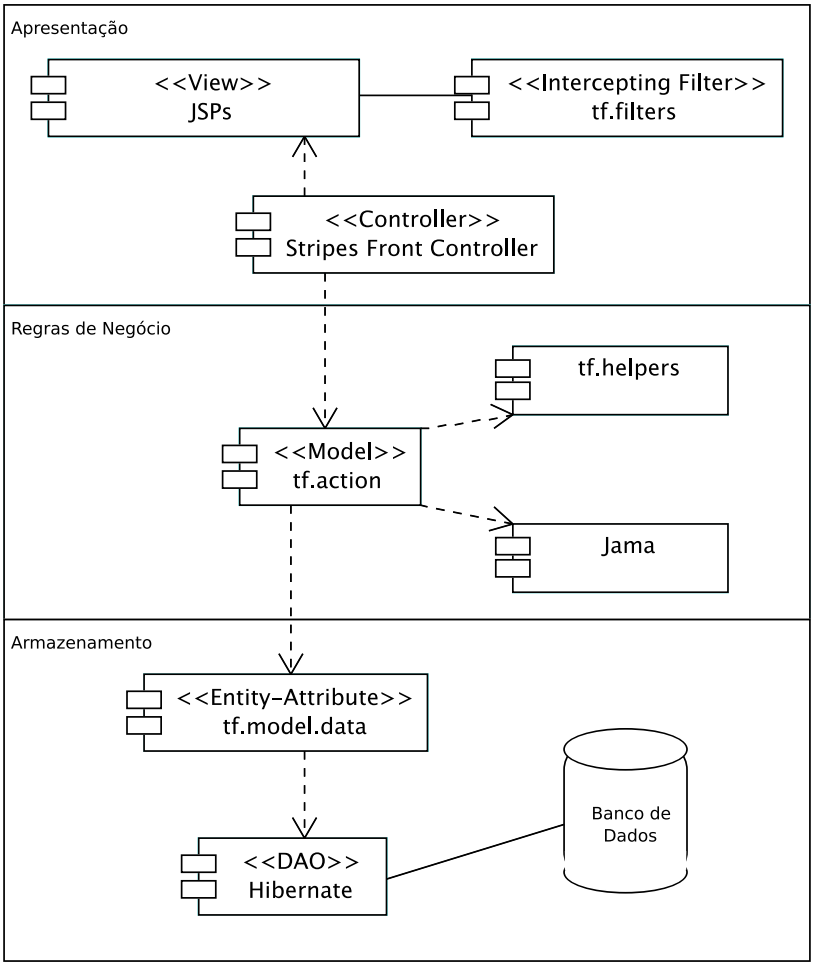

Figura 3: **Diagrama de Componentes da aplicação, separado por camadas**

entidade.

- Uma camada de armazenamento (*back-end*), que, entre outras coisas, garante a independência da aplicação de detalhes específicos de cada banco de dados e abriga as classes de entidade (tornando sua interação com o banco natural).

As melhores práticas de desenvolvimento de software muitas vezes demandam a implementação de diversos padrões de projeto[8]. 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/econométrico.

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

3.5.1 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[9].

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.

[8]Dados empíricos mostram[2] a relação entre este tipo de padrão e a qualidade do software
[9]Para uma perspectiva histórica, consulte Codd[4]

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 consequência 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 torna-se um mero armazém de objetos[10].

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[11], 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[12]. Neste caso uma análise mais aprofundada pôde ser dispensada, visto que o Hibernate[13] é o padrão de facto adotado pela comunidade Java para sistemas com características

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

[11] Kilov novamente

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

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

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 um framework como Prevayler[14] 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.

3.5.2 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

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

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[15].

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.

[15]Tygre descreve sua primeira implementação de MVC na interface gráfica original da Xerox, mas a consolidação do modelo em sistemas web é atribuída a Gamma / Helm / Johnson / Vlissides[8]

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 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.

3.5.3 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[16] (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 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[17]. 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[18] tem se tornado um padrão no mundo Java para esta tarefa, a exemplo do que o Hibernate faz no mapeamento objeto-relacional. No entanto, isso deve ser balanceado com a real neceissade do sistema, evitando torná-lo mais complexo que o necessário.

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

[17] Fowler descreve esta implementação na forma de um padrão de projeto[7]

[18] `http://www.springsource.org/`

4 Resolução do Problema

4.1 Métodos de Gradiente

O problema descrito anteriormente pode ser resolvido através de métodos de aplicados a problemas sem restrições. A proposta dos autores (pp. 951 – 954) é trabalhar métodos iterativos, nos quais se busca uma sequência de parâmetros $\theta_1, \theta_2, ..., \theta_n$ que minimizam a função-objetivo $H(\theta)$ (cuja definição está em 1.1).

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:

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

O método torna-se completo quando determinamos uma condição de parada. Idealmente o faríamos quando o valor da função objetivo não puder mais ser reduzido. Para fins práticos, paramos quando a redução passar a ser insignificante, por exemplo, se, para um $\epsilon > 0$ suficientemente pequeno e um l inteiro e positivo tivermos uma das seguintes condições:

1. $(\theta_{n+l} - \theta_n)'(\theta_{n+l} - \theta_n) < \epsilon$
2. $H(\theta_n) - H(\theta_{n+l}) < \epsilon$
3. $[\frac{\partial H}{\partial \theta}|_{\theta_n}]'[\frac{\partial H}{\partial \theta}|_{\theta_n}] < \epsilon$

Além disso, devemos impor limites no número de passos e/ou tempo de compilação, e tentar novos valores iniciais quando este tipo de condição ocorrer.

A diferença principal entre os vários métodos de gradiente se dá no critério para a escolha da direção do passo, isto é, da matriz de direções P_n.

A seguir apresentamos alguns métodos detalhados pelos autores, acompanhados do código Java correspondente. Estes trechos de código podem ser usados diretamente na construção de uma aula sobre o assunto na plataforma.

4.1.1 Newton-Raphson

Este algoritmo consiste em usar o inverso da matriz Hessiana para especificar a direção do passo em cada iteração, ajustando-o pelo gradiente, isto é:

$$\theta_{n+1} = \theta_n - \mathcal{H}_n^{-1}\gamma_n$$

Aqui entendemos $\mathcal{H}_n$ como o Hessiano de $H(\theta)$ aplicado em θ_n, isto é, a matriz definida por:

$$\mathcal{H}_n = \left[\frac{\partial^2 H}{\partial\theta\partial\theta'}\bigg|_{\theta_n} \right]^{-1}$$

O código abaixo calcula, para um par *theta1 / theta2*, a matriz com o passo, isto é, os valores que, subtraídos deste par, resultarão no par correspondente ao passo seguinte[1]:

```
double[][] hess = hess(theta1, theta2, y, x2, x3);
Matrix invHess = new Matrix(hess).inverse();
Matrix grad = new Matrix(gradH(theta1, theta2, y, x2, x3), 2);
Matrix passo = invHess.times(grad);
```

É interessante notar que este código utiliza a classe *Matrix* pertencente à ao

[1]O código da função hess, que calcula a matriz Hessiana para a função objetivo no ponto, encontra-se no Apêndice A - bem como o de outras funções de apoio como esta.

pacote JAMA[2]. Este pacote possui como principal vantagem a escrita "natural"de operações encadeadas - por exemplo, dada uma matriz M, se quiséssemos calcular $(M^{-1})'$, isto é, a transposta da inversa de M, o código seria diretamente: *M.inverse().transpose()*.

Os resultados obtidos através da aplicação sucessiva deste passo (conforme descrito no Capítulo 1 coincidem com os apresentados pelos autores e reproduzidos aqui na Tabela 2.

Esta tabela apresenta três casos de execução: o primeiro iniciando no ponto $\theta = (3, 2)$, o segundo no ponto $\theta = (0, 2)$ e o terceiro no ponto $\theta = (1.5, 0.5)$. Nos dois primeiros casos, a função $H(\theta)$ converge para o mínimo local (16.0817) em $\theta = (0.864787, 1.235748)$, e no último para outro mínimo local (20.9805), localizado em $\theta(2.354471, -0, 319186)$[3].

O exemplo mostra que não existe garantia de convergência para o mínimo global em algoritmos de otimização como estes. De qualquer forma, é possível observar a velocidade de convergência e comparar o algoritmo com alternativas como as que seguem adiante.

4.1.2 Gauss-Newton

Este método utiliza uma particularidade da forma da função objetivo, calculando o passo como:

$$\theta_{n+1} = \theta_n + [Z(\theta_n)'Z(\theta_n)]^{-1} Z(\theta_n)'[y - f(\theta_n)]$$

onde $Z(\theta) = [\partial f / \partial \theta'|_\theta]$. O texto supra mencionado mostra como este algoritmo pode ser visto como uma sequência de regressões lineares (sendo θ_{n+1} o estimador

[2]Sua especificação e uma implementação de referência podem ser encontradas em `http://math.nist.gov/javanumerics/jama/`. A plataforma implementada neste trabalho disponibiliza o JAMA automaticamente para a escrita de algoritmos.

[3]Estes valores são aproximações numéricas

n	$\theta_{n,1}$	$\theta_{n,2}$	$H(\theta_n)$
1	3.000000	2.000000	264.3918
2	-0.084033	1.811210	20.6328
3	0.625029	1.423940	16.5105
4	0.817259	1.272776	16.0961
5	0.862590	1.237516	16.0818
6	0.864782	1.235753	16.0817
7	0.864787	1.235748	16.0817
8	0.864787	1.235748	16.0817
1	0.000000	2.000000	29.2758
2	0.334936	1.600435	17.7382
3	0.735040	1.336953	16.1955
4	0.849677	1.247743	16.0832
5	0.864541	1.235946	16.0817
6	0.864787	1.235749	16.0817
7	0.864787	1.235748	16.0817
8	0.864787	1.235748	16.0817
1	1.500000	0.500000	20.2951
2	2.256853	0.007135	20.7735
3	2.467047	-0.436460	21.0312
4	2.316982	-0.202435	20.9467
5	2.359743	-0.320579	20.9809
6	2.354457	-0.319153	20.9805
7	2.354471	-0.319186	20.9805
8	2.354471	-0.319186	20.9805

Tabela 2: Algumas iterações do método de Newton-Rhapson.

de mínimos quadrados para o modelo $\bar{y}(\theta_n) = Z(\theta_n)\theta + e$). Novamente representando o ponto atual através do par (*theta1*,*theta2)*), uma possível implementação para o cálculo do passo seria:

```
Matrix Z = Z(theta1, theta2, x2, x3);
Matrix Zt = Z.transpose();
Matrix f = f(theta1, theta2, x2, x3);
Matrix passo = Zt.times(Z).inverse().times(Zt).times(y.minus(f)).times(-1);
```

Neste código, *Zt* representa a transposta de $Z(\theta)$, e f o valor de $f(\theta)$ para o conjunto de parâmetros no passo atual (θ_n).

Novamente, encontramos os mesmos resultados obtidos por Judge et al, conforme a Tabela 3. O experimento com o par $(3, 2)$ permite comparar a convergência com o o algoritmo de Newton-Rhapson, e observa-se convergência mais rápida para o ponto $(0.864787, 1.235749)$.

O terceiro experimento também usa os mesmos dados do método anterior, mas observa-se que, partindo do mesmo ponto $(1.5, 0.5)$, este método converge para o ponto de mínimo local do primeiro experimento - diferindo do resultado obtido com Newton-Rhapson para este ponto inicial. Novamente evidencia-se a inexistência de controle sobre qual ponto de mínimo local será atingido.

4.1.3 Marquardt-Levenberg

Para tornar o exemplo completo, apresentamos um algoritmo gerado através da modificação de algoritmos anteriores. A motivação para esta modificação está em garantir que a matriz Z seja não-singular (Z singular seria algebricamente não-inversível - como lidamos com aproximações, isso se traduz em perturbações e consequente perda de robustez do algoritmo).

Para tanto, precisaríamos garantir que P_n seja positiva definida e, neste sentido, é possível usar o fato de que

n	$\theta_{n,1}$	$\theta_{n,2}$	$H(\theta_n)$
1	3.000000	2.000000	264.3918
2	-0.084033	1.811210	20.6328
3	0.625029	1.423940	16.5105
4	0.817259	1.272776	16.0961
5	0.862590	1.237516	16.0818
6	0.864782	1.235753	16.0817
7	0.864787	1.235748	16.0817
8	0.864787	1.235748	16.0817
1	0.000000	2.000000	29.2758
2	0.334936	1.600435	17.7382
3	0.735040	1.336953	16.1955
4	0.849677	1.247743	16.0832
5	0.864541	1.235946	16.0817
6	0.864787	1.235749	16.0817
7	0.864787	1.235748	16.0817
8	0.864787	1.235748	16.0817
1	1.500000	0.500000	20.2951
2	2.256853	0.007135	20.7735
3	2.467047	-0.436460	21.0312
4	2.316982	-0.202435	20.9467
5	2.359743	-0.320579	20.9809
6	2.354457	-0.319153	20.9805
7	2.354471	-0.319186	20.9805
8	2.354471	-0.319186	20.9805

Tabela 3: Algumas iterações do método de Gauss-Newtion

$$P_n + \lambda_n \bar{P}_n$$

é sempre positiva definida se $\bar{P}_n$ também o for e o escalar for suficientemente grande Uma idéia sugerida pelos autores é aplicar esta idéia no método de Gauss, modificando $Z(\theta_n)'Z(\theta)$ (e não sua inversa), i.e.:

$$P_n = [Z(\theta_n)'Z(\theta_n) + \lambda_n \bar{P}_n]^{-1}$$

Valores reduzidos de λ_n toranm o método equivalente ao método de Gauss, e valores maiores aproximam a direção daquela de declive máximo. É possível trabalhar com o escalar constante, ou começar com ele pequeno e diminuir a cada iteração, a menos que isso leve a um tamanho de passo inaceitável.

De forma similar, modificar o Hessiano da função objetivo e usar, como matriz de direção:

$$P_n = [\mathcal{H}_n + \lambda_n I_K]^{-1}$$

Novamente, o código-fonte para o cálculo da matriz:

```
Matrix f = f(theta1, theta2, x2, x3);
Matrix Z = Z(theta1, theta2, x2, x3);
Matrix Zt = Z.transpose();
Matrix lambda_I = Matrix.identity(2, 2).times(0.7);
Matrix passo = Zt.times(Z).minus(lambda_I).inverse().times(Zt.times(y.minus(f))).times(-1);
```

Por simplicidade, trabalhamos o escalar constante (0.7), e a variável *lambda_I* é a identidade multiplicada por ele, sendo as outras matrizes definidas de forma idêntica ao caso anterior. A velocidade de convergência é comparável à dos outros métodos (e pode ser melhorada trabalhando o fator λ_n conforme descrito acima), mas a grande vantagem é a robustez decorrente da modificação.

5 *Conclusões e Sugestões Para Continuidade*

O objetivo do trabalho era demonstrar a viabilidade da construção de uma plataforma para facilitar a criação de aulas experimentais, testando sua viabilidade em um exemplo de sala de aula. Neste sentido, verificou-se que, embora os features básicos tenham sido implementados em tempo razoável e o sistema tenha se demonstrado eficiente para a composição de blocos didáticos específicos, é preciso um tratamento muito mais abrangente para se chegar a uma ferramenta de abrangência mais universal e eficácia mensurável.

Contudo, o resultado obtido (tanto em termos de projeto quanto de implementação) pode ser uma base para a construção desta ferramenta. Um aspecto que deve ser trabalhado de forma mais profunda do que o escopo do trabalho permitiu é o didático - um estudo um pouco mais focado nas necessidades do aluno durante o aprendizado da Econometria poderia dar ao sistema uma orientação que lhe permitisse atender melhor a estas demandas.

De qualquer forma, mesmo a visão centrada na exibição de conteúdo e execução de algoritmos sugere a implementação de um conjunto de funcionalidades que, dados os limites naturais de um trabalho de graduação, não pôde ser implementado em sua totalidade.

Assim, o sistema seria enriquecido se fossem implementados itens funcionais tais como:

- Cadastro de usuários (possivelmente vinculado a algum sistema de matrícula);
- Possibilidade de salvar e recuperar dados;
- Permitir ao código decidir o próximo passo a ser executado;
- Uma interface mais amigável para o professor (em particular na visualização do texto das aulas e da depuração do código dos algoritmos);
- Possibilidade de usar outros sistemas de codificação, como LaTeXou MathML na composição do texto das aulas;
- Conversão semi-automática de algoritmos em outras linguagens (ex.: FORTRAN).

É importante notar que o sistema comporta a inclusão de todas estas características, sem prejuízo das já existentes, e sem a necessidade de refatoração em larga escala.

Sob o ponto de vista técnico, algumas das possíveis melhorias incluem:

- Utilização de um sistema mais eficiente de Inversão de Controle: a IoC foi aplicada de forma bastante moderada, sem o uso de um framework específico. Para o tamanho atual do sistema isto não representou maiores problemas (mesmo com as constantes mudanças feitas ao longo do projeto), mas uma revisão neste aspecto seria interessante caso o mesmo venha a crescer no futuro;
- Criação de uma rotina formal de testes unitários - o sistema atual tem apenas testes ad-hoc no próprio código;
- Medição e melhorias na performance de pontos específicos - em particular no sistema de compliação dinâmica, que recompila as classes a cada execução. Uma alternativa é atrelar um único classloader para cada passo, e mantê-los

no escopo da aplicação. Esta refatoração pode ser feita diretamente na classe Passo, sem prejuízo para o restante do código.

APÊNDICE A – Complementos Técnicos

A.1 Listagens de Software

O código-fonte completo da implementação contém cerca de 5000 linhas de código[1], tornando sua reprodução aqui pouco produtiva. Optou-se por reproduzir os trechos mais relevantes, mantendo o código-completo disponível para download na Internet[2].

A.1.1 Compilação Dinâmica de Classes

Como cada algoritmo a ser executado estava atrelado a um passo de uma aula, optou-se por vincular a capacidade de compilação à classe *Passo*. A estratégia é discutida na Seção 3.3, e o código segue abaixo.

```
public boolean compila() {

  // Monta o código-fonte
  StringBuilder fonte = new StringBuilder();
  fonte.append("package " + this.getNomePackage() + ";\n");
  fonte.append("import java.util.*;\n");
  fonte.append("import Jama.*;\n");
  fonte.append("public class " + this.getNomeClasse() + " extends tf.codigodinamico.Base {\n");
```

[1] Excluindo comentários, segundo relatório obtido através do software *cloc*, em `http://cloc.sourceforge.net/`

[2] O código completo pode ser baixado no endereço `http://chester.blog.br/tf`

```
fonte.append(" public Map<String, Object> executa(Map<String, Object> __entrada) {\n");
// Recupera os parâmetros de entrada do mapa, com statements no formato:
// Classe nome = (Classe)__entrada.get("nome");
// e declara os de saída como simplesmente
// Classe nome = null;
if (this.getParametros() != null)
  for (Parametro p : this.getParametros()) {
    fonte.append("  ").append(p.getClasse()).append(' ').append(
        p.getNome()).append(" = ");
    if (p.isTipoEntrada())
      fonte.append('(').append(p.getClasse()).append(
          ")__entrada.get(\"").append(p.getNome()).append(
          "\")");
    else
      fonte.append("null");
    fonte.append(";\n");
  }
// Enxerta o código escrito pelo professor
fonte.append('\n').append(this.getCodigo_java()).append('\n');
fonte.append(";;\n"); // Evita que fórmulas simples gerem erros
// Monta o mapa com os parâmetros de saída, com statements no formato:
// __saida.add("nome",nome);
fonte.append("  Map<String,Object> __saida = new HashMap<String,Object>();\n");
if (this.getParametros() != null)
  for (Parametro p : this.getParametros())
    // if (!p.isTipoEntrada())
    fonte.append("  __saida.put(\"").append(p.getNome()).append(
        "\",").append(p.getNome()).append(");\n");
fonte.append("  return __saida;\n");
fonte.append(" }\n");

// Enxerta o código auxiliar
fonte.append('\n').append(this.getCodigo_java_auxiliar()).append('\n');
fonte.append(";;\n"); // Evita que fórmulas simples gerem erros

fonte.append("}\n");

// Prepara o compilador
javax.tools.JavaCompiler compilador = ToolProvider
    .getSystemJavaCompiler();
JavaSourceFromString javaString = new JavaSourceFromString(this
    .getNomePackage()
    + "." + this.getNomeClasse(), fonte.toString());
ArrayList<JavaSourceFromString> al = new ArrayList<JavaSourceFromString>();
al.add(javaString);
```

```
    // Saída (erros, etc.)
    ByteArrayOutputStream baos = new ByteArrayOutputStream();
    OutputStreamWriter osw = new OutputStreamWriter(baos);

    // Diretório-destino e classpath
    List<String> opcoes = new ArrayList<String>();
    opcoes.add("-d");
    opcoes.add(ConfigHelper.getClasspathDinamico());
    opcoes.add("-cp");
    opcoes.add(ConfigHelper.getClaspathApp());

    // Compila
    JavaCompiler.CompilationTask task = compilador.getTask(osw, null, null, opcoes, null, al);
    boolean sucesso = task.call();

    this.errosDeCompilacao = baos.toString();
    return sucesso;
  }
```

A.1.2 Carregamento Dinâmico de Classes

O código abaixo carrega uma clase, executa-a e a descarrega. Uma alternativa mais performática seria atrelar o classloader a algum objeto compartilhável pela aplicação, carregando-o uma vez só, mas para fins de estudo este método é bastante conveniente.

```
    // Instancia o classloader, apontando para o diretório de classes dinâmicas
    URL[] urls = new URL[1];
    try {
      urls[0] = new File(ConfigHelper.getClasspathDinamico()).toURI().toURL();
    } catch (MalformedURLException e) {
      // ... Trata o erro - é um problema de configuração
    }
    ClassLoader cl = new URLClassLoader(urls, Thread.currentThread().getContextClassLoader());

    // Carrega a classe através do classloader
    Class<Base> classe;
    Base codigo;
    classe = (Class<Base>) cl.loadClass(this.getNomePackage() + "." + this.getNomeClasse());
    codigo = (Base) classe.getConstructor(new Class[0]).newInstance(new Object[0]);
```

```
// Executa o método
Map<String, Object> resultado = codigo.executa(entrada);

// Descarrega a classe (matando as referências ao objeto, a ela e ao classloader)
ReferenceQueue weakQueueCl = new ReferenceQueue();
WeakReference weakRefCl = new WeakReference(classe, weakQueueCl);
weakRefCl.enqueue();
codigo = null;
classe = null;
cl = null;
System.gc();
System.gc();
return resultado;
```

A.1.3 Funções de Apoio Para Métodos Numéricos

Para criar passos de aula usando os trechos de código apresentados para a resolução do problema proposto através de métodos numéricos, é preciso inserir na seção "Código Auxiliar"as funções abaixo (que, respectivamente, calculam a função objetivo, seu hessiano, o gradiente e $\partial f/\partial\theta'|_\theta$ num ponto particular $\theta = (\theta_1, \theta_2)$).

```
private static Matrix f(double theta1, double theta2, Matrix x2, Matrix x3) {
  Matrix f = new Matrix(20,1);
  for (int t = 0; t <= 19; t++) {
    f.set(t, 0, theta1 + theta2 * x2.get(t,0) + (theta2 * theta2) * x3.get(t,0));
  }
  return f;
}

private static double[][] hess(double theta1, double theta2, Matrix y, Matrix x2, Matrix x3) {
  double[][] hess = new double[2][2];
  for (int t = 0; t <= 19; t++) {
    double f = f(theta1, theta2, x2, x3).get(t,0);
    hess[0][0] += 2;
    hess[0][1] += 2 * (x2.get(t,0) + 2 * theta2 * x3.get(t,0));
    hess[1][0] += 2 * x2.get(t,0) + 4 * theta2 * x3.get(t,0);
    hess[1][1] += 4 * x3.get(t,0) * (f - y.get(t,0))
    + (x2.get(t,0) + 2 * theta2 * x3.get(t,0))
    * (2 * x2.get(t,0) + 4 * theta2 * x3.get(t,0));
  }
  return hess;
```

```
}

private static double[] gradH(double theta1, double theta2, Matrix y, Matrix x2, Matrix x3) {
  double[] gradH = new double[2];
  for (int t = 0; t <= 19; t++) {
    double f = f(theta1, theta2, x2, x3).get(t,0);
    gradH[0] += -2 * y.get(t,0) + 2 * f;
    gradH[1] += 2 * (x2.get(t,0) + 2 * theta2 * x3.get(t,0)) * (f - y.get(t,0));
  }
  return gradH;
}

private static Matrix Z(double theta1, double theta2, Matrix x2, Matrix x3) {
  Matrix Z = new Matrix(20,2);
  for (int t = 0; t <= 19; t++) {
      Z.set(t, 0, 1);
      Z.set(t, 1, x2.get(t,0) + 2 * theta2 * x3.get(t,0));
  }
  return Z;
}
```

Além disso, é preciso definir como parâmetros de entrada as matrizes *y*, *x2 x3*, bem como os parâmetros reais *theta1* e *theta2*. Os parâmetros de saída devem ser as varíaveis reais *passo1* e *passo2* (representando a matriz de direção), e o trecho de código abaixo deve ser acrescentado ao código dos algoritmos (para garantir que tanto a matriz de direção quanto o ponto correspondente ao passo corrente sejam atualizadas para a execução do passo seguinte):

```
passo1 = passo.get(0,0);
passo2 = passo.get(1,0);
theta1 -= passo1;
theta2 -= passo2;
```

A.2 Configuração do Ambiente

Esta seção procura auxiliar pessoas interessadas em expandir as capacidades do código, facilitando a configuração do ambiente de trabalho.

O desenvolvimento pode ser feito em qualquer desktop que possua o Sun JDK

1.6 ou superior, ou um software compatível que permita compilar código para esta plataforma. Após a instalação do mesmo e o download do software (veja seção anterior), o caminho recomendado consiste em instalar o Apache Maven[3], que irá gerenciar de todo o processo de download de dependências e bibliotecas, bem como da compilação e empacotamento do software.

Uma vez no diretório-raiz da aplicação (`tf`), o comando `mvn package` irá executar o download das dependências, compilar o software e gerar o arquivo`tf.war` que pode ser instalado em qualquer *web container* compatível com a especificação Servlet 2.3. Uma alternativa interessante é instruir o Maven a executar uma versão mínima do Apache Tomcat já com a aplicação dentro. Para isso, basta usar o comando `mvn tomcat:run`.

Os diretórios estão divididos seguindo a padronização do Maven, incluindo os arquivos de configuração, localizados em `tf/src/resources`. Dois arquivos precisam ser configurados:

- O arquivo `hibernate.cfg.xml`, que está configurado para usar um banco de dados HSQLDB, residindo no diretório de deployment da própria aplicação - uma opção apropriada para desenvolvimento, mas que certamente deve ser revisada ao colocar o sistema em produção. Em qualquer caso, a propriedade `hbm2ddl.auto` pode ser des-comentada uma vez para criar o banco (e depois re-comentada para evitar apagá-lo!)
- O arquivo `tf.properties.exemplo` deve ser copiado para `tf.properties` e editado para apontar para dois diretórios: um que é onde o web container armazena as suas classes (tipicamente o `WEB_INF/classes` de onde o arquivo JAR foi descompactado), o outro é um diretório criado para armazenar as classes dinâmicas (que não pode estar dentro do diretório anterior, e para o qual a aplicação deve ter direitos de escrita).

Qualquer IDE pode ser utilizada, mas os fontes incluem um projeto do Eclipse.

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

Caso seja desejado usar esta IDE, deve-se instalar a mesma (`http://www.eclipse.org`) na versão 3.4 ou superior, e, em seguida, adicionar o suporte a Maven (`http://m2eclipse.codehaus.org/`) e o componente Web Tools Platform (WTP, `http://www.eclipse.org/webtools/`).

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.