6
Referências Bibliográficas

[1] ATTENE, M.; FALCIDIENO, B. ; SPAGNUOLO, M. Hierarchical mesh segmentation based on fitting primitives, **The Visual Computer**, v.22, n.3, p. 181–193, Fev. 2006.

[2] BINDEL, D.; DEMMEL, J.; KAHAN, W. ; MARQUES, O. On computing givens rotations reliably and efficiently, **ACM Transactions on Mathematical Software**, v.28, n.2, p. 206–238, Jun 2002.

[3] CUNO, A.; ESPERANÇA, C. ; OLIVEIRA, A. **Shape aware deformation using a Skeleton-Guided scheme**. p. 278–285. IEEE, Out. 2008.

[4] CUNO PARARI, A. E.; ESPERANÇA, C. ; OLIVEIRA, A. A. F. Shape-sensitive MLS deformation, **The Visual Computer**, v.25, n.10, p. 911–922, Mai 2009.

[5] DEBUNNE, G.; DESBRUN, M.; CANI, M. ; BARR, A. H. **Dynamic real-time deformations using space & time adaptive sampling**. p. 31–36. ACM Press, 2001.

[6] DEVLOO, P. R.; FERNANDES, P. D.; GOMES, S. M.; BRAVO, C. M. A. A. ; DAMAS, R. G. A finite element model for three dimensional hydraulic fracturing, **Mathematics and Computers in Simulation**, v.73, n.1-4, p. 142–155, Nov. 2006.

[7] GARLAND, M.; WILLMOTT, A. ; HECKBERT, P. S. **Hierarchical face clustering on polygonal surfaces**. p. 49–58. ACM Press, 2001.

[8] GILMAN, J. An efficient Finite-Difference method for simulating phase segregation in the matrix blocks in Double-Porosity reservoirs, **SPE Reservoir Engineering**, v.1, n.4, Jul 1986.

[9] GREENEMEIER, L. "when will virtual surgery make the cut?", **Scientific American**, Dez. 2007.

[10] MARTIN, S.; KAUFMANN, P.; BOTSCH, M.; GRINSPUN, E. ; GROSS, M. Unified simulation of elastic rods, shells, and solids, **ACM Transactions on Graphics**, v.29, n.4, p. 1, Jul 2010.

[11] MÜLLER, M.; HEIDELBERGER, B.; TESCHNER, M. ; GROSS, M. **Meshless deformations based on shape matching**. p. 471. ACM Press, 2005.

[12] MÜLLER, M.; KEISER, R.; NEALEN, A.; PAULY, M.; GROSS, M. ; ALEXA, M. **Point based animation of elastic, plastic and melting objects**. p. 141. ACM Press, 2004.

[13] NEALEN, A.; MÜLLER, M.; KEISER, R.; BOXERMAN, E. ; CARLSON, M. Physically based deformable models in computer graphics, **Computer Graphics Forum**, v.25, n.4, p. 809–836, Dez. 2006.

[14] NEFF, M.; FIUME, E. **A visual model for blast waves and fracture**. Em: PROCEEDINGS OF THE 1999 CONFERENCE ON GRAPHICS INTERFACE '99, p. 193–202, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[15] O'BRIEN, J. F.; BARGTEIL, A. W. ; HODGINS, J. K. Graphical modeling and animation of ductile fracture, **ACM Transactions on Graphics**, v.21, n.3, Jul 2002.

[16] OHTA, M.; KANAMORI, Y. ; NISHITA, T. Deformation and fracturing using adaptive shape matching with stiffness adjustment, **Computer Animation and Virtual Worlds**, v.20, n.2-3, p. 365–373, Jun 2009.

[17] PAULY, M.; KEISER, R.; ADAMS, B.; DUTRÉ, P.; GROSS, M. ; GUIBAS, L. J. Meshless animation of fracturing solids, **ACM Transactions on Graphics**, v.24, n.3, p. 957, Jul 2005.

[18] PRATT, V. Direct least-squares fitting of algebraic surfaces, **ACM SIGGRAPH Computer Graphics**, v.21, n.4, p. 145–152, Ago. 1987.

[19] SURAZHSKY, V.; SURAZHSKY, T.; KIRSANOV, D.; GORTLER, S. J. ; HOPPE, H. Fast exact and approximate geodesics on meshes, **ACM Transactions on Graphics**, v.24, n.3, p. 553, Jul 2005.

[20] TERZOPOULOS, D.; PLATT, J.; BARR, A. ; FLEISCHER, K. Elastically deformable models, **ACM SIGGRAPH Computer Graphics**, v.21, n.4, p. 205–214, Ago. 1987.

[21] WU, C.; HARRIS, J. M.; NIHEI, K. T. ; NAKAGAWA, S. Two-dimensional finite-difference seismic modeling of an open fluid-filled fracture: Comparison of thin-layer and linear-slip models, **Geophysics**, v.70, n.4, p. T57, 2005.

A
Apêndice I - Implementação

Este apêndice explica as muitas decisões a respeito de sua implementação as quais foram tomadas durante sua execução. Apresentaremos as bibliotecas e módulos do projeto e ilustraremos a estrutura de classes utilizada, com detalhes de sua implementação, na Seção A.4. Nos aprofundaremos em especial nas classes de matrizes programadas para as operações necessárias aos nossos algoritmos na Seção A.5.

A.1
Linguagem

Escolhemos o C++ como linguagem para o desenvolvimento em função tanto do grande número de bibliotecas e ferramentas do campo da Computação Gráfica disponíveis para ela como também nossa familiaridade com a linguagem.

A.2
Visualização

Embora nossos algoritmos de renderização não representem um diferencial, a arquitetura de visualização desenvolvida por nós nele utiliza o OpenGL 2 como framework para desenho 3D. É trivial a aplicação feita das capacidades de renderização que essa versão do OpenGL oferece, como pipeline programável através de shaders de vértice e fragmento. Contudo, é parte implícita da proposta realizar uma implementação dos nossos modelos de fratura e deformação cuja visualização se baseie em tecnologias de renderização modernas e compatíveis com as principais tendências tando da indústria como da academia.

A.3
Código

Além de utilizar OpenGL 2, integramos dois trabalhos prévios distintos, os quais apresentamos nessa Seção. Eles são:

- A classe `ModelOBJ`, do framework da dhpoware[1];
- O módulo de aproximação de formas do EfPiSoft, do IMATI-GE/CNR[2];

[1] http://www.dhpoware.com/
[2] http://efpisoft.sourceforge.net/

A.3.1 ModelOBJ

O formato OBJ foi escolhido como o utilizado graças a sua popularidade e por ser suportado pela maioria dos softwares 3D. A fim de utilizar esse simples formato, a classe `ModelOBJ` foi utilizada, uma classe C++ que integra um framework da dhpoware para leitura de malhas no formato Wavefront OBJ. Essa classe está integrada a série de aplicações de demonstração da dhpoware chamada "OpenGL 2 Shading Language Demo Series", e possui, além da funcionalidade de importar uma malha em formato OBJ, métodos úteis para geração de normais, tangentes e bitangentes dessa malha, o qual utilizamos. Alteramos essa classe para que satisfizisse nossas necessidades, transformando-a na classe `FractureMesh`.

A.3.2 EfPiSoft

A aplicação *EfPiSoft* foi implementada para demonstrar o algoritmo de segmentação de malha apresentado por Attene *et al.* em (1). O EfPiSoft utiliza a biblioteca JMeshLib para o processamento de malhas, e seu módulo de segmentação nos permite utilizá-lo na fase de segmentação de malhas para calcular os vértices de seus agrupamentos. Esse módulo permite ao seu cliente decidir qual conjunto de primitivas deseja-se utilizar, dentre planos, esferas e cilindros, e realiza o algoritmo de segmentação como apresentado em 1. Seu uso pode ser visto na Seção A.4.1, e na Figura A.1 esse módulo é representado pelo pacote HFP.

A.4 Estrutura de Classes

Nessa Seção apresentamos a estrutura de classes utilizada no intuito de, através dela, explicar de forma mais profunda a maneira através da qual nossos modelos foram traduzidos em classes e as relações entre esses componentes do sistema. Essa estrutura pode ser encontrada no diagrama de classes da Figura A.1, o qual ilustra as classes e enumerações mais relevantes da implementação. Dessas classes, os atributos e os métodos mais relevantes também foram destacados nessa Figura e suas funções serão explicitadas individualmente.

A.4.1 FractureMesh

Criada a partir da classe `ModelOBJ` da dhpoware, a classe `FractureMesh` representa um objeto o qual possui uma malha sujeita a deformações e

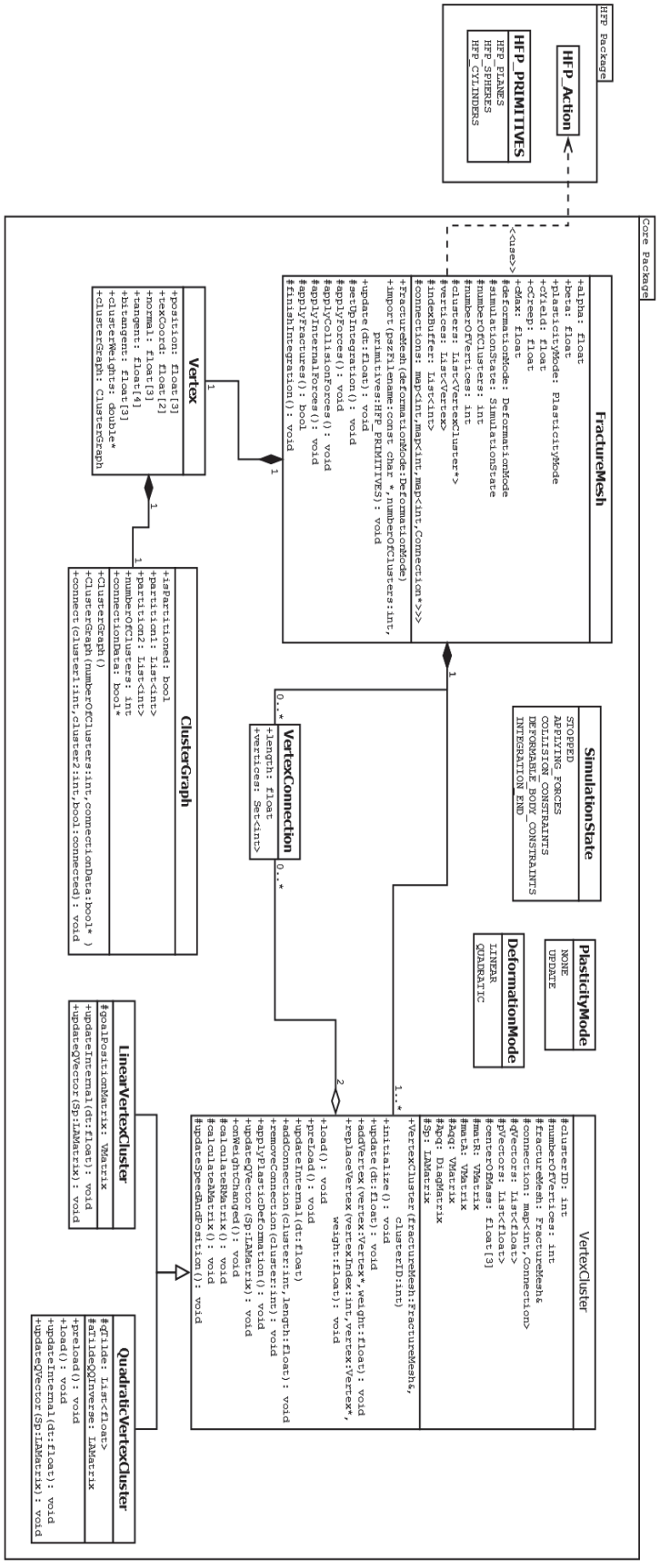

Figura A.1: Diagrama de classes do sistema, apresentando as principais classes relacionadas ao modelo

fraturas. São dois os principais conjuntos de atributos configuráveis de uma `FractureMesh`:

- Seu modo de deformação `deformationMode`, o qual pode ser Linear ou Quadrático, como explicado em 2.4.2 e 2.4.2, e seus parâmetros `alpha` e `beta`.
- Seu modo de plasticidade `plasticityMode` o que determina se deformações malhas atualizam ou não as posições originais dos vértices da malha, assim como seus parâmetros de plasticidade, `cYield`, `cCreep` e `cMax`.

Todos os parâmetros acima podem ser alterados durante a execução da simulação e atuam diretamente sobre a simulação da malha, exceto pelo modo de deformação, o qual é uma configuração fixa da `FractureMesh`, a qual precisa ser especificada durante sua importação. O método `import` é responsável por essa fase, descrita na Seção 2.2, responsável por preencher a lista de vértices da malha, `vertices`,com instâncias da classe `Vertex` criadas a partir do arquivo OBJ e também as estruturas auxiliares, como os `VertexClusters` e as `Connections`. Em função de uma limitação do módulo de segmentação do pacote HFP, que será usado posteriormente, precisamos realizar um pré-processamento nos arquivos OBJ: criamos uma versão temporária dele que possui apenas as coordenadas de posição e os índice dos vértices das faces, e um segundo arquivo temporário auxiliar que contém as informações de textura. Ignoramos informações adicionais relativas as normais do modelo e as recalculamos internamente, através dos métodos herdados de `ModelOBJ`.

A classe FractureMesh é a que coordena a simulação do objeto, se responsabilizando pelas aplicações de forças externas, tratamento de resposta de colisão e do tratamento das fraturas, através dos métodos `applyForces`, `applyCollisionForces` e `applyFractures`, respectivamente. No método `apply InternalForces` são delegados aos `VertexClusters` os cálculos das influências das forças internas de cada agrupamento.

A.4.2 VertexCluster

A classe abstrata `VertexCluster` faz parte de uma hierarquia para a qual existem duas subclasses, `LinearVertexCluster` e `QuadraticVertexCluster`, as quais possuem lógica interna a cada um desses modos de deformação. A classe `VertexCluster` é o cerne do algoritmo de deformação, e sua implementação recebeu atenção especial para que seu ciclo de execução fosse otimizado. Como resultado, essa classe possui muitas variáveis as quais podem

ser pré-calculadas, como os vetores de posição $\mathbf{q_i}$, implementados no atributo `qVector`, e todas as matrizes que dependem apenas deles (como, por exemplo, a matriz $\mathbf{A_{qq}}$).

De forma semelhante, otimizamos nossa classe reservando espaço para muitas matrizes as quais embora não possam ser pré-calculadas, seriam instanciadas a cada execução do laço de atualização (como as matrizes $\mathbf{A}$ e $\mathbf{R}$), economizando assim o tempo necessário para essas alocações e dealocações desnecessárias. Essa classe também oferece uma interface a qual possui alguns métodos para que vértices possam ser adicionados e substituídos, respectivamente os métodos `addVertex` e `replaceVertex`, os quais são utilizados tanto na criação dos `VertexClusters` como na execução das fraturas da classe `FractureMesh`.

A.4.3 ClusterGraph

O propósito da classe ClusterGraph é essencialmente guardar o estado atual das conexões que passam pelo vértice e detectar quando houve uma partição do conjunto de agrupamentos associados ao vértice, de acordo com o algoritmo descrito em 3.2.1. O atributo `connectionData` representa a matriz de conectividade dos agrupamentos ligados a cada vértice. Sempre que uma `Connection` se rompe, o método `connect` dos `ClusterGraphs` dos vértices dessa conexão são invocados, e os desligamentos das conexões causam alterações na matriz de conectividade do `ClusterGraph`. O atributo `isPartitioned` indica se houve ou não partição.

A.5 Matrizes

Desenvolvemos um pequeno módulo focado em matrizes e suas operações. Esse módulo possui três classes diferentes de matrizes. Essas classes implementam as operações básicas: além das aritméticas, implementam operações que calculam a transposta, inversa, determinante e norma de Frobenius. Essas classes são:

- `VMatrix`: classe que modela uma pilha de matrizes 4×4, com foco em operações de interesse da Computação Gráfica, como `lookAt` e `frustrum`. Essa classe foi utilizada em todos os pontos do código onde era suficiente uma matriz quadrada de ordem 4×4 e não eram necessárias operações diferentes das mostradas na Figura A.2.
- `DiagMatrix`: subclasse de `VMatrix`, a qual a estende no intuito de oferecer uma nova operação, a raiz quadrada matricial.

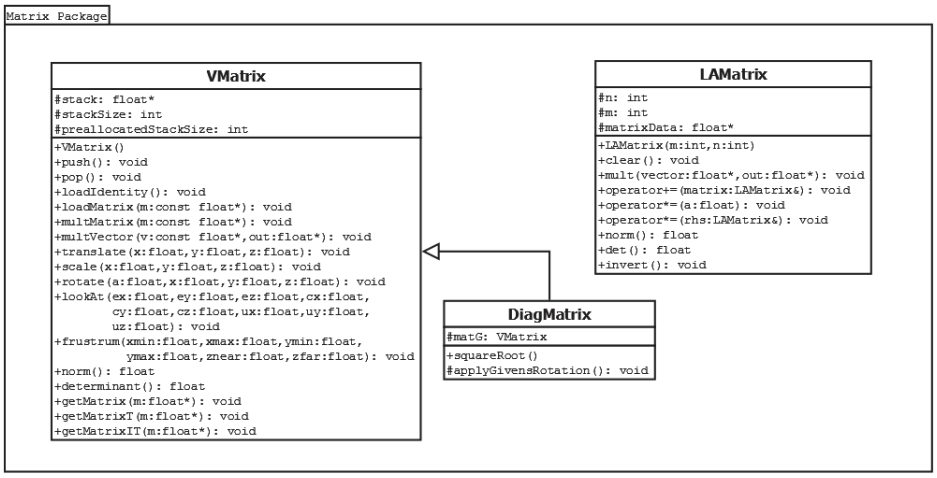

Figura A.2: Diagrama de classes do pacote de matrizes

– `LAMatrix`: classe para representar matrizes $m \times n$ quaisquer. Possui uma interface mais simples que a da classe `VMatrix`, e é voltada para Álgebra Linear em geral. Foi usada em todos os pontos onde precisávamos de matrizes não quadradas.

Das operações mencionadas nessa Seção, a única a qual consideramos merecedora de receber uma exposição foi a raiz quadrada matricial, a qual pode ser encontrada nas Seções abaixo.

A.5.1 Raiz Quadrada

Definimos $\sqrt{\mathbf{A}}$ como a matriz $\mathbf{B}$ tal qual satisfaz

$$\mathbf{BB} = \mathbf{A}. \tag{A-1}$$

A operação de raiz quadrada matricial surge em nossos algoritmos quando implementamos o procedimento descrito na Seção 2.4.2. A equação para determinar $\mathbf{S}$

$$\mathbf{S} = \sqrt{{\mathbf{A_{pq}}}^T \mathbf{A_{pq}}}. \tag{A-2}$$

Como ${\mathbf{A_{pq}}}^T \mathbf{A_{pq}}$ é simétrica, o Teorema Espectral nos garante que caso $\mathbf{A_{pq}}$ seja não singular ${\mathbf{A_{pq}}}^T \mathbf{A_{pq}}$ é diagonalizável e nos permite basear uma implementação da raiz quadrada no método da diagonalização.

O método da diagonalização consiste em encontrar uma matriz diagonal $\mathbf{D}$ tal que

$$\mathbf{A} = \mathbf{VDV}^{-1}. \tag{A-3}$$

Encontrando tal matriz, podemos escrever $\mathbf{B} = \mathbf{V}\sqrt{\mathbf{D}}\mathbf{V}^{-1}$, pois

$$\mathbf{V}\sqrt{\mathbf{D}}\mathbf{V}^{-1}\mathbf{V}\sqrt{\mathbf{D}}\mathbf{V}^{-1} = \mathbf{V}\sqrt{\mathbf{D}}\sqrt{\mathbf{D}}\mathbf{V}^{-1} = \mathbf{V}\mathbf{D}\mathbf{V}^{-1} = \mathbf{A}. \quad \text{(A-4)}$$

A raiz quadrada matricial $\mathbf{E}$ da matriz diagonal $\mathbf{D}$ possui definição trivial como uma outra matriz diagonal, cujos elementos e_i da diagonal de $\mathbf{E}$ satisfazem a seguinte equação que os relaciona com os elementos d_i da diagonal de $\mathbf{D}$:

$$e_i = \sqrt{d_i}. \quad \text{(A-5)}$$

Logo, diagonalizamos ${\mathbf{A_{pq}}}^T\mathbf{A_{pq}}$ a fim de encontrar sua raiz quadrada.

A.5.2 Método de Jacobi

O método de diagonalização implementado por nós foi o Método de Jacobi. Esse método consiste em aplicar à $\mathbf{A}$ sucessivas rotações de Givens $\mathbf{G_i}$ (2) a fim de aproximar a matriz $\mathbf{A}$ de uma matriz diagonal. Uma rotação de Givens $\mathbf{G}(i, j, \theta)$ pode ser escrita como

$$\mathbf{G}(i,j,\theta) = \begin{bmatrix} 1 & \dots & 0 & \dots & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \dots & cos(\theta) & \dots & -sen(\theta) & \dots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \dots & sen(\theta) & \dots & cos(\theta) & \dots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix},$$

onde os valores para $cos(\theta)$ nos componentes g_{ii} e g_{jj} da matriz enquanto que $sen(\theta)$ ocupa o componente g_{ji} e $-sen(\theta)$ o g_{ij}.

O método de Jacobi se baseia no fato de que para uma matriz simétrica $\mathbf{S_n}$, podemos escrever

$$\mathbf{S_{n+1}} = \mathbf{G}^T\mathbf{S_n}\mathbf{G}, \quad \text{(A-7)}$$

onde $\mathbf{S_{n+1}}$ é também simétrica e similar a $\mathbf{S_n}$, ou seja, $\mathbf{S_{n+1}}$ equivale a $\mathbf{S_n}$ exceto por uma mudança de base. $\mathbf{S_{n+1}}$ também compartilha com $\mathbf{S_n}$ sua norma de Frobenius, pois $\mathbf{G}$ é ortogonal, independente dos parâmetros i, j e θ escolhidos para $\mathbf{G}$. Podemos, então, nos aproveitar da preservação dessas características para escolher esses parâmetros de forma a mais rapidamente aproximar, a cada iteração do método, $\mathbf{S_{n+1}}$ de uma matriz diagonal, simétrica e similar a matriz original $\mathbf{S_0}$. Seja s_{ij}^n a componente s_{ij} da matriz simétrica $\mathbf{S_n}$. Escolhendo os parâmetros de $\mathbf{G}$ corretamente, podemos fazer com que

$$s_{ij}^{n+1} = 0. \quad \text{(A-8)}$$

Essa técnica é o conceito básico do método de Jacobi e faz parte de sua iteração. A cada passo, eliminamos os componentes não diagonais (s_{ij}^n para $i \neq j$) da matriz $\mathbf{S_n}$ para diagonalizá-la. A escolha de qual componente é eliminado a cada passo é feita, no intuito de otimizar o efeito da rotação de Givens, de forma a satisfazer

$$|s_{ij}^n| > |s_{kl}^n| \ \forall s_{kl}^n \ de \ \mathbf{S_n}. \tag{A-9}$$

Chama-se o componente s_{ij}^n de *pivô*. Já tendo determinado quem é o componente e suas coordenadas i e j, resta-nos determinar qual θ anula esse componente na matriz $\mathbf{S_{n+1}}$. Como

$$s_{ij}^{n+1} = cos(2\theta)s_{ij}^n + sen(2\theta)(s_{ii}^n - s_{jj}^n), \tag{A-10}$$

podemos igualar essa equação a zero e encontrar, caso $s_{jj}^n \neq s_{ii}^n$:

$$\theta = \frac{atan(\frac{2s_{ij}^n}{s_{jj}^n - s_{ii}^n})}{2}. \tag{A-11}$$

Caso $s_{jj}^n = s_{ii}^n$, a solução é trivial:

$$\theta = \frac{\pi}{4}. \tag{A-12}$$

B
Apêndice II - Estabilidade de Sistemas Massa-Mola

Neste apêndice, reproduziremos a breve análise de Muller *et al.* (11) na qual busca-se justificar a estabilidade do esquema de integração escolhido.

B.1
Sistema Massa-Mola Clássico

Seja um sistema massa-mola baseado na lei de Hooke tal que l_0 seja o comprimento da mola em repouso, k seu coeficiente de elasticidade e m represente a massa do objeto preso à mola. A posição ao longo do eixo da mola do objeto pode ser descrita como $x(t)$, e dizemos que a força que atua sobre o objeto é:

$$f = -k(x(t) - l_0). \tag{B-1}$$

Enuncia-se esse sistema utilizando o método de integração de Euler modificado, o qual nos dá as equações:

$$v(t+h) = v(t) + h\frac{-k(x(t) - l_0)}{m}, \tag{B-2}$$

$$x(t+h) = x(t) + hv(t+h) \tag{B-3}$$

Esse sistema de equações, se escrito de forma matricial $s(t+h) = \mathbf{U}s(t)$ para $s(t) = \left[\begin{array}{cc} v(t) & x(t) \end{array}\right]$, resulta em:

$$\left[\begin{array}{c} v(t+h) \\ x(t+h) \end{array}\right] = \left[\begin{array}{cc} 1 & -\frac{kh}{m} \\ h & 1 - \frac{h^2k}{m} \end{array}\right] \left[\begin{array}{c} (t) \\ x(t) \end{array}\right]. \tag{B-4}$$

Usamos a matriz $\mathbf{U}$ para inspecionar esse sistema quanto as condições na qual ele se mantém estável. Para que a estabilidade possa ser matematicamente garantida, é necessário que seus autovalores e_+ e e_- satisfaçam $|e_+| \leq 1, |e_-| \leq 1$. A fórmula explícita para esses autovalores é:

$$e_\pm = 1 - \frac{1}{2m}(h^2k \pm \sqrt{-4mh^2k + h^4k^2}). \tag{B-5}$$

O autovalor e_+, contudo, depende de h para que sua magnitude seja menor ou igual a um. Podemos mostrar isso considerando $h > 2\sqrt{\frac{m}{k}}$:

$$\frac{\sqrt{-4mh^2k + h^4k^2}}{2m} > \frac{\sqrt{-16m^2 + 16m^2}}{2m} = 0, \tag{B-6}$$

$$\frac{h^2k}{2m} > \frac{4mk}{2mk} = 2. \tag{B-7}$$

Aplicando a Equação B-6 e a Equação B-7 à fórmula de e_+, temos que

$$e_+ < -1. \tag{B-8}$$

Essa inequação implica em um sistema instável caso $h \geq 2\sqrt{\frac{m}{k}}$.

B.2 Variação do Sistema Massa-Mola

A fim de utilizar um sistema massa-mola incondicionalmente estável, pode-se utilizar uma variação do sistema massa-mola clássico. Desconsiderando forças externas, pode-se escrevê-lo como:

$$v(t+h) = v(t) + \alpha\frac{(l_0 - x(t))}{h}, \tag{B-9}$$

$$x(t+h) = x(t) + hv(t+h). \tag{B-10}$$

Nessa reformulação, o parâmetro $\alpha \in [0,1]$ representa a influência da força de restituição da mola no movimento do objeto, ou seja, um parâmetro análogo à constante elástica k. Com efeito, podemos notar que para $\alpha = 1$, temos um objeto o qual sempre possuirá velocidade suficiente para atingir sua posição de repouso em um passo de integração, pois passamos a ter a seguite equação de integração em $x(t+h)$

$$x(t+h) = x(t) + hv(t+h) = x(t) + hv(t) + (l_0 - x(t)) = l_0 + hv(t), \tag{B-11}$$

enquanto que para $\alpha = 0$, a mola simplesmente não exerce influência sobre o objeto e ele pode se movimentar sem forças de restituição.

Na forma matricial, escrevemos o novo sistema $s(t+h) = \mathbf{U}'s(t) + C$ como

$$\begin{bmatrix} v(t+h) \\ x(t+h) \end{bmatrix} = \begin{bmatrix} 1 & -\frac{\alpha}{h} \\ h & 1-\alpha \end{bmatrix} \begin{bmatrix} (t) \\ x(t) \end{bmatrix} + \begin{bmatrix} \alpha l_0 h \\ \alpha l_0 \end{bmatrix}. \tag{B-12}$$

A fim de realizar a análise de estabilidade, computamos os autovalores $e_\pm$ de $\mathbf{U}'$, e encontramos a seguinte equivalência:

$$e_\pm = (1 - \frac{\alpha}{2}) \pm i\frac{\sqrt{4\alpha - \alpha^2}}{2}. \tag{B-13}$$

Tanto $|e_+| = 1$ como $|e_-| = 1$, o que garante a estabilidade dessa variação sem amortecimento do sistema massa-mola. Logo, essa formulação do método de integração é incondicionalmente estável.