METODOLOGIA DE BUSCA DE SOLUÇÕES ESPECIAIS EM SISTEMAS DINÂMICOS NÃO LINEARES

Héctor Napoleão Cozendey da Silva

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia Química, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia Química.

Orientadores: Argimiro Resende Secchi
Príamo Albuquerque Melo Junior

Rio de Janeiro
Junho de 2015

METODOLOGIA DE BUSCA DE SOLUÇÕES ESPECIAIS EM SISTEMAS DINÂMICOS NÃO LINEARES

Héctor Napoleão Cozendey da Silva

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA QUÍMICA.

Examinada por:

Prof. Argimiro Resende Secchi, D.Sc.

Prof. Príamo Albuquerque Melo Junior, D.Sc.

Prof. Marcelo Amorim Savi, D.Sc.

Prof. Heloísa Lajas Sanches, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
JUNHO DE 2015

Silva, Héctor Napoleão Cozendey da

Metodologia de busca de soluções especiais em sistemas dinâmicos não lineares/Héctor Napoleão Cozendey da Silva. – Rio de Janeiro: UFRJ/COPPE, 2015.

XV, 107 p.: il.; 29,7 cm.

Orientadores: Argimiro Resende Secchi

Príamo Albuquerque Melo Jr.

Dissertação (mestrado) – UFRJ/ COPPE/ Programa de Engenharia Química, 2015.

Referências Bibliográficas: p. 51-55.

1. Análise Dinâmica. 2. Sistemas de Equações Diferenciais não-lineares. I. Secchi, Argimiro Resende *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Química. III. Título.

"Não há pactos entre leões e homens."(Aquiles)
"Mas até mesmo os inimigos podem demonstrar respeito."(Príamo)
"Você fala da guerra como se fosse um jogo. Mas quantas esposas esperam nos portões de Tróia os maridos que nunca irão ver de novo?"(Hector)
"Talvez seu irmão possa consolá-las. Ouvi dizer que ele é bom em encantar mulheres de outros homens". (Aquiles)
"Você diz que está disposto a morrer por amor, mas você não sabe nada sobre a morte, nem sobre o amor!"(Hector)

Agradecimentos

Aos meus orientadores, Argimiro Resende Secchi e Príamo A. Melo Jr., por todo ensinamento e orientação, pela confiança e pela amizade.

Agradeço à minha família, em especial aos meus pais, Eliana e Cláudio, e ao meu irmão Humberto, por todo apoio, incentivo, ajuda, compreensão, paciência e carinho durante toda a minha vida.

Ao grupo do LMSCP e LADES, que sempre me trataram com atenção e respeito, me auxiliando nos momentos de dúvida e sempre mostrando onde o trabalho poderia ser melhorado.

Aos amigos Bruno Francisco Oechsler e Saulo Barbosa Mansur, meu especial agradecimento pelos debates sobre o tema deste trabalho e a atenção para conversar mesmo nos momentos de aflição.

Agradeço também aos amigos Mariana Kuster Moro, Felipe Tadeu Fiorini Gomide, Jimena Ferreira Quagliata, João Luiz Cardia Tavares da Costa, Katherine Amaral Frias, Francine Duarte Castro, Joana Athayde Lapagesse Correa, Gilliani Peixoto Miranda e Rogger Azevedo de Souza Perazzo pelos momentos de descontração, diversão, açaí e pelo apoio.

Obrigado à CAPES e ao CNPq pelo suporte financeiro durante todo o desenvolvimento deste trabalho.

Agradeço pela atenção e boa vontade dos funcionários da secretaria do PEQ, sempre prontos a nos ajudar; a todos os professores do Programa de Engenharia Química da COPPE pelos ensinamentos.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

METODOLOGIA DE BUSCA DE SOLUÇÕES ESPECIAIS EM SISTEMAS DINÂMICOS NÃO LINEARES

Héctor Napoleão Cozendey da Silva

Junho/2015

Orientadores: Argimiro Resende Secchi
Príamo Albuquerque Melo Junior

Programa: Engenharia Química

Este trabalho apresenta uma metodologia para mapear o espaço paramétrico de um sistema de equações diferenciais ordinárias autônomas. A caracterização do comportamento dinâmico é feita com o cômputo do expoente de Lyapunov que, associado ao algoritmo de otimizaçao DIRECT, avalia regiões paramétricas nas quais soluções estacionárias, periódicas e atratores caóticos são os mais esperados. Diversos modelos foram testados.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

SPECIAL SOLUTIONS SEARCH METHODOLOGY IN NON LINEAR DYNAMIC SYSTEMS

Héctor Napoleão Cozendey da Silva

June/2015

Advisors: Argimiro Resende Secchi
Príamo Albuquerque Melo Junior

Department: Chemical Engineering

This work presents a methodology to map the parameter space of an autonomous system of ordinary differential equations. The characterization of the dynamic behavior is done with the calculation of Lyapunov exponent associated with the optimization algorithm DIRECT to evaluate parametric regions where stationary, periodic and chaotic attractors are the most expected solutions. Various dynamic models have been tested.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Símbolos

B	Parâmetro do modelo do reator CSTR isotérmico clássico
b	Parâmetro do modelo de Lorenz
$\boldsymbol{d}_0$	Perturbação em relação à órbita periódica
Ef	Eficiência do processamento paralelo
$\boldsymbol{f}$	Sistema de transformações não lineares
$\boldsymbol{f}_{lin}$	Sistema linearizado
$\mathbf{J}$	Matriz Jacobiana
N	Dimensão do vetor de variáveis de estado
r	Parâmetro do sistema de Lorenz
s	Parâmetro do sistema de Lorenz
Sp	*Speedup* do processamento paralelo
T	Período
t	Tempo
$\boldsymbol{x}$	Vetor de variáveis de estado
$\boldsymbol{x}^*$	Ponto sobre a órbita periódica
$\boldsymbol{x}_e$	Ponto de equilíbrio
y	Variável de estado
α	Parâmetro do modelo de um biorreator
β	Parâmetro dos modelos do reator CSTR exotérmico clássico e de um biorreator

γ	Parâmetro do modelo do reator CSTR exotérmico clássico
Λ	Valor característico da matriz Monodromia
λ	Vetor de parâmetros
μ	Expoentes de Lyapunov
ϕ	Matriz Monodromia
φ	Trajetórias
σ	Expoentes de Floquet
θ	Variável de estado do modelo do reator CSTR exotérmico clássico
θ_c	Parâmetro do modelo do reator CSTR exotérmico clássico
ξ	Vetor característico da matriz Monodromia

Lista de Abreviaturas

BP	Ponto de bifurcação
CSTR	*Continous Stirred Tank Reactor*
DIRECT	Método de otimização *Dividing Rectangles*
HB	Bifurcação de Hopf
LP	Ponto limite
PBLP	Ponto de bifurcação limite periódico
PDP	Ponto de duplicação de período
PE	Processador escravo
PLP	Ponto limite periódico
PM	Processador Mestre
PSO	Método de otimização enxame de partículas
RPDE	Ramo de período duplo estável
RPDI	Ramo de período duplo instável
RPE	Ramo periódico estável
RPI	Ramo periódico instável

Capítulo 1

Introdução

Acontecimentos naturais estão repletos de fenômenos dinâmicos não lineares. Os comportamentos dinâmicos podem ser descritos por sistemas de equações matemáticas, dando origem a modelos matemáticos, podendo ser analíticos, empírico ou hibrido.

A multiplicidade de soluções estacionárias, periodicidade e comportamentos mais complexos são características da análise dinâmica de sistemas não lineares. Os anos 50 do século XX foram marcados por estudos de van Heerden (1953), os quais apontaram o comportamento dinâmico de reatores tipo tanque agitado, e a possibilidade de existência de múltiplas soluções estacionárias e soluções estacionárias instáveis em sistemas de reação.

Muitos fenômenos dinâmicos podem ser descritos por sistemas autônomos de equações diferenciais ordinárias não lineares:

$$\frac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x}, \lambda) \tag{1.1}$$

na qual $\mathbf{x}$ é o vetor de variáveis de estados, λ o vetor de parâmetros, t o tempo e $\mathbf{f}$ é o sistema de transformações não lineares.

Existem três classificações para sistemas dinâmicos não lineares (OURIQUE, 2000): explosivos, conservativos e dissipativos. Se houver ganho de energia do exterior, ou seja expansão do volume de estados possíveis, diz-se que o sistema é explosivo. Em sistemas conservativos ou Hamiltoniano, o volume de estados possíveis e a energia total se conservam. Um sistema considerado dissipativo tem o volume contraído continuamente ao longo do tempo. Esse constitui a grande maioria dos sistemas de interesse da engenharia química.

Na década de 1950, analisar o comportamento dinâmico de sistemas era, essencialmente, feito por meio de planos de fase. Esse método consiste em representar trajetórias no espaço das variáveis de estado do sistema. Entretanto, se houver alguma alteração no valor de algum parâmetro, é necessário repetir a análise.

Atualmente, a busca de soluções especiais em sistemas dinâmicos não lineares é feita, majoritariamente, por meio da continuação paramétrica de soluções estacionárias, detecção de pontos de bifurcação de Hopf (HB), continuação paramétrica de HB e a simulação dinâmica de casos escolhidos. Apesar de ser um método clássico, introduzido por Doedel e Heinemann (1983), é pouco eficiente para varrer todos os parâmetros, considerando que poucos são analisados por vez, além dessa metodologia exigir o conhecimento de, pelo menos, uma solução estacionária.

Uma alternativa é o método da busca por varredura, que consiste em realizar diversas simulações do sistema para diversos conjuntos paramétricos. Isso é feito excessivamente até que o espaço de parâmetros esteja bem explorado. No entanto, esse método não garante que todas as soluções sejam encontradas.

Motivado pelo estudo da busca mais efetiva de soluções especiais no espaço paramétrico o objetivo deste trabalho é desenvolver um algoritmo em processamento paralelo para agilizar a busca e análise de soluções especiais de sistemas dinâmicos não lineares, autônomos e dissipativos.

A estrutura desta dissertação é composta por seis capítulos. O Capítulo 1 expressa alguns conceitos introdutórios, a motivação e os objetivos deste estudo. No Capítulo 2, apresenta-se uma breve revisão da literatura sobre dinâmica não-linear e os fundamentos teóricos básicos para entender a metodologia aplicada. O Capítulo 3 expõe a metodologia desenvolvida para abordar os casos estudados, dividida em explorar o espaço paramétrico e caracterizar as soluções estacionárias e dinâmicas dos sistemas estudados através do cálculo dos expoentes de Lyapunov. No Capítulo 4 são apresentados os resultados obtidos e as respectivas discussões para os casos estudados com a metodologia implementada. Por fim, as conclusões e sugestões são apresentadas no Capítulo 5. As referências bibliográficas utilizadas na fundamentação deste estudo são mostradas no Capítulo 6.

Capítulo 2

Revisão Bibliográfica

Considerando a abrangência de estudos relativos à Teoria de Sistemas Dinâmicos Não Lineares, este capítulo se limitará a apresentar alguns conceitos fundamentais para a compreensão deste trabalho. Para leitura aprofundada sobre o tema, sugere-se os trabalhos de Glendinning (1995), Argyris *et al.* (1994), Jackson (1991), Wiggins (1990), Ruelle (1989), Thompson e Stewart (1986), Guckenheimer e Holmes (1983), Kubicek e Marek (1983), Chow e Hale (1982), Ioss e Joseph (1980), Arnol'd (1978, 1988), Mardsen e McCracken (1976) e Rabinowitz (1977).

2.1 O Estado Estacionário

Este estudo abordará apenas modelos matemáticos de sistemas a parâmetros concentrados autônomos, portanto não será exposta a Teoria de Sistemas Distribuídos. Por definição, variáveis de estado em modelos de sistemas concentrados não são função da posição espacial.

Um sistema dinâmico é denominado autônomo quando ele não depende explicitamente do tempo (SEYDEL, 2010). Esse sistema consiste de três características básicas: tempo, espaço de estados ou espaço de fases e lei de evolução.

Na Equação 1.1, se $\mathbf{f}$ é contínua, as soluções $\mathbf{x}(t)$ deste sistema, quando representadas em espaço de estados, cujos eixos são variáveis de estado, formam um conjunto de trajetórias, órbitas ou linhas de fluxo (OURIQUE, 2000). Assim, a solução $\mathbf{x}(t,\lambda)$ converge para uma figura geométrica, denominada atrator, quando $t \to \infty$ em sistemas dissipativos (PINTO, 1991). Podem-se identificar particularmente quatro tipos de atratores:

- Estado estacionário ou ponto fixo;
- Ciclo limite;
- Toro;

- Atrator estranho.

Um sistema em regime estacionário é inalterável no tempo, ou seja, quando o sistema atinge o equilíbrio, as variáveis de estado assumem valores constantes $\mathbf{x}_e$. Isto implica que o sistema fica reduzido a:

$$\frac{d\mathbf{x}}{dt} = 0 \tag{2.1}$$

Portanto, a solução estacionária, estado estacionário, ponto de equilíbrio ou ponto fixo pode ser obtida pela resolução do sistema de equações algébricas:

$$\mathbf{f}(\mathbf{x}_e, \lambda) = \mathbf{0} \tag{2.2}$$

No espaço de fases, o estado estacionário é visto como um ponto em que as trajetórias se aproximam ou se afastam dependendo da estabilidade de cada ponto. Esse pode ser classificado como estável, assintoticamente estável ou instável.

De acordo com o comportamento do sistema sujeito a uma perturbação, pode-se classificar o tipo de ponto de equilíbrio. Se a resposta a uma pequena perturbação permanece pequena com o passar do tempo, a solução é estável, ou Lyapunov estável (SAVI, 2006). Enquanto que, se uma trajetória quando iniciada próxima a esse ponto, se afastar continuamente dele com o passar do tempo, a solução é classificada como instável. Por fim, quando o sistema dinâmico é sujeito a uma pequena perturbação, se o desvio em relação ao ponto de equilíbrio tende a zero, à medida que o tempo tende ao infinito, a solução estacionária é dita assintoticamente estável (RODRIGUES, 2011).

Essas soluções são exemplos de atratores. O conjunto de todas as condições iniciais a partir dos quais as trajetórias convergem para o atrator considerado é chamado de domínio de atração (MELO e PINTO, 2008).

Assim, quando o sistema é linearizado em torno de $\mathbf{x}_e$, a estabilidade das soluções estacionárias pode ser verificada pelo cálculo dos valores característicos da matriz Jacobiana ($\mathbf{J}$) da seguinte forma:

$$\mathbf{f}_{lin}(x) = \mathbf{f}(\mathbf{x}_e) + \mathbf{J}(\mathbf{x}_e)(\mathbf{x} - \mathbf{x}_e) \tag{2.3}$$

na qual,

$$\mathbf{J}(x) = \frac{\partial \mathbf{f}}{\partial \mathbf{x}} \tag{2.4}$$

e $\mathbf{f}_{lin}(\mathbf{x})$ é o sistema linearizado.

Quando todos os valores característicos forem reais e negativos, todas as trajetórias iniciadas em torno do ponto de equilíbrio convergem para ele. Sendo classificado

como nó estável, enquanto que, se todos os valores característicos forem reais positivos, as trajetórias próximas ao ponto de equilíbrio se afastam e ele é chamado de nó instável. A Figura 2.1 apresenta planos de fases de algumas estruturas especiais, conforme se segue contextualizando.

Vale ressaltar que, como a análise é feita em torno do estado estacionário, ela é classificada como comportamento local da solução. Em suma, se a parte real de todos os valores característicos são negativos, a solução é localmente estável, enquanto que basta ter um valor característico com a parte real positiva para a solução perder a estabilidade (SEYDEL, 2010).

No caso particular de um sistema com duas variáveis de estado, a Jacobiana é uma matriz de ordem dois e possui apenas dois valores característicos. Se um for real positivo e outro real negativo o ponto de estado estacionário é chamado de ponto de sela.

Quando, dentre os valores característicos, existir, pelo menos um par complexo conjugado com parte real não nula. O estado estacionário é chamado de foco e as trajetórias tem uma forma espiral em torno do estado estacionário (SEYDEL, 2010). Se a parte real for nula, o ponto é chamado de centro. A orientação da rotação depende do sentido do vetor $\frac{d\mathbf{x}}{dt}$.

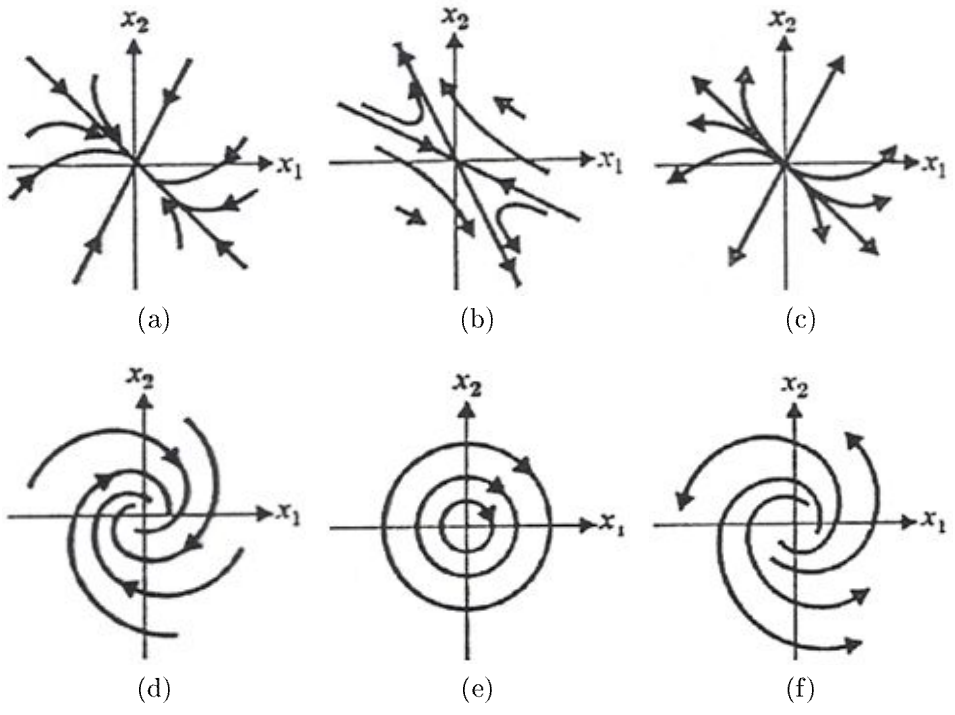


Figura 2.1: Plano de fases de algumas estruturas especiais: a) Nó estável; b) Ponto de sela; c) Nó instável; e) Centro; f) Foco instável. (FRIEDLY, 1972)

2.2 Teoria da Bifurcação

A Teoria de Bifurcações estuda as alterações de origem qualitativa que ocorrem na estrutura das trajetórias de um sistema, quando os parâmetros dos quais este sistema depende são perturbados. Se a análise é feita nas vizinhanças das soluções, tem-se a Teoria Local de Bifurcações. E, quando se estudam as mudanças das trajetórias em uma região extensa do espaço, tem-se a Teoria Global de Bifurcações (SEYDEL, 2010).

Este estudo utilizará amplamente o conceito de bifurcação, conforme mencionado, ele descreve mudanças que fazem com que, por exemplo, um sistema dinâmico instável, ao ter o parâmetro perturbado, leva a comportamentos dinâmicos distintos, podendo ser definido como um ponto de bifurcação.

O fenômeno da bifurcação está relacionado com a possibilidade de existência de comportamento caótico, no sentido de que um sistema dinâmico que não apresenta algum tipo de bifurcação não pode apresentar uma resposta caótica, tratada no item 2.4. Entretanto, o inverso não é verdadeiro. Ou seja, um sistema que contém bifurcações não necessariamente apresenta uma resposta caótica (SAVI, 2006).

Em estudo de bifurcações locais, procuram-se normalmente as ramificações ou ramos de soluções permanentes utilizando-se o Método de Continuação Paramétrica (DOEDEL *et al.*, 1986). Nesta técnica, um parâmetro, chamado de parâmetro de continuação, é variado continuamente a partir de uma solução conhecida, analisando-se a dependência da solução em relação aos parâmetros. Em sendo encontrado um ponto singular com derivadas nulas ou matriz Jacobiana singular, a natureza da singularidade e a estrutura dos sistemas dinâmicos que se organizam em torno deste ponto são analisadas (FREITAS FILHO, 1993). Isso faz com que a combinação da Teoria de Bifurcações e os Métodos da Continuação tenha vasta aplicação nos estudos de multiplicidade de soluções.

Porém, conforme apontado por Pinto (1991), esse método apresenta duas desvantagens. Inicialmente é necessário conhecer pelo menos uma solução estacionária do sistema dinâmico. Por fim, não se pode garantir que outros padrões dinâmicos não existam no espaço analisado, mas apenas afirmar que estes padrões não foram encontrados.

Ainda, algumas singularidades ocorrem em regiões paramétricas estreitas e podem acabar, mesmo após um estudo exaustivo, passando despercebidas devido às aproximações numéricas. Assim, seja qual for o sistema dinâmico, a Teoria de Bifurcações não garante encontrar o conjunto total de estruturas dinâmicas existentes, embora esse conjunto obtido possa ser, por vezes, bastante extenso (PINTO, 1991).

Pontos de ramificação são aqueles em que dois ou mais ramos de soluções permanentes se encontram. Nesses pontos, a quantidade de soluções do sistema muda

com a variação do parâmetro, que pode ser determinado por (SEYDEL,2010):

$$\mathbf{f}(\mathbf{x}_e, \lambda) = \mathbf{0},\ det\left[\frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x}_e, \lambda)\right] = 0,\ det\left[\frac{\partial^2 \mathbf{f}}{\partial \mathbf{x}^2}(\mathbf{x}_e, \lambda)\right] = 0,\ \ldots,\ det\left[\frac{\partial^k \mathbf{f}}{\partial \mathbf{x}^k}(\mathbf{x}_e, \lambda)\right] \neq 0 \tag{2.5}$$

implicando que o sistema possui k soluções estacionárias.

Ainda com a aplicação da Teoria de Bifurcações, é possível encontrar três tipos de pontos de ramificação estacionários: o ponto limite, o ponto de bifurcação e o ponto de bifurcação limite (OURIQUE, 2000). Um Ponto Limite (LP) acontece quando um valor característico real da matriz Jacobiana cruza o eixo imaginário no plano complexo. Fazendo com que a matriz se torne singular, mas:

$$\frac{\partial \lambda}{\partial \mathbf{x}} \neq \mathbf{0} \tag{2.6}$$

Em um ponto de bifurcação (BP), surgem quatro trajetórias, sendo que duas possuem a mesma tangente no ponto de bifurcação. Enquanto que, se os pontos limite e de bifurcação coincidirem, esse passa a ser chamado de ponto de bifurcação limite.

As bifurcações de ramos permanentes são classificadas como *pitchfork* supercríticas, pitchfork subcríticas ou transcríticas. As duas primeiras podem ocorrer em um ponto de bifurcação limite, enquanto que a última ocorre em um ponto de bifurcação (OURIQUE, 2000).

A bifurcação é classificada como pitchfork supercrítica quando um ramo de soluções estacionárias estáveis perde a estabilidade ao passar pelo ponto de bifurcação limite e os ramos originados são estáveis.

Assim como, quando o contrário acontece, um ramo de soluções estacionárias instáveis, ao passar pelo ponto de bifurcação limite, torna-se estável e os ramos originados são instáveis, a bifurcação é dita pitchfork subcrítica.

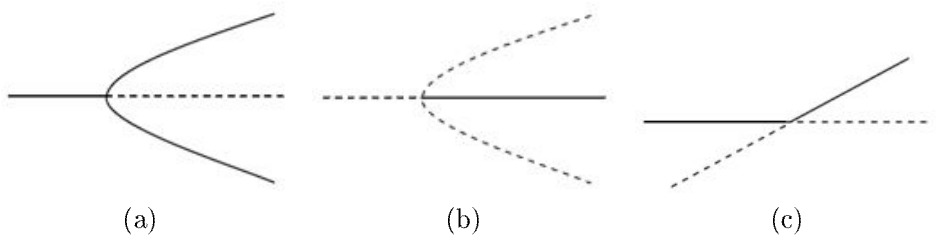


Figura 2.2: Bifurcações do tipo pitchfork e trancrítica, respectivamente: a) Supercrítica; b) Subcrítica; c) transcrítica. (OURIQUE, 2000)

Na bifurcação transcrítica, o que ocorre são dois ramos de soluções se cruzando no ponto de bifurcação. E ao se cruzarem, trocam de estabilidade. Conforme exem-

plificado na Figura 2.2, os três casos já discutidos. Lembrando que, usam-se linhas contínuas para representar o ramo de soluções estacionárias estáveis e linhas tracejadas mostram um ramo de soluções estacionárias instáveis.

Outra estrutura especial é a isola. Essa é formada por ramos de soluções unidos por LPs separada dos demais ramos de solução. Normalmente, detecta-se uma isola observando um ramo ao variar um segundo parâmetro, conforme ilustra a Figura 2.3.

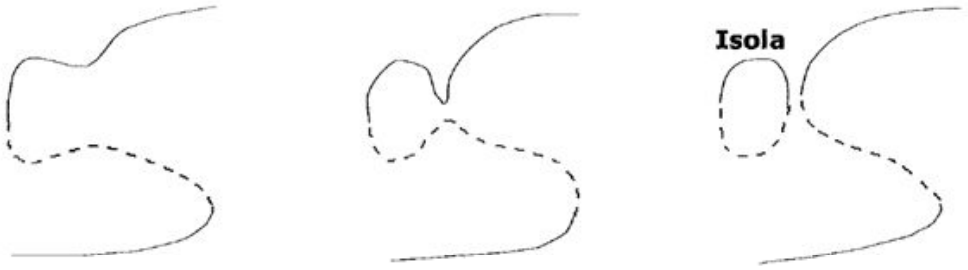


Figura 2.3: Surgimento da isola com a variação de um segundo parâmetro. (OURIQUE, 2000)

Quando um ou mais pares de valores característicos complexos conjugados do Jacobiano cruzam o eixo imaginário do plano complexo, satisfazendo a Equaçao 2.6 a bifurcação é conhecida como Bifurcação de Hopf (HB). Em suma, quando se tem um foco e ele, ao variar um parâmetro, ganha ou perde estabilidade, passando por trajetórias que formam um ciclo limite, isso caracteriza que, naquele ponto, ocorreu uma HB. Fisicamente, os pontos de bifurcação de Hopf apontam o surgimento de oscilações periódicas no sistema (PINTO, 1991).

Os pontos de bifurcação de Hopf podem ser continuados em dois parâmetros, o que permite determinar uma região no espaço de parâmetros onde soluções oscilatórias estão presentes (FREITAS FILHO, 1993). Uma solução periódica de período T satisfaz a condição:

$$\mathbf{x}(t+T,\lambda) = \mathbf{x}(t,\lambda) \tag{2.7}$$

A partir do ponto HB, forma-se um ciclo limite, representado por uma curva fechada em um espaço de fases. A Figura 2.4 apresenta a formação de um ciclo limite após a mudança do parâmetro λ e o ponto de HB localizado na origem (SEYDEL, 2010).

Soluções oscilatórias periódicas apresentam ciclos limites no planos de fases, que podem ser estáveis ou instáveis, definindo assim uma região de atração ou repulsão (MELO, 2001).

Vale ressaltar que, um corte do plano $x_1\lambda$ ou $x_2\lambda$ da Figura 2.4 remete aos gráficos de bifurcação da Figura 2.2, enquanto que, cortes paralelos ao plano x_1x_2

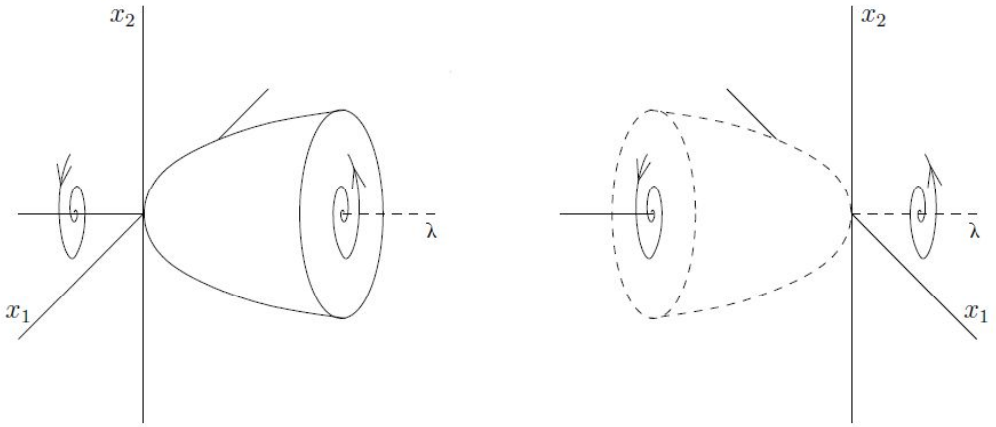


Figura 2.4: Geração de um Ciclo Limite. (DERCOLE e RINALDI, 2011)

são planos de fase para cada valor do parâmetro λ, que representam a Figura 2.1.

2.3 Estabilidade da Solução Periódica

Haja vista que o Mapeamento de Poincaré e os multiplicadores de Floquet, tradicionalmente, estão contidos na metodologia para a análise de estabilidade e identificação de bifurcações periódicas secundárias, eles serão abordados aqui de modo introdutório. Para um estudo mais detalhado,recomenda-se ver Seydel (2010), utilizado nesta demonstração.

2.3.1 Teoria de Floquet

Uma solução periódica é assintoticamente estável se, com o transcorrer do tempo, todas as trajetórias iniciadas em suas vizinhanças convergem para ela. À medida que o tempo passa, a distância entre as trajetórias $\varphi(t, \mathbf{x})$ e $\varphi(t, \mathbf{x}^*)$ é uma função do tempo e pode ser aproximada pela série de Taylor sem os termos de ordem superior:

$$\mathbf{d}(t) = \frac{\partial \varphi(t, \mathbf{x}^*)}{\partial \mathbf{x}} \cdot \mathbf{d}_0 \tag{2.8}$$

na qual, $\mathbf{x}$ representa um vetor de estados do sistema, $\mathbf{x}^*$ é um ponto sobre a órbita periódica no espaço de fases, $\mathbf{d}$ representa a distância entre as trajetórias, $\mathbf{d}_0 = \mathbf{x} - \mathbf{x}^*$ é a perturbação em relação à órbita periódica e $\varphi(t, \mathbf{x}^*)$ é uma trajetória periódica para o sistema:

$$\frac{\partial \varphi(t, \mathbf{x})}{\partial t} = f(\varphi(t, \mathbf{x}), \lambda) \tag{2.9}$$

então, a matriz Φ expressada por:

$$\Phi(T) = \frac{\partial \varphi(t, \mathbf{x}^*)}{\partial \mathbf{x}} \tag{2.10}$$

é responsável para decidir se a perturbação inicial $\mathbf{d}_0$, em relação à órbita periódica, diminui ou aumenta a cada período T, ditando a estabilidade do sistema. Essa matriz é chamada de monodromia. É importante salientar que isso remete a um sistema discreto no qual a cada ponto de análise o tempo evolui T unidades de uma amostragem para a outra. A seguir, é mostrado um resumo do que é apresentado em Seydel (2010).

$$\Phi(t+T) = \Phi(t)\Phi(T) \tag{2.11}$$

então,

$$\Phi(2T) = \Phi^2(t) \tag{2.12}$$

logo,

$$\Phi(kT) = \Phi^k(t) \tag{2.13}$$

se ξ é um vetor característico qualquer de $\Phi(T)$ e $\Lambda(T)$ é o valor característico associado, da álgebra se tem:

$$\Phi(T)\xi = \Lambda(T)\xi \tag{2.14}$$

multiplicando-se ambos os lados da expressão pela esquerda por $\Phi(T)$:

$$\Phi^2(T)\xi = \Lambda(T)\Phi(T)\xi = \Lambda^2(T)\xi \tag{2.15}$$

generalizando,

$$\Phi(kT) = \Phi^k(T)\xi = \Lambda^k(T)\xi = \Lambda(kT)\xi \tag{2.16}$$

então, a solução são funções exponenciais e os valores característicos podem ser expressos por:

$$\Lambda(T) = \exp(\sigma T) \tag{2.17}$$

na qual, $\sigma = \sigma_1 + \sigma_2 i$ são chamados de expoentes de Floquet e os valores característicos da matriz monodromia são conhecidos como multiplicadores de Floquet.

Pode-se dizer então que Φ mapeia o estado da solução em $t + T$ a partir do estado no tempo t. Então, como a órbita periódica é mapeada nela mesma, um dos multiplicadores de Floquet será sempre igual a um (OURIQUE, 2000).

Podem-se utilizar tanto os multiplicadores quanto os expoentes de Floquet para analisar a estabilidade de órbitas periódicas. As regiões de estabilidade para os dois casos são apresentadas na Figura 2.5. Importante ressaltar que, assim como a estabilidade de soluções estacionárias com os valores característicos da matriz Jacobiana, basta um multiplicador ou expoente fora da região de estabilidade para a órbita se tornar instável.

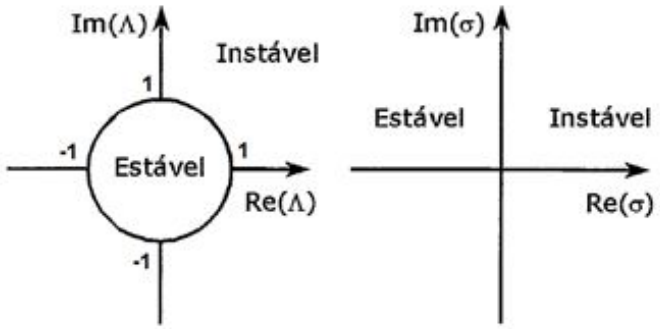


Figura 2.5: Regiões de estabilidade para multiplicadores e expoentes de Floquet. (OURIQUE, 2000)

Em virtude do círculo unitário que delimita a região de estabilidade para os multiplicadores de Floquet, existem três formas de verificar a mudança de estabilidade conforme os multiplicadores entram ou saem do círculo, conforme exposto na Figura 2.6.

Dado que a matriz Φ avalia o desvio entre uma órbita periódica e uma trajetória iniciada nas vizinhanças dela, para multiplicadores de Floquet com módulo menor que um, o desvio diminui com o tempo, caracterizando a órbita periódica como estável. Entretanto, se um dos multiplicadores de Floquet, em módulo, for maior que um, a trajetora se afasta da órbita periódica, caracterizando-a como instável (SEYDEL, 2010).

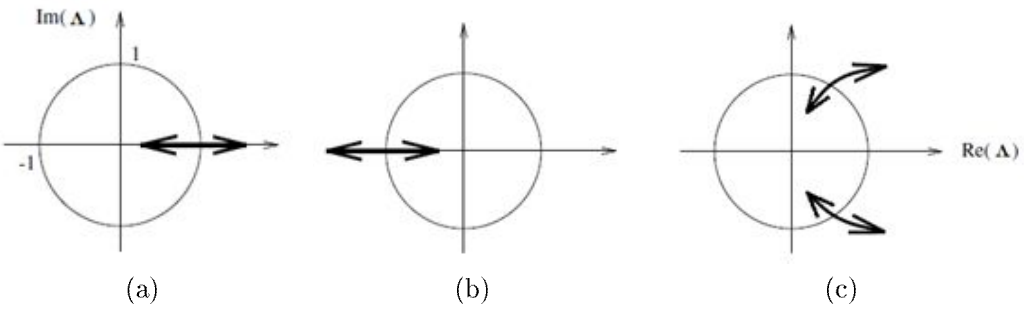


Figura 2.6: Mudança de estabilidade da órbita periódica. a) Em um ponto de bifurcação de solução periódica; b) Ponto de duplicação de período; c) Formação de toróide. (SEYDEL, 2010)

2.3.2 Mapeamento de Poincaré

Para analisar uma trajetória $\mathbf{x}(t)$ contínua é usual aproximá-la por vetores em tempos discretos. Uma técnica de discretização foi desenvolvida por Henri Poincaré que possibilita a redução da dimensão do sistema em um estado. Esta técnica é chamada de Mapeamento de Poincaré, tal que:

$$\mathbf{x}(t_{k+1}) = f(\mathbf{x}(t_k)) \qquad \forall \quad k \in \mathbb{N}^* \tag{2.18}$$

então,

$$\mathbf{x}(t_{k+2}) = f(\mathbf{x}(t_{k+1})) = f(f(\mathbf{x}(t_k))) = f^2(\mathbf{x}(t_k)) \tag{2.19}$$

portanto, a trajetória atravessa uma hipersuperfície de dimensão N - 1, em um espaço de fases com N dimensões. Seja $\mathbf{F}(\mathbf{x})$ o vetor tangente à trajetória e $\mathbf{n}$ o vetor normal à superfície no ponto $\mathbf{x}$, é importante que, nesses pontos de intersecção, o produto escalar satisfaça a seguinte condição:

$$\mathbf{F}^k \cdot \mathbf{n} \neq 0 \tag{2.20}$$

para que a trajetória não passe através da hiper superfície ortogonal ao vetor normal. Dessa forma, cada ponto $\mathbf{P}$ e a própria hiper superfície são chamados de seção de Poincaré. Usando, analogamente, a forma discreta apresentada:

$$\mathbf{P}_{k+1} = \mathbf{F}(\mathbf{P}_k) \tag{2.21}$$

pode-se interpretar cada ponto como imagem de seu predecessor que representa um mapa iterativo, não linear e genérico da hiper superfície, chamado de Mapeamento de Poincaré. Então, para uma órbita periódica simples, o mapeamento pode ser descrito como:

$$\mathbf{P}_S = \mathbf{F}(\mathbf{P}_S) \tag{2.22}$$

na qual as soluções representam pontos fixos $\mathbf{P}_S$ em uma seção de Poincaré. Portanto, a estabilidade está diretamente relacionada ao afastamento ou a aproximação de órbitas iniciadas próximas a pontos $\mathbf{P}_S$ em relação a ele. Ou seja, comparando com a teoria de Floquet, a matriz

$$\frac{\partial \mathbf{F}(\mathbf{P}_S)}{\partial \mathbf{P}} \tag{2.23}$$

é análoga à matriz monodromia, ou seja, se todos os N - 1 valores característicos estiverem dentro do círculo unitário a trajetória $\mathbf{P}_S$ é estável, enquanto que basta um fora para a órbita ser instável.

O Mapeamento de Poincaré é útil no sentido de classificar soluções oscilatórias, tendo em vista que estas podem construir atratores mais complexos, possibilitando um entendimento melhor da dinâmica do sistema (SAVI, 2006). Para esse autor, existem quatro casos padrão para a construção de uma seção de Poincaré, são eles: Estudo de órbitas próximas a órbitas periódicas; plano de fases periódico com forçamento periódico; plano de fases quasi-periódico com forçamento quasi-periódico; e, estrutura de órbitas próxima das órbitas homoclínica ou heteroclínica.

2.3.3 Mudança de Estabilidade da Solução Periódica

É possível representar, também, ramos de pontos fixos $\mathbf{P}_S$ em uma seção de Poincaré e avaliar a estabilidade deles com a alteração do parâmetro. Como já foi abordado, há três formas desses ramos se tornarem estáveis ou instáveis: em pontos de ramificação de solução periódica, pontos de duplicação de período ou com a formação de um toróide.

Alguns exemplos de pontos de ramificação de solução periódica são o ponto limite periódico (PLP) e o ponto de bifurcação limite periódico (PBLP). No PLP há a mudança de estabilidade do ramo periódico, ou seja, duas órbitas periódicas coexistem no espaço de estados, sendo uma estável e outra instável. Enquanto que no PBLP, ao variar um parâmetro, há o surgimento de duas outras trajetórias periódicas, ou seja, surgem duas órbitas, interna e externa em relação à órbita original, no espaço de estados.

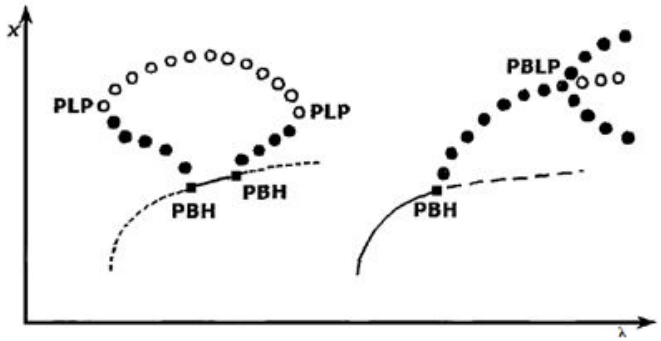


Figura 2.7: Diagrama de bifurcação para pontos de ramificação de soluções periódicas. (OURIQUE, 2000)

Na Figura 2.7, as órbitas estáveis são representadas por bolas fechadas, órbitas instáveis por bolas abertas e pontos de bifurcação de Hopf por quadrados preenchidos, conforme convencionado.

Quando há duplicação do período a estabilidade da solução de período simples é alterada. A estabilidade da solução de período duplo, que envolve a solução de período simples, é a mesma que tinha a ramificação de período simples antes do

ponto de duplicação de período (PDP) (OURIQUE, 2000), conforme ilustra a Figura 2.8.

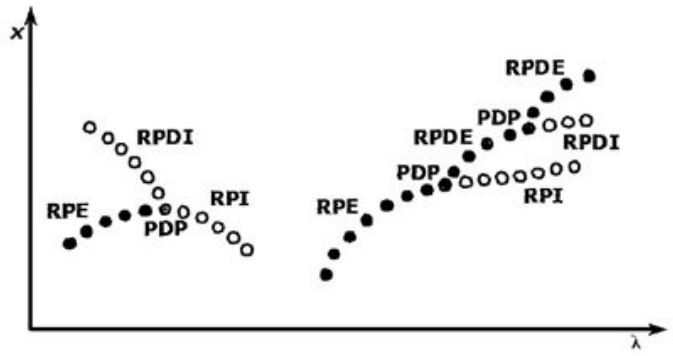


Figura 2.8: Diagrama de bifurcação para pontos de duplicação de período. (OURIQUE, 2000)

Na Figura 2.8, RPE significa ramo periódico estável, RPI ramo periódico instável, RPDE ramo de período duplo estável e RPDI ramo de período duplo instável. Após uma duplicação de períodos, outra duplicação é esperada (CRAWFORD e OMOHUNDRO, 1984).

O toróide é uma superfície de revolução gerada pela rotação completa de uma circunferência em torno de um eixo que não seja secante a ela. Portanto, um dos planos de corte de um toróide é a própria circunferência. Supondo que esse plano seja a seção de Poincaré, a trajetória de soluções dinâmicas do sistema seria uma curva na superfície do toróide, formando a circunferência na seção de Poincaré conforme a Figura 2.9.

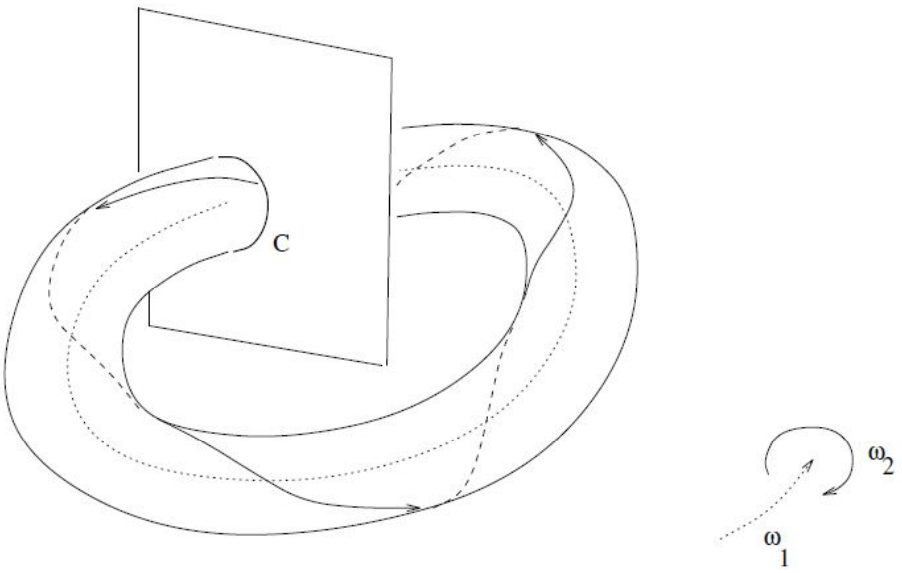


Figura 2.9: Seção de Poincaré de um toróide. (SEYDEL, 2010)

Dessa forma, existem duas frequências ω_1 e ω_2. Uma em torno do eixo de revo-

lução e outra em torno do centro da circunferência. Na seção de Poincaré o centro da circunferência representa o ponto $\mathbf{P}_S$, solução periódica de origem. Quando a razão das frequências é irracional, a trajetória iniciada sobre a curva fechada na seção de Poincaré não retorna à posição original e o movimento é classificado como quasi-periódico.

2.4 Atrator Estranho ou Caótico

O Caos não é uma solução estacionária nem periódica, que resulta em uma trajetória irregular com relação ao espaço e ao tempo, havendo sensibilidade do comportamento dinâmico do sistema a perturbações nas condições iniciais.

Muitas rotas para o caos foram observadas, dentre elas: via transição da quasi-periodicidade; via cascata de duplicação de períodos; e via intermitência (OURIQUE, 2000). Uma forma de verificar a existência do caos é com a utilização dos expoentes de Lyapunov.

Em relação a uma trajetória de referência, a divergência entre trajetórias próximas a ela pode ser analisada (SAVI, 2006). A vizinhança em relação à trajetória de referência é definida como uma hiperesfera de raio d_0 e de dimensão associada à quantidade de variáveis de estado do sistema.

Os expoentes de Lyapunov avaliam o desvio exponencial no tempo da trajetória em relação à trajetória de referência. Dessa forma, avaliam a evolução no tempo dos eixos de uma esfera suficientemente pequena de estados do sistema dinâmico. Assim, a variação do diâmetro da esfera pode ser expressa por:

$$\mathbf{d}(t) = \mathbf{d}_0 b^{\mu t} \tag{2.24}$$

na qual, b é uma base de referência. Portanto, os expoentes de Lyapunov μ são definidos por:

$$\mu_i = \lim_{t\to\infty} \frac{1}{t} \log_b \frac{(d_i(t))}{d_{0i}} \tag{2.25}$$

sendo i um índice relativo a cada variável do sistema e b usualmente igual ao número de Euler.

Portanto, se um expoente for positivo a trajetória diverge exponencialmente da órbita de referência, caracterizando o caos. Nesse sentido, quando o maior valor dos expoentes de Lyapunov for nulo, tem-se um ciclo limite e quando todos forem negativos, isso está associado a um estado estacionário.

Quando o sistema dinâmico é descrito por um modelo matemático que permite a sua linearização em torno de uma determinada trajetória, os expoentes de Lyapunov podem ser calculados a partir do algoritmo proposto por Wolf et al. (1985). Savi

(2006) apresenta o algoritmo clássico para o computo dos expoentes de Lyapunov que foi adotado neste trabalho.

2.5 Análise de Sistemas Dinâmicos da Engenharia Química

Em Ourique (2000), são apresentados alguns trabalhos que, na Engenharia Química, possuem sistemas dinâmicos que geram comportamentos complexos. Esses são os que envolvem reações químicas, sistemas biológicos, processos de separação e sistemas de escoamento e mistura.

Com a finalidade de facilitar o entendimento do leitor com relação ao tema deste trabalho na Engenharia Química, a Tabela 2.1 apresenta, resumidamente, alguns trabalhos e os conceitos abordados.

Tabela 2.1: Alguns estudos de sistemas dinâmicos não lineares na Engenharia Química

Ano	Autor	Conceito
1953	van Heerden	Dinâmica de reatores do tipo CSTR com reação exotérmica de primeira ordem (reator clássico).
1955	Bioulus e Amundson.	Caracterização do número de estados estacionários e da estabilidade das soluções obtidas nos reatores clássicos.
1974	Uppal *et al.*	Introdução da Teoria da Bifurcação em problemas da Engenharia Química.
1980	Golubitsky e Keyfitz	Mapeamento de todos os padrões de multiplicidade de soluções estacionárias no reator clássico.
1983	Balakotaiah e Luss	Dinâmica de reatores do tipo CSTR com três reações em paralelo.
1983	Smith *et al.*	Existência de soluções caóticas em reatores químicos.
1990	Pinto *et al.*	Comportamento dinâmico de reatores de polimerização em suspensão.
1991	Pinto	Análise do Comportamento Dinâmico de Sistemas de Polimerização pela Teoria de Bifurcações.
1994	Freitas F° *et al.*	Comportamento dinâmico de reatores de polimerização em massa.
1997	Nele e Pinto	Comportamento dinâmico de reatores de polimerização em solução.
2000	Ourique	Métodos alternativos para análise dinâmica em processos químicos.
2000	Melo	Dinâmica e Estabilidade de Reatores Tubulares de Polimerização com Reciclo.
2002	Ourique, Biscaia e Pinto	O uso do algoritmo de otimizaçao do enxame de partículas para análise dinâmica em processos químicos.
2003	Melo, Biscaia e Pinto	Comportamento da bifurcação de reatores contínuos de radicais livres em soluçao do tipo loop.
2007	Ramminger e Secchi	Integraçao do *software* AUTO com o simulador de processos EMSO.
2008	Melo e Pinto	Introdução à Modelagem Matemática e Dinâmica Não-Linear de Processos Químicos.
2009	Salau *et al.*	Controle multivariável baseado em análise de bifurcaçao de reator de polimerização em leito fluidizado.
2011	Rodrigues	Comportamento caótico em reatores contínuos de polimerização em solução via radicais livres.
2012	Oechsler	Análise de bifurcações de problemas de micromistura em reatores de polimerização em solução.
2013	Rosa	Análise dinâmica e de estabilidade de reatores tubulares de polimerização de propeno do tipo loop.

2.6 Metodologias Usualmente Aplicadas

Os algoritmos clássicos usados para o estudo do comportamento dinâmico de sistemas dinâmicos não lineares usam como ferramenta básica diagramas de bifurcação e os planos de fases. A construção dos diagramas de bifurcação é realizada pela técnica de continuação paramétrica ao longo do comprimento do arco do ramo de soluções (SEYDEL, 2010).

Vários pacotes computacionais propostos usam como base essa técnica. O mais usado para a construção dos diagramas de bifurcação atualmente é o AUTO (DOEDEL E HEINEMANN, 1983, DOEDEL et al., 1994). Capaz de fazer a continuação de soluções, desde soluções estacionárias até o caos.

A estratégia adotada por todos os algoritmos clássicos é basicamente a mesma. Inicialmente é determinada uma solução estacionária do sistema e, a partir dessa solução estacionária, é usada a técnica de continuação ao longo do comprimento do arco (ou pseudo comprimento) e é adquirido o ramo de soluções estacionárias. Pontos como LP, BP e HB são obtidos pelo acompanhamento dos valores característicos da matriz Jacobiana.

Encontrado um HB, este é usado como ponto de partida para a continuação dos ramos de soluções periódicas. O acompanhamento dos valores característicos da matriz monodromia detecta pontos de bifurcação como PLP, PBLP e PDP. Cada um desses pontos pode ser usado como estimativa para a continuação dos ramos que surgem.

Logo, segundo Ourique (2000), a análise das soluções de sistemas dinâmicos não lineares é um longo e exaustivo processo e frequentemente não conclusivo, sem dizer que depende do conhecimento de pelo menos uma solução estacionária e só é possível a continuação de um parâmetro por vez.

Sabendo disso, Ourique (2000) propôs um algoritmo cujo objetivo é determinar soluções especiais para a análise de bifurcações de sistemas não lineares. Em sua metodologia só é preciso conhecer o modelo matemático e a faixa de valores que cada parâmetro pode assumir. Resumidamente, é escolhido um conjunto de pontos do espaço paramétrico de ordem M, sendo esse a quantidade de parâmetros não constantes. Para cada ponto desse conjunto é realizada uma simulação do problema.

A resposta dinâmica é dividida em duas partes e a segunda é analisada para garantir que um comportamento dinâmico está estabelecido. Se a diferença entre os pontos for desprezível a solução é dita estacionária, se forem detectados picos, esses são armazenados. Uma vez comparados é possível dizer se a solução é periódica com um ou dois períodos de oscilação ou se a solução é assintoticamente estável. Caso não haja repetição dos valores dos picos a solução é classificada como caótica.

Cada um desses tipos recebe um valor para caracterizar a resposta. Estacionária

é 1, periódica de período T é 2, periódica de período 2T é 3 e caótica é 4. Assim é definida uma função objetivo que é usada no método de otimização não determinístico enxame de partículas (PSO) para avaliar dentre os conjuntos de pontos de parâmetros qual apresenta simulação com o comportamento desejado. Criando um mapa no espaço paramétrico do comportamento do sistema.

Outros trabalhos, Rodrigues (2011), Oechsler (2012) e Rosa (2013), usam uma estratégia parecida, porém apenas escolhem pontos aleatórios no espaço de parâmetros, sem associar a um método de otimização, ou seja, através da busca exaustiva. Nesses estudos são escolhidos muitos pontos até se considerar que o espaço foi bem preenchido, mas mesmo assim não é possível dizer que todas as soluções foram encontradas. Em Rodrigues (2011) são escolhidos um milhão de pontos no espaço paramétrico e apenas foram encontrados vinte e três casos de interesse.

É importante ressaltar que, analisar o comportamento dinâmico do sistema com relação à resposta de uma simulação pode levar a falhas na detecção das soluções dinâmicas dependendo do passo de integração, da quantidade de picos armazenados e do tempo total escolhido. Ainda, como cada um destes métodos é não determinístico, é necessário que o procedimento seja repetido inúmeras vezes para comparar o mapeamento obtido em termos probabilísticos.

Freire et al. (2005), fazem uma abordagem diferente. A metodologia aplicada por esses autores consiste em realizar a continuação das curvas das bifurcações globais, permitindo aprimorar os resultados obtidos pelo uso das bifurcações locais e detectar novas conexões homoclínicas. Porém, os gráficos mapeiam os pontos limites e de bifurcação sem apresentar qual o tipo de comportamento que o sistema possui e nem o valor dos parâmetros naquela posição sobre as curvas homoclínicas.

Já Mallory e Van Gorder (2015) utilizam o método Métodos Competitivos para detectar regimes caóticos para um conjunto de parâmetros. Dessa forma, é possível delimitar, no espaço de estados, atratores estranhos. Entretanto, é necessário aplicar o procedimento para cada conjunto de parâmetros que se deseja investigar. Logo, quanto mais parâmetros o sistema possuir, maior será a quantidade de combinações possíveis para as avaliações, implicando em mais simulações para uma análise completa.

Na busca por estudos que tratam de análise de sistemas dinâmicos não lineares foram utilizados termos indexados ("multiparametric", "bifurcation", "analysis" e "systems") em base de dados tais como; Scopus, Science Direct e ASME nos últimos cinco anos. Visando reduzir a limitação da estratégia de busca, visitaram-se também as referências dos artigos encontrados e as produções científicas de autores que estudam o tema. Porém, encontrou-se que a metodologia clássica tem sido a mais utilizada.

Haja vista os desafios apresentados nas metodologias anteriores, uma nova estra-

tégia é proposta neste estudo. Essa, associa um método de otimização determinístico em processamento paralelo para detectar a diferença entre as soluções e os principais comportamentos dinâmicos, estacionário, periódico e caótico, de sistemas não lineares. Para detectar cada um deles é feito o cômputo dos expoentes de Lyapunov, segundo Savi (2006), e observado o maior valor deles.

Para explorar o espaço de parâmetros, o algoritmo DIRECT (JONES *et al*, 1993) foi adotado, explicado no capítulo seguinte, juntamente com a estratégia adotada para selecionar os possíveis pontos ótimos. Para a visualização do espaço mapeado, foi usado o software livre ParaView© que possui uma extensa biblioteca de ferramentas de visualização, normalmente utilizado para trabalhos de fluidodinâmica computacional.

Capítulo 3

Metodologia

Neste capítulo é apresentada uma proposta alternativa para a busca direta de um mapa de soluções no espaço de parâmetros. O algoritmo é composto pelas etapas de definição do espaço paramétrico, divisão do espaço em subespaços, detecção dos subespaços de interesse, divisão dos subespaços e assim sucessivamente até um critério de parada preestabelecido.

No ponto central de cada subespaço, é feita a simulação do sistema e caracterizada a resposta, que representa o comportamento do sistema para todo o subespaço. Portanto, quanto mais bem dividida for a sub-região, melhor será a qualidade do mapeamento (JONES *et al*, 1993).

Foi adotado o algoritmo de otimização DIRECT (JONES *et al*, 1993), descrito a seguir, pois ele faz uso dessa estratégia para buscar a solução desejada. Também é importante observar que cada ponto central do subespaço é independente do outro, cada um representa um conjunto de parâmetros para a simulação do sistema. Dessa forma, pode-se associar o processamento paralelo para diminuir o tempo computacional de análise.

Para criar o mapa de soluções no espaço de parâmetros, foi escolhido buscar por soluções diferentes que fossem vizinhas. Caso uma solução periódica fosse encontrada ao lado de um estado estacionário, por exemplo, essas duas regiões seriam escolhidas como possivelmente ótimas e seriam, posteriormente, subdivididas.

A Figura 3.1 apresenta um fluxograma que explica o algoritmo adotado na metodologia. Os próximos itens mostram as etapas detalhadas do algoritmo proposto.

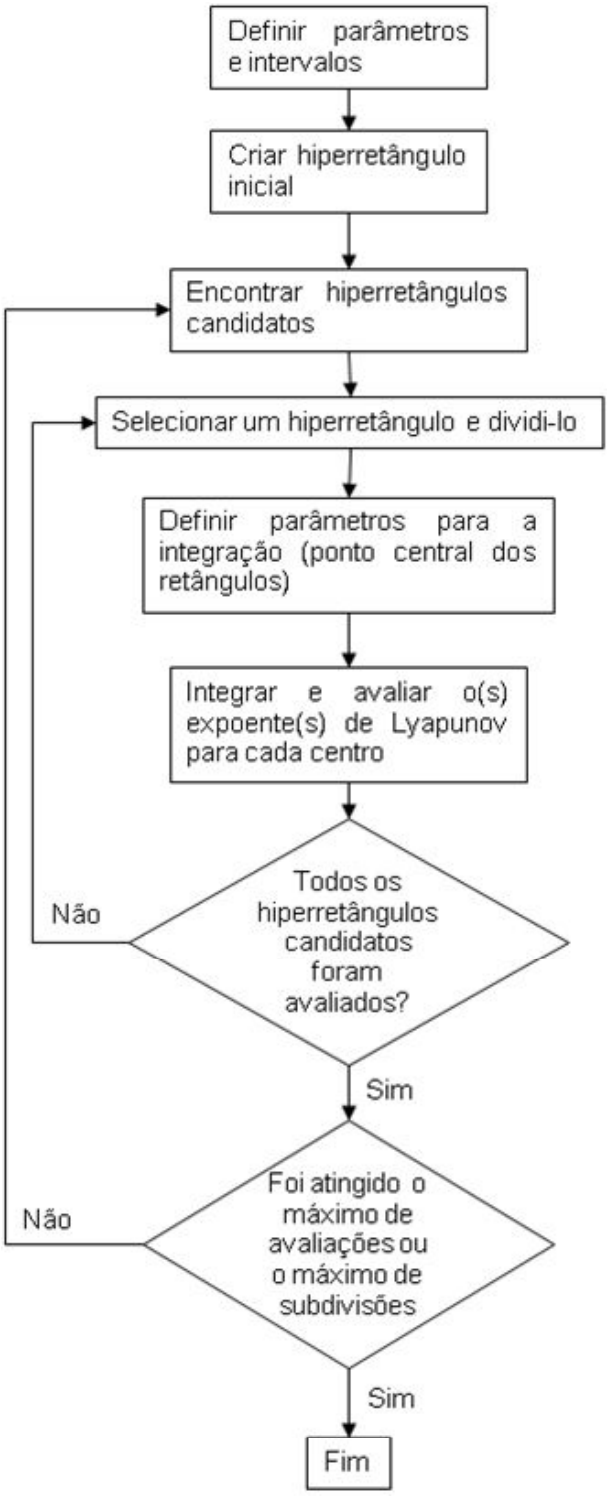


Figura 3.1: Algoritmo proposto para a busca por soluções especiais.

3.1 Caracterização das Soluções

A caracterização das soluções obtidas, como já foi dito, é feita de acordo com o sinal do maior expoente de Lyapunov. Se o sistema tiver N variáveis de estado, também terá N expoentes de Lyapunov associados. O sinal dos valores dos expoentes está associado à convergência ou divergência das trajetórias. Por isso, foi imple-

mentado o cômputo dos expoentes nesta metodologia para caracterizar a solução obtida.

De acordo com essa propriedade, se dentre todos os expoentes houver pelo menos um positivo, a trajetória diverge exponencialmente e evidencia uma solução caótica. Porém, se a solução dinâmica for periódica, isso indica que o maior expoente de Lyapunov, o expoente dominante do sistema, é nulo. Enquanto que se todos forem negativos, as trajetórias convergem exponencialmente para um ponto fixo caracterizando a solução como um estado estacionário (SAVI, 2006).

Por isso, para cada conjunto de parâmetros, pontos do espaço paramétrico, foi realizada a simulação do sistema e o cálculo dos expoentes de Lyapunov para caracterizar a solução, segundo o algoritmo proposto por Wolf *et al.* (1985), e apresentado por Savi (2006) em forma de pseudo-código. Esse pseudo-código foi adaptado e incorporado neste estudo.

Com relação à caracterização feita pela detecção de picos, uma solução é dita estacionária estável quando, por integração numérica, não se percebe variação significativa dos valores atribuídos aos estados com o passar do tempo. Isso pode ocorrer quando não são observados picos ou se a amplitude de oscilação é desprezível até o final da análise da resposta obtida na integração numérica.

Quando são detectados picos, esses são armazenados para que a reincidência seja verificada. Se houver repetição desses picos, dentro de uma dada tolerância, a solução é classificada como periódica. Caso contrário, a solução é dita caótica.

É atribuído a cada centro de um subespaço um valor que varia entre 1, 2 e 3 para as soluções estacionárias, periódicas e caóticas, respectivamente. Dessa forma, é possível comparar os comportamentos dinâmicos para cada ponto do espaço de parâmetros.

3.2 Função Objetivo

Caracterizadas as soluções como estacionária (1), ciclo limite (2) ou caos (3), é possível aplicar um algoritmo de otimização, tendo como variáveis os parâmetros do sistema com a intenção de explorar ao máximo e melhorar a resolução do mapa de soluções gerado. Direcionando a busca para encontrar o contorno das regiões do espaço paramétrico que contém cada tipo de soluções estacionária e dinâmica.

Comparando o comportamento da resposta da integração numérica entre os hiperretângulos vizinhos, usando o centro de cada hiperretângulo como parâmetros para a integração, é possível encontrar comportamentos distintos e delimitar melhor a interface entre cada região, que ó o objetivo para mapear o espaço de parâmetros.

Então, para encontrar os hiperretângulos candidatos a função objetivo adotada aponta os casos possivelmente ótimos sempre que, pelo menos, duas soluções vizinhas

apresentem comportamentos diferentes entre si. Dessa forma, assim que uma solução diferente das soluções imediatamente próximas for detectada, é feita a subdivisão dessas regiões, melhorando a resolução do mapa localmente. Neste caso, não há uma expressão para a função objetivo, pois esta faz apenas uma comparação entre o comportamento das regiões vizinhas para buscar as fronteiras.

É feita também uma ponderação entre o tamanho dos subespaços e o valor da função objetivo para evitar que existam regiões mal exploradas, a despeito de apresentar apenas um tipo de comportamento nas redondezas. Impedindo a omissão de um possível comportamento dinâmico diferente em um espaço mais estreito.

Apesar da função objetivo ser descontínua, é definida em todo o domínio, que é o próprio espaço de parâmetros investigado.

3.3 DIRECT

O algoritmo DIRECT foi proposto por Jones *et al.* (1993) e é um algoritmo determinístico para encontrar o mínimo global de uma função objetivo de múltiplas variáveis sem restrições.

Uma vantagem do método é a forma como ele pondera entre busca global e local. Uma vez encontradas as regiões de interesse dentre todo o espaço, essas são exploradas até que se atinja o critério de parada. Outra vantagem é que por ser determinístico, não é preciso sobrepor sucessivas avaliações do método como acontece com os métodos não determinísticos.

Primeiramente, o domínio é normalizado, transformando-o no hipercubo unitário. Em seguida a função objetivo é avaliada no centro do hipercubo. O próximo passo avalia a função objetivo em um terço da dimensão do centro do hipercubo em todas as direções. Deixando os melhores valores da função para os maiores espaços.

Na Figura 3.2 a região cinza representa o menor valor da função objetivo dentre os novos centros adicionados. É possível observar que o primeiro ponto central pode ser aproveitado e apenas foram adicionados novos centros vizinhos. Isso ocorrerá para as próximas subdivisões também.

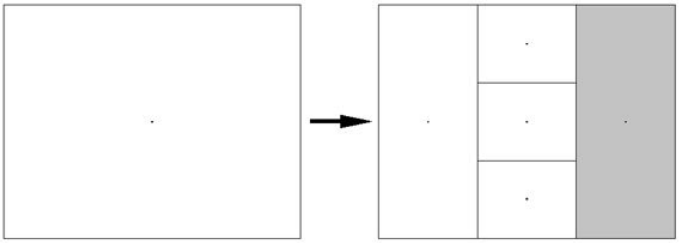


Figura 3.2: Inicialização do algoritmo para um caso de duas dimensões.

Um hiperretângulo é considerado como possivelmente ótimo se possuir a menor

distância entre o centro e o vértice e possuir o menor valor da função objetivo. Para garantir que o espaço não fique mal explorado, é considerado como possivelmente ótimo aquele que foi pouco dividido em comparação ao que sofreu mais divisões. Esse valor pode ser ajustado para alterar o equilíbrio entre a busca local e global.

O Algoritmo proposto pode ser descrito pelos seguintes passos:

1. Normalizar o espaço de busca para um hipercubo unitário. Fazer $\mathbf{C}_1$ o centro do hipercubo e avaliar f($\mathbf{C}_1$). Atribuir fmin = f($\mathbf{C}_1$), $m = 1$ e $t = 0$ (contador de iterações);

2. Identificar o conjunto S dos hiperretângulos potencialmente ótimos;

3. Selecionar qualquer hiperretângulo $j \in S$;

4. Dividir o hiperretângulo j em 3. Atualizar f_min e fazer $m = m + \Delta m$, no qual Δm é o número de novos pontos avaliados;

5. Fazer $S = S - \{j\}$, se $S \neq \{\}$ ir para o passo 3;

6. Atribuir $t = t + 1$. Se t for igual ao número máximo de iterações, então pare. Caso contrário, voltar ao passo 2.

Destaca-se que o DIRECT tem como objetivo encontrar mínimos, não sendo esse o objetivo deste estudo, neste sentido, foi necessário adaptar o método para encontrar fronteiras. Foi adotado, também, um número máximo de subdivisões para evitar que sejam executadas partições do espaço que sejam fisicamente desprezíveis.

3.4 Hierarquia dos Hiperrretângulos

Para garantir o funcionamento do algoritmo deste trabalho, foi criada uma hierarquia entre todos os hiperretângulos e armazenados em uma estrutura de dados. Isso é importante para comparar quais são considerados vizinhos para a função objetivo.

A Figura 3.3, apresenta um exemplo de um espaço paramétrico bidimensional para sucessivas etapas do algoritmo DIRECT com valores que caracterizam os possíveis tipos de solução.

Pode-se observar que na última iteração, na etapa de Divisão e Caracterização, a região hachurada passa a ter dois tipos de solução do sistema em comparação com a iteração anterior. Com isso é possível estender essa concepção e entender que quanto mais dividido o espaço, melhor será a resolução e a acurácia do mapa.

Ainda é preciso definir quais subespaços são considerados vizinhos. Para isso, foi adotada a ideia de nível hierárquico na qual um hiperretângulo (pai) gera três

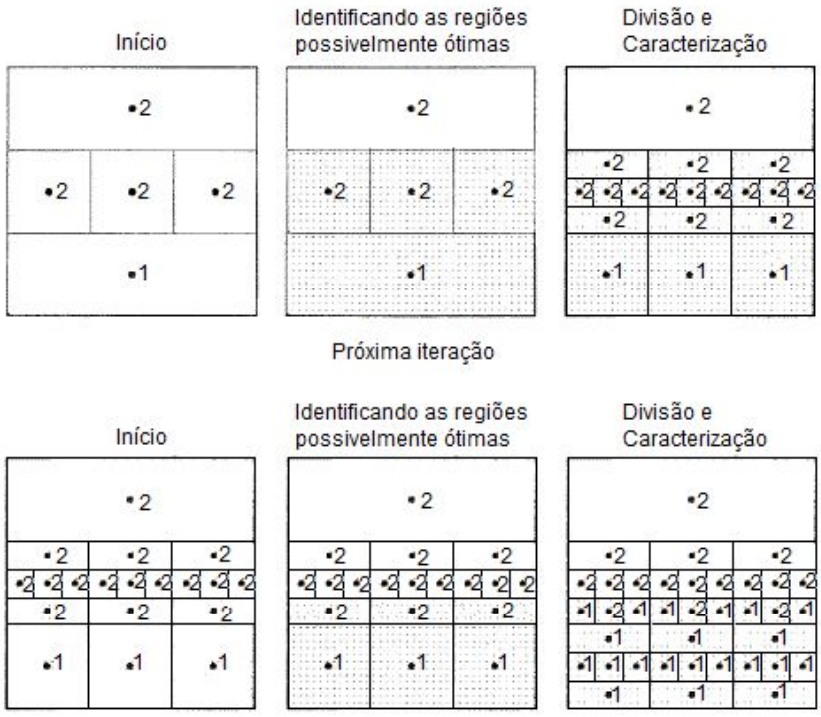


Figura 3.3: Duas iterações do DIRECT modificado.

subespaços (filhos). Entretanto, nem todos os filhos são vizinhos e quando mais dimensões tiver o espaço de parâmetros, mais evidente isso se torna.

Existe o cenário que quando dois pais são vizinhos, possuem dois ou mais vértices em comum, seus filhos também podem ser vizinhos. Por isso, é importante sempre acessar um nível hierárquico superior para buscar por outras possibilidades de vizinhanças que não seus irmãos. A Figura 3.4 apresenta como o algoritmo aborda essa característica.

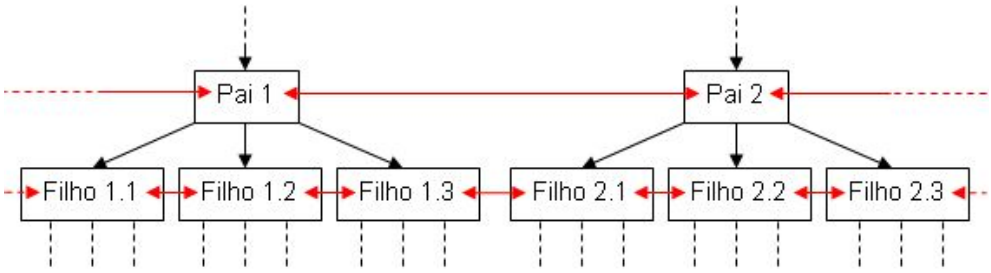


Figura 3.4: Nível hierárquico (setas verticais) e vizinhanças dos hiperretângulos (setas horizontais).

3.5 Processamento Paralelo

Como a tarefa de integração e caracterização da solução é independente para cada centro de um hiperretângulo, foi implementada uma técnica de paralelismo, na

qual um processador mestre (PM) coordena a divisão do espaço e a análise das regiões possivelmente ótimas e atribui tarefas para outros processadores escravos (PE). Essas tarefas consistem em integrar o sistema, calcular os expoentes de Lyapunov e devolver como resposta a caracterização da solução.

À medida que toda a lista de hiperretângulos possivelmente ótimos já foi analisada, o processador mestre gera uma nova lista, divide os candidatos e envia as coordenadas para os escravos repetindo o ciclo até o momento que se atinge o critério de parada.

Para analisar o desempenho da aplicação do processamento paralelo foram adotadas as métricas do *Speedup* ($Sp(N)$) e a eficiência ($Ef(N)$, em porcentagem, em função do número de processamento. O *Speedup* é uma medida do grau de desempenho, medindo a razão entre o tempo de execussão sequencial $T(1)$ e o tempo de execussão com N núcleos de processamento $T(N)$, defido por:

$$Sp(N) = \frac{T(1)}{T(N)} \tag{3.1}$$

A eficiência é uma medida do grau de aproveitamento dos recursos computacionais. Dada pela razão entre o *Speedup* e os recursos computacionais disponíveis segundo:

$$Ef(N) = \frac{Sp(N)100}{N} \tag{3.2}$$

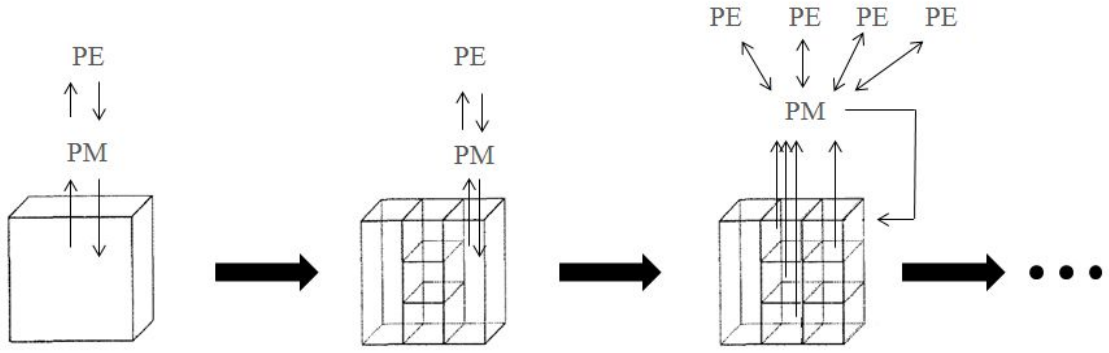


Figura 3.5: Esquema simplificado do processamento paralelo.

Como não há dependência alguma entre os centros, o problema é bem escalável. Ou seja, quanto mais processadores, menor será o tempo para finalizar o mapa. Isso ocorre até um determinado momento, quando as atividades de comunicação do processador mestre passam a limitar o desempenho.

Para coordenar o processamento paralelo foram adotadas *threads*, fluxos de execução dentro de um processador (TANENBAUM, 2009). Um *thread* principal determina os retângulos possivelmente ótimos e organiza uma fila. Depois, distribui para

outras *threads* realizarem a integração e caracterização. Quando essa fila termina, a *thread* principal faz uma nova avaliação dos hiperretângulos possivelmente ótimos.

Cada *thread* avaliador possui sua própria fila, uma fila secundária, para reduzir o tempo ocioso. Assim, quando ele termina de avaliar o centro de um hiperretângulo, ele já escreve direto na memória compartilhada o resultado. Conforme cada *thread* avaliador executa a sua tarefa, o *thread* principal distribui um novo centro para ser avaliado, sendo adicionado no final da lista secundária.

3.6 Casos Estudados

A metodologia foi aplicada em alguns sistemas dinâmicos não lineares. O clássico sistema de Lorenz, o modelo de um CSTR exotérmico clássico e para um reator biológico.

No sistema de Lorenz foram usadas as seguintes condições para o estudo:

$$\begin{cases} \dfrac{dx}{dt} = s(-x + y) \\ \dfrac{dy}{dt} = rx - y - xz \\ \dfrac{dz}{dt} = -bz + xy \end{cases} \tag{3.3}$$

Com $(x_0, y_0, z_0) = (1, 1, 1)$, $s \in [0, 30]$, $b \in [0, 8]$ e $r \in [0, 300]$

No modelo de um CSTR exotérmico clássico, apresentado por van Heerden (1953) e estudado posteriormente por Uppal *et al.* (1974) para avaliar a sensibilidade à variação de parâmetros, quando foram delimitadas regiões de estabilidade de estados estacionários, existência de soluções periódicas e outros fenômenos. A forma adimensional do sistema é definida por:

$$\begin{cases} \dfrac{dy}{d\tau} = -y + Da(1 - y) \exp\left(\dfrac{\theta}{1 + \frac{\theta}{\gamma}}\right) \\ \dfrac{d\theta}{d\tau} = -\theta + Da(1 - y)B \exp\left(\dfrac{\theta}{1 + \frac{\theta}{\gamma}}\right) - \beta(\theta - \theta_c) \end{cases} \tag{3.4}$$

O sistema de equações acima descreve uma reação exotérmica irreversível de primeira ordem que se desenvolve em um reator tipo tanque contínuo de mistura perfeita (CSTR) e é conhecido como sistema clássico.

Com $(y_0, \theta_0) = (0,3; 0,4)$, $B \in [0, 25]$, $\beta \in [0, 5]$, $Da \in [0; 0,3]$, $\gamma = 40$ e $\theta_c = 0$

Um modelo de reator CSTR biologico isotérmico formado por equações de balanço de células e substrato, na forma adimensional, a equacao pode ser escrita, desenvolvido por Agrawal *et al.* (1982), como:

$$\begin{cases} \dfrac{dx_1}{dt} = -x_1 + x_1 Da \dfrac{(1+\alpha)(1-x_2)}{1+\alpha-x_2} \\ \dfrac{dx_2}{dt} = -x_2 + x_1 Da \dfrac{(1+\alpha)(1-x_2)(1-x_2)}{(1+\alpha-x_2)(1+\beta-x_2)} \end{cases} \tag{3.5}$$

Este sistema representa uma cultura pura que se desenvolve em um reator contínuo de volume constante e mistura perfeita, alimentado por meio nutriente estéril. O substrato é o fator limitante do crescimento das celulas.

Com $(x_{1,0}, x_{2,0}) = (0,3,\ 0,4)$, $\alpha \in [0,1]$, $\beta \in [0,1]$ e $Da \in [0,2]$

Capítulo 4

Resultados e Discussões

No presente capítulo são apresentados os resultados das simulações dos três casos estudados. As seções apresentam, para cada caso, o mapa do espaço paramétrico desenvolvido e a simulação dinâmica para alguns pontos selecionados pertencentes às regiões encontradas, com a finalidade de corroborar os resultados. Ainda, foram comparados os resultados obtidos entre a metodologia usando os expoentes de Lyapunov e a caracterização por picos, para evidenciar a diferença entre os dois com relação ao desempenho e a resolução.

4.1 Resultados

É importante ressaltar que o usuário define os critérios de tolerância, e que esses afetam a caracterização das respostas. A tolerância absoluta e relativa do integrador (DASSL) foram de 10^{-7} e a tolerância dos expoentes de Lyapunov foi de 0,01 com 350 divisões, ou seja, o método analisa a dispersão da resposta, ortonormaliza e integra novamente e repete o processo 350 vezes. Já a detecção por picos possui uma tolerância de 10^{-8} com intervalos de 0,01, exceto o caso do sistema de Lorenz que o intervalo foi de 0,001. O tempo final de simulação foi de 80, 80 e 100 unidades de tempo para os casos 1, 2 e 3, respectivamente. Sendo que a análise só era feita na segunda metade de cada intervalo de tempo para eliminar a dinâmica inicial. O algoritmo foi implementado na linguagem *Delphi*.

A Tabela 4.2 apresenta o tempo computacional em processamento sequancial necessário para realizar todo o processo de acordo com os critérios estabelecidos nos subitens seguintes. A configuração do computador utilizado para as análises é Intel Core i7 dual core, 2.7 GHz e 12GB de RAM. Para testar o processamento paralelo, para cada núcleo de processamento físico desta máquina, foram criados três núcleos de processamento lógicos, totalizando seis núcleos de processamento.

Para todos os gráficos apresentados como produto deste trabalho, utiliza-se a cor azul para representar a solução estacionária, a cinza apresenta a solução periódica,

Tabela 4.1: Tempo aproximado para realizar 100.000 avaliações

Metodologia	Caso 1	Caso 2	Caso 3
Lyapunov	3h	1h	48min
Picos	1h 50min	5,5min	9,8min

já a cor vermelha, a solução caótica.

4.1.1 Caso 1 - Sistema de Lorenz

Para as equações de Lorenz são encontrados todos os tipos de comportamento dinâmico que se propõe determinar pelo algoritmo. Os resultados são apresentados nas Figuras 4.1 a 4.3, para a busca de regiões onde há comportamento diferente entre os retângulos vizinhos. Vale ressaltar que o gráfico gerado pela caracterização feita pela detecçao de picos e pelos expoentes de Lyapunov foram praticamente iguais, por isso apenas os gráficos do segundo caso serão apresentados.

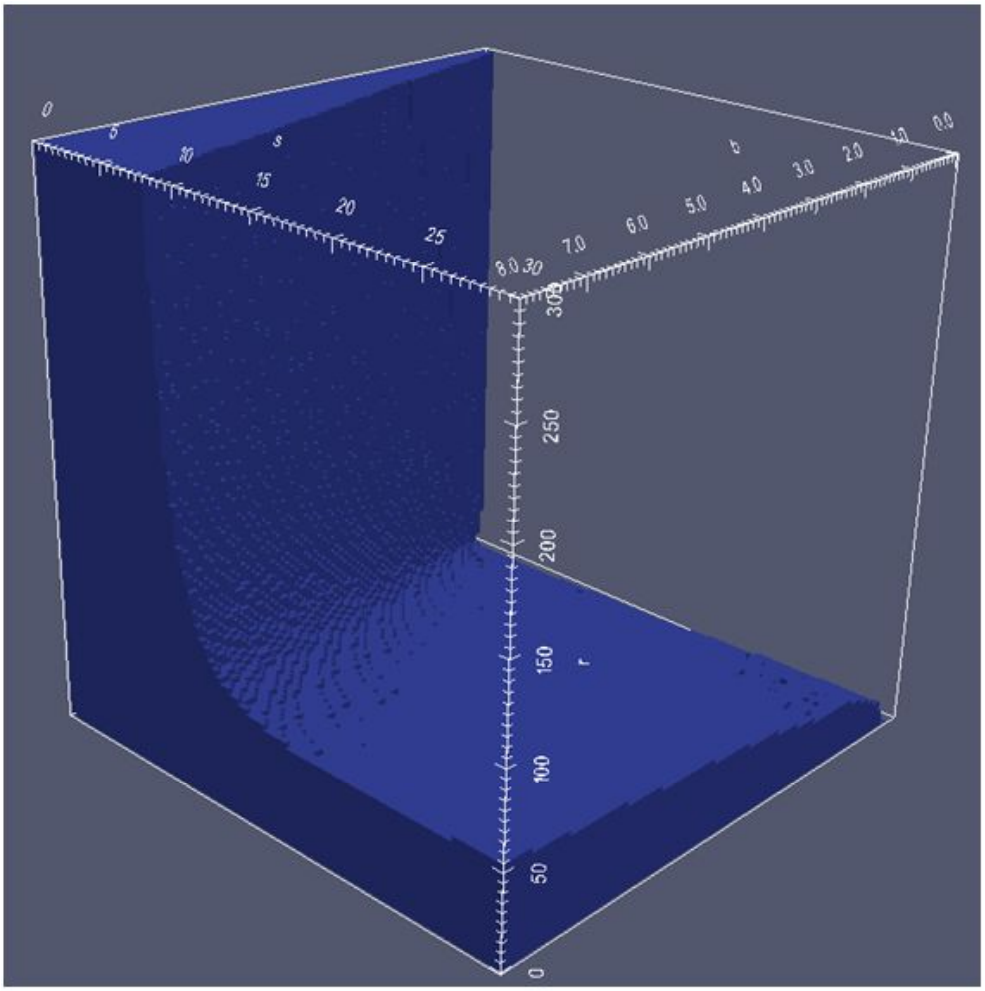


Figura 4.1: Soluções estacionárias do sistema de Lorenz usando o expoente de Lyapunov.

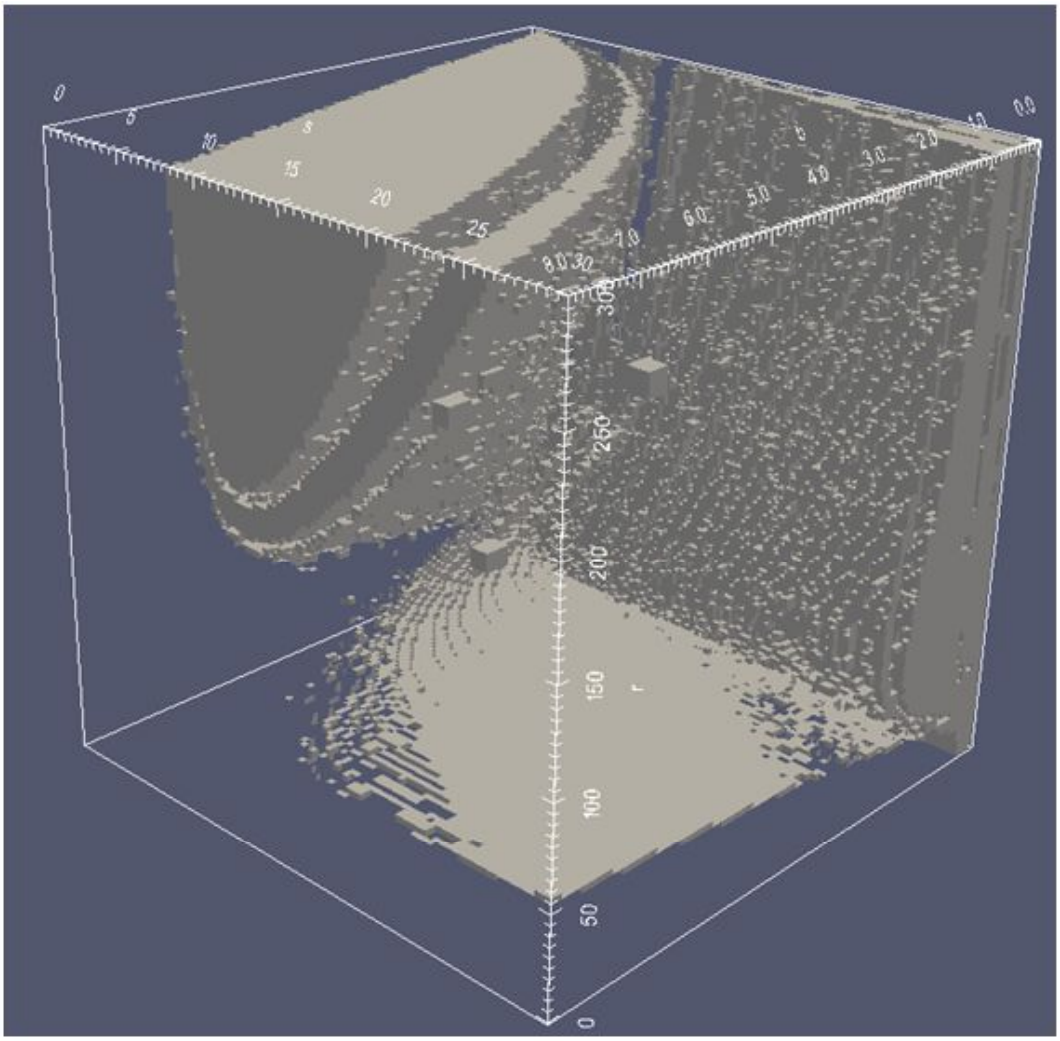


Figura 4.2: Soluções periódicas do sistema de Lorenz usando os expoentes de Lyapunov.

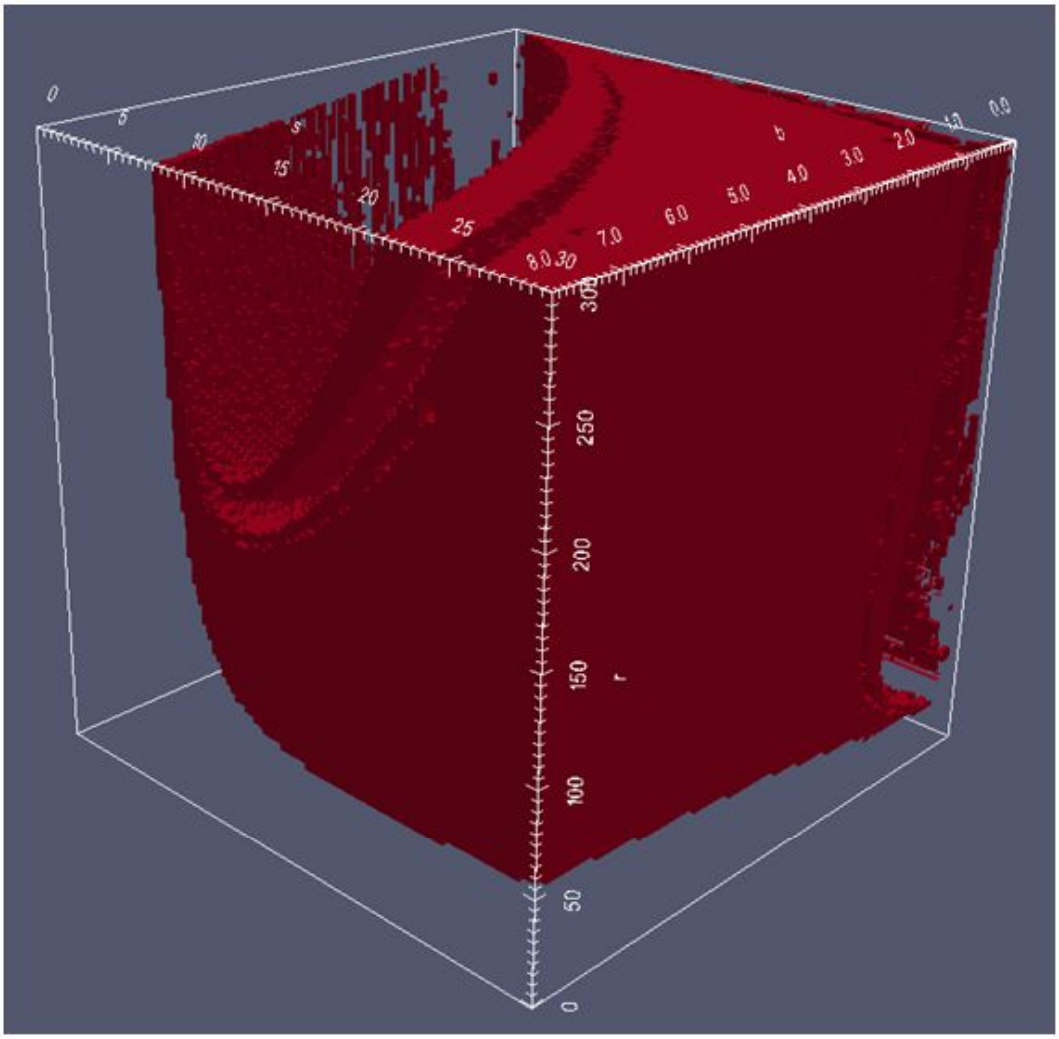
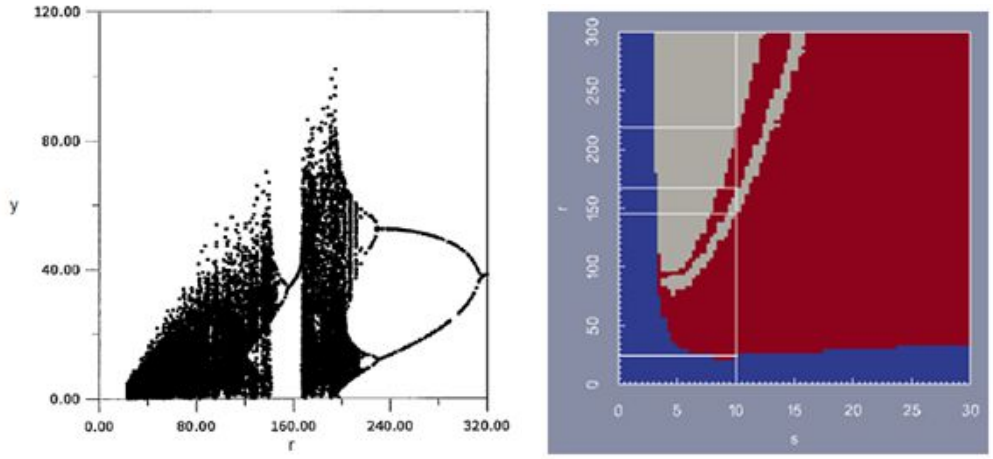


Figura 4.3: Soluções caóticas do sistema de Lorenz usando o expoente de Lyapunov.

Comparando a Figura 4.4a, que é o diagrama de Feigenbaum para $b = 8/3$ e $s = 10$, com o plano de corte $b = 8/3$, Figura 4.4b, pode-se observar, no que diz respeito sobre o eixo $s = 10$, que o caos se estabelece para $25 < \mathrm{r} < 145$ e $165 < \mathrm{r} < 245$. Para a região $145 < \mathrm{r} < 165$ é notada a existência de uma janela de periodicidade em meio ao caos, tanto como para $245 < \mathrm{r} < 300$ que o comportamento também é periódico. Isso é corroborado pela metodologia clássica na construção do diagrama de Feigenbaum.



(a) (b)

Figura 4.4: Janelas de periodicidade na metodologia clássica e na proposta aplicados ao sistema de Lorenz utilizando o expoente de Lyapunov para $s = 10$ e $b = 8/3$.

Para $r < 25$, predomina a ação dos atratores estáveis, como é atestado pelo diagrama de bifurcaçao na Figura 4.5.

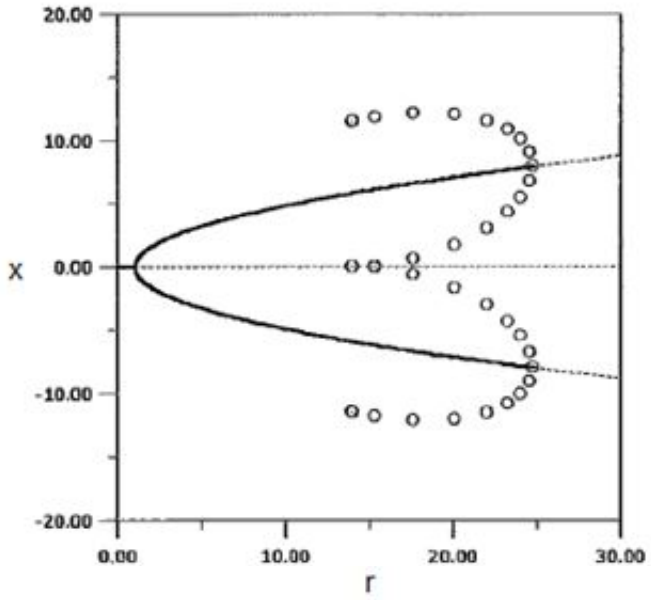


Figura 4.5: Diagrama de bifurcação do sistema de Lorenz. (OURIQUE, 2000)

4.1.2 Caso 2 - Sistema do Reator Exotérmico Clássico

Foram encontradas apenas soluções estacionárias e periódicas, dado que a dimensão do sistema é igual a dois e não é possível encontrar soluções de maior complexidade. Na Figuras 4.6 a 4.9, encontram-se os mapeamentos para cada tipo de solução.

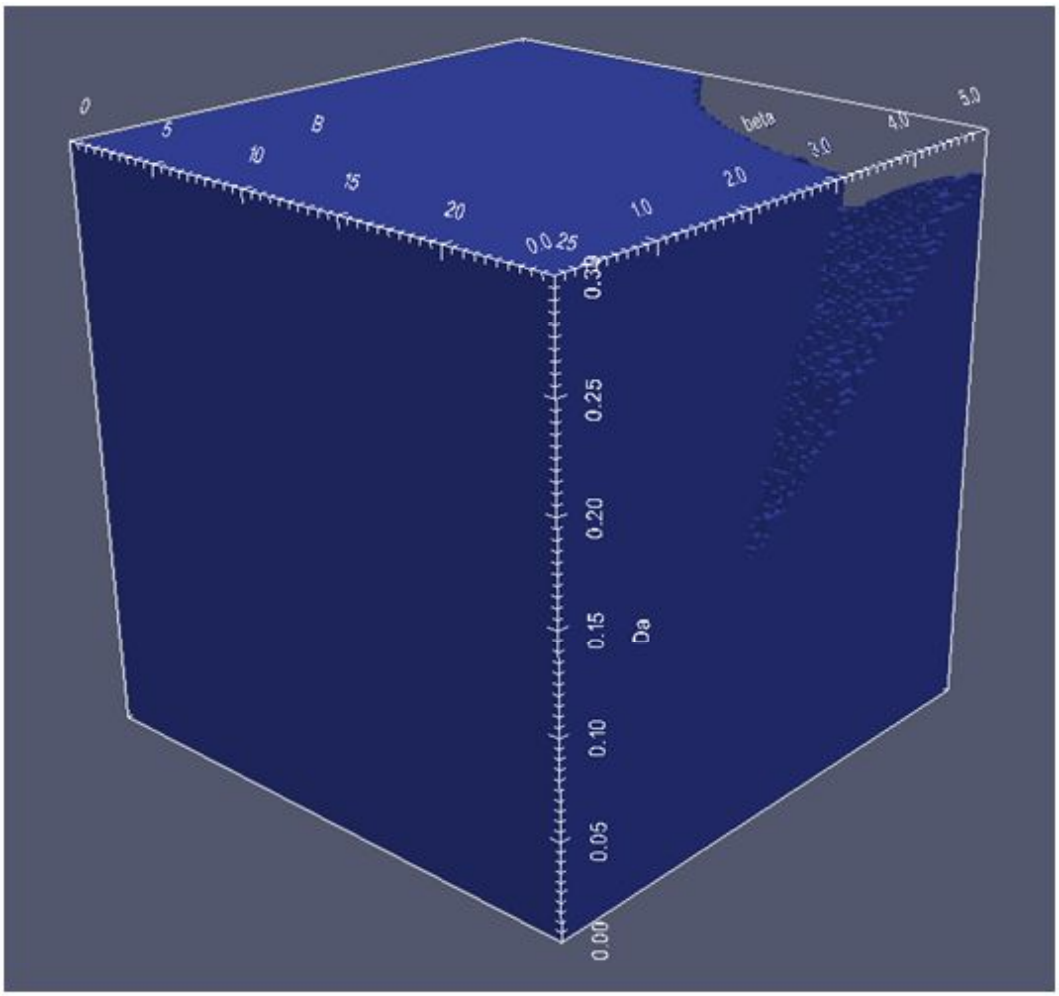


Figura 4.6: Soluções estacionárias do reator CSTR usando o expoente de Lyapunov.

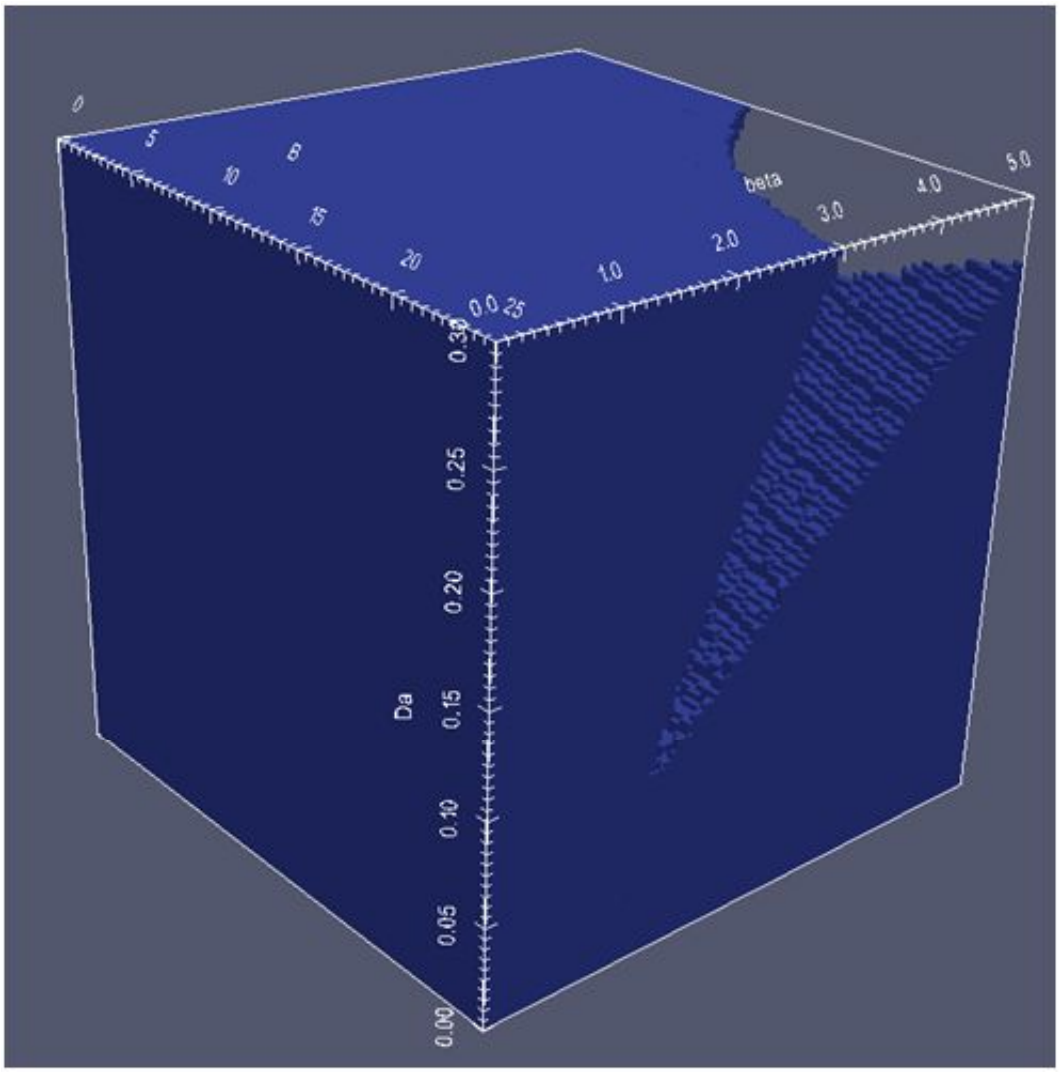


Figura 4.7: Soluções estacionárias do reator CSTR usando detecçao por picos.

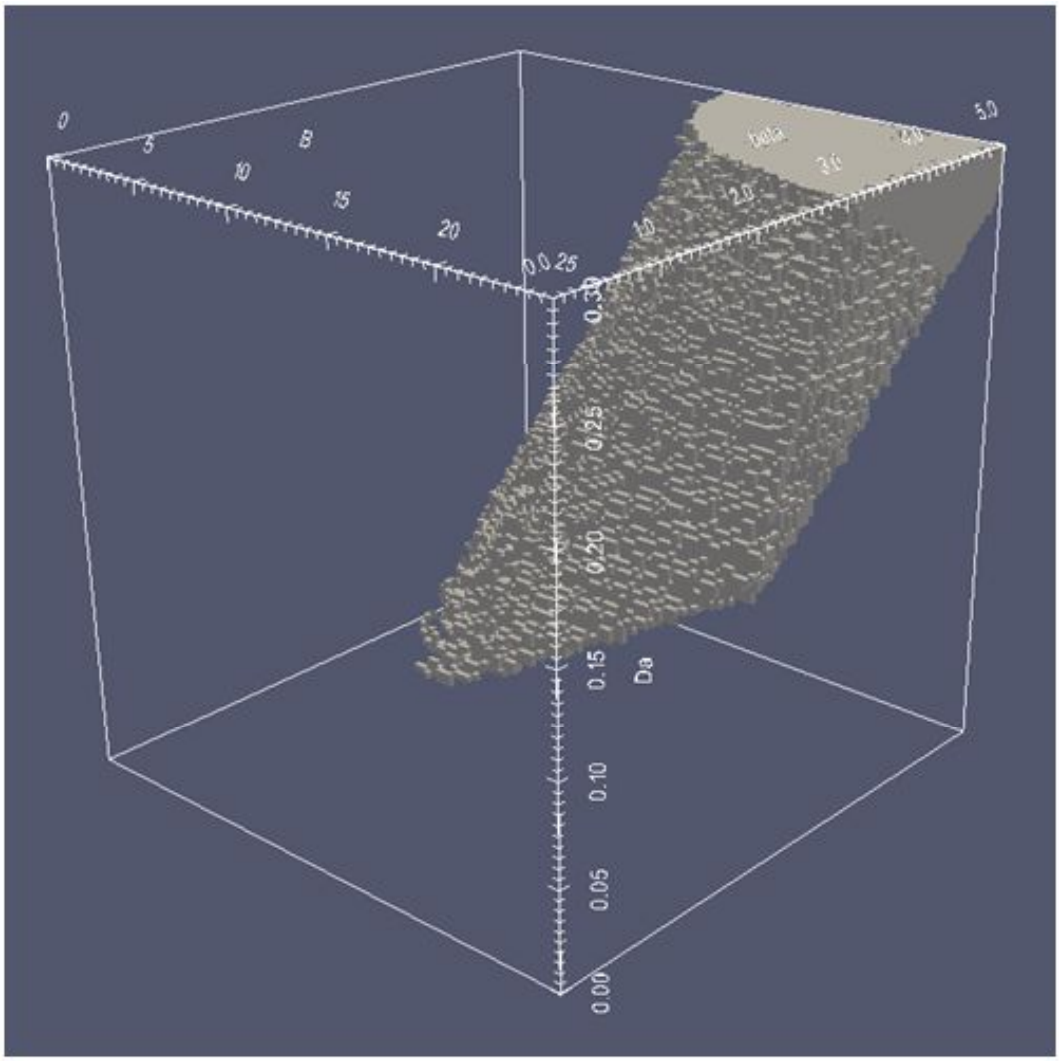


Figura 4.8: Soluções periódicas do reator CSTR usando o expoente de Lyapunov.

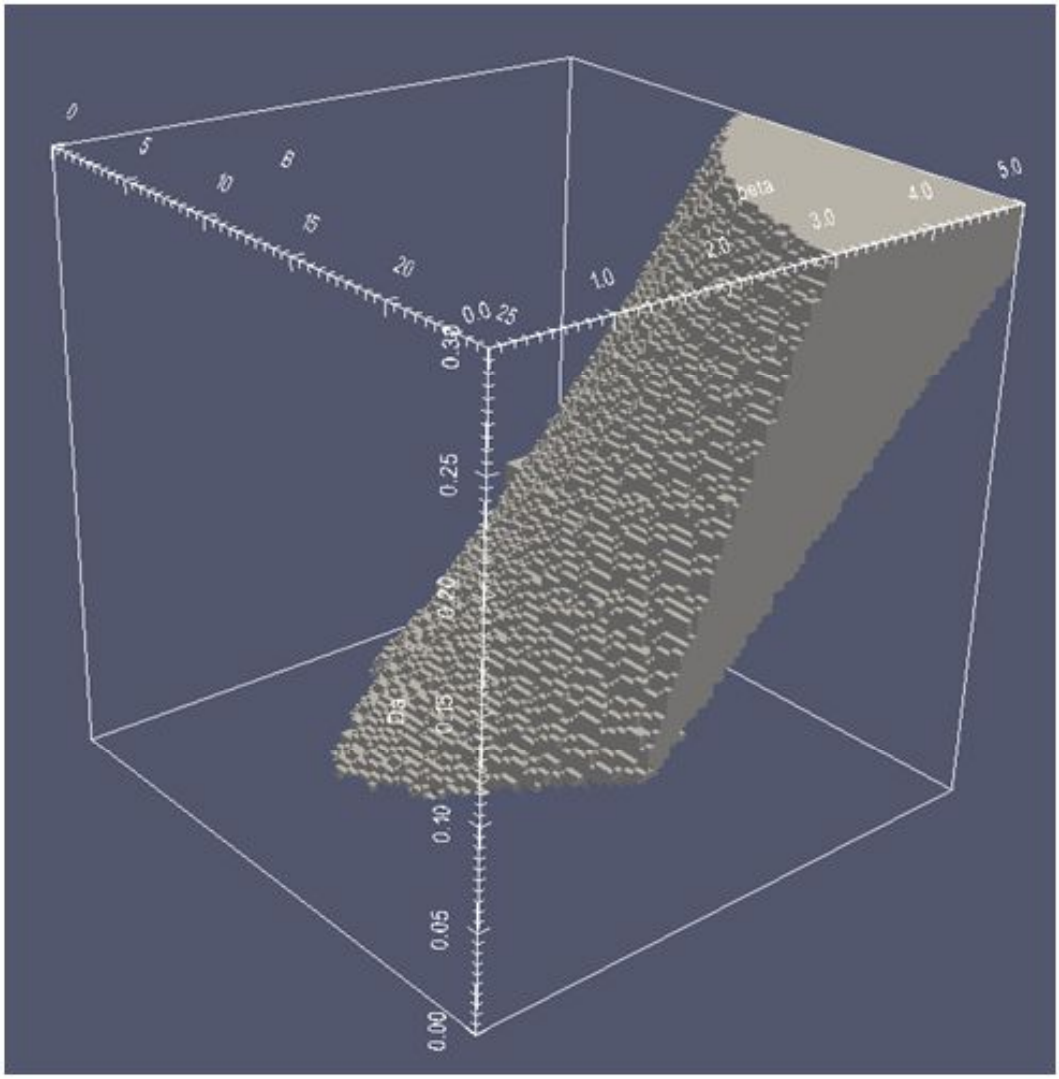


Figura 4.9: Soluções periódicas do reator CSTR usando detecçao por picos.

Para ilustrar cada tipo de comportamento dinâmico das Figuras 4.6 a 4.9, nas Figuras 4.10 e 4.11 são apresentados os planos de fases para os parâmetros $Da = 0,18$; $\beta = 3,5$; $B = 15$; $\theta_c = 0$; $\gamma = 40$ e $Da = 0,18$; $\beta = 3,5$; $B = 17$; $\theta_c = 0$; $\gamma = 40$, respectivamente.

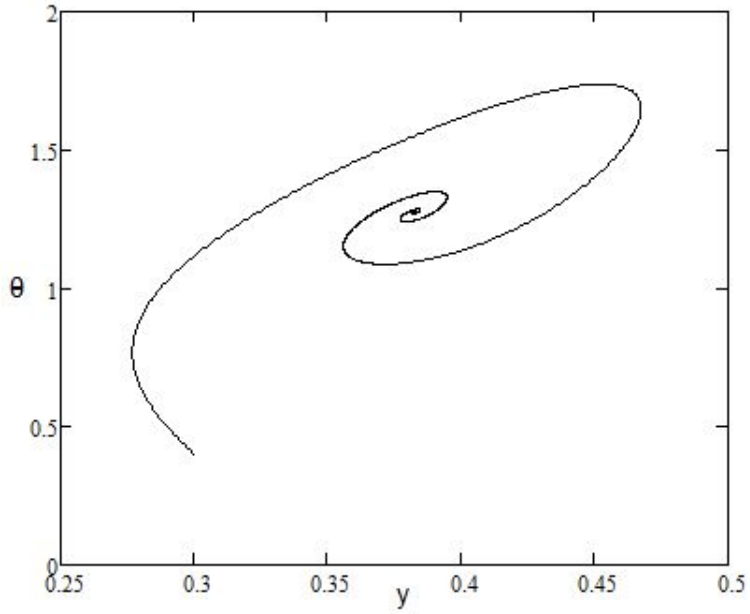


Figura 4.10: Plano de fase do reator CSTR para $Da = 0,18$; $\beta = 3,5$; $B = 15$; $\theta_c = 0$; $\gamma = 40$.

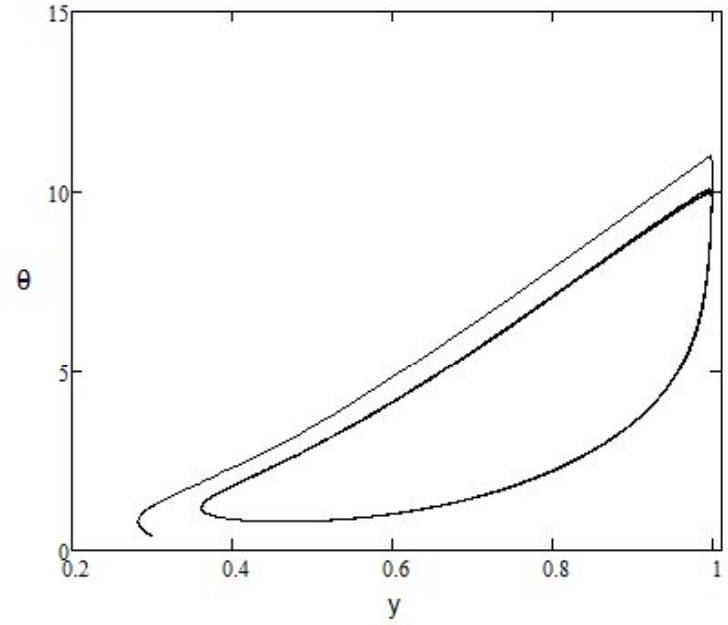


Figura 4.11: Plano de fase do reator CSTR para $Da = 0,18$; $\beta = 3,5$; $B = 17$; $\theta_c = 0$; $\gamma = 40$.

4.1.3 Caso 3 - Sistema do Reator Biológico

Novamente, apenas os comportamentos estacionário e periódico foram detectados. Como este exemplo é composto por apenas duas equações diferenciais (dois estados), somente estes comportamentos são esperados, conforme apresentado nas Figuras 4.12 a 4.15.

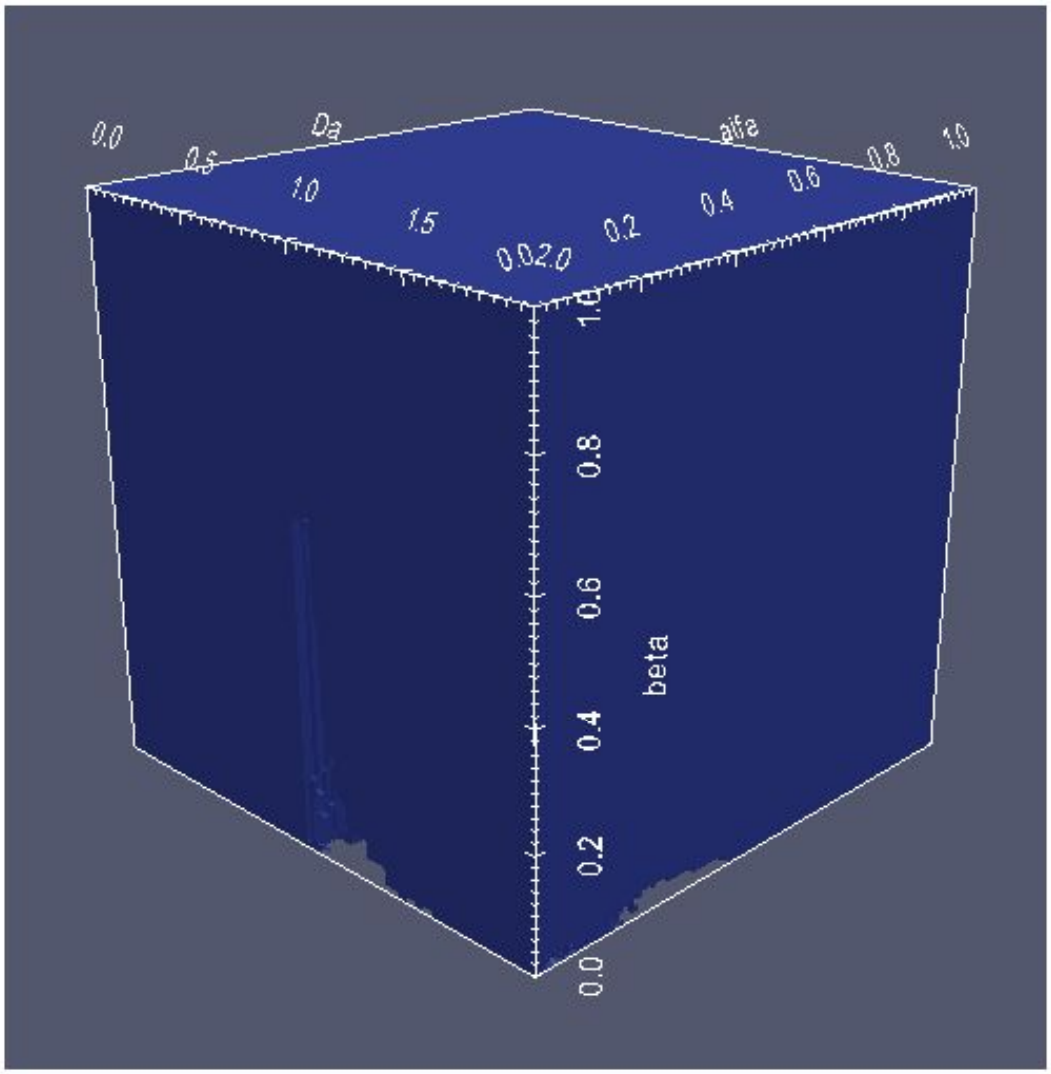


Figura 4.12: Soluções estacionárias do biorreator usando o expoente de Lyapunov.

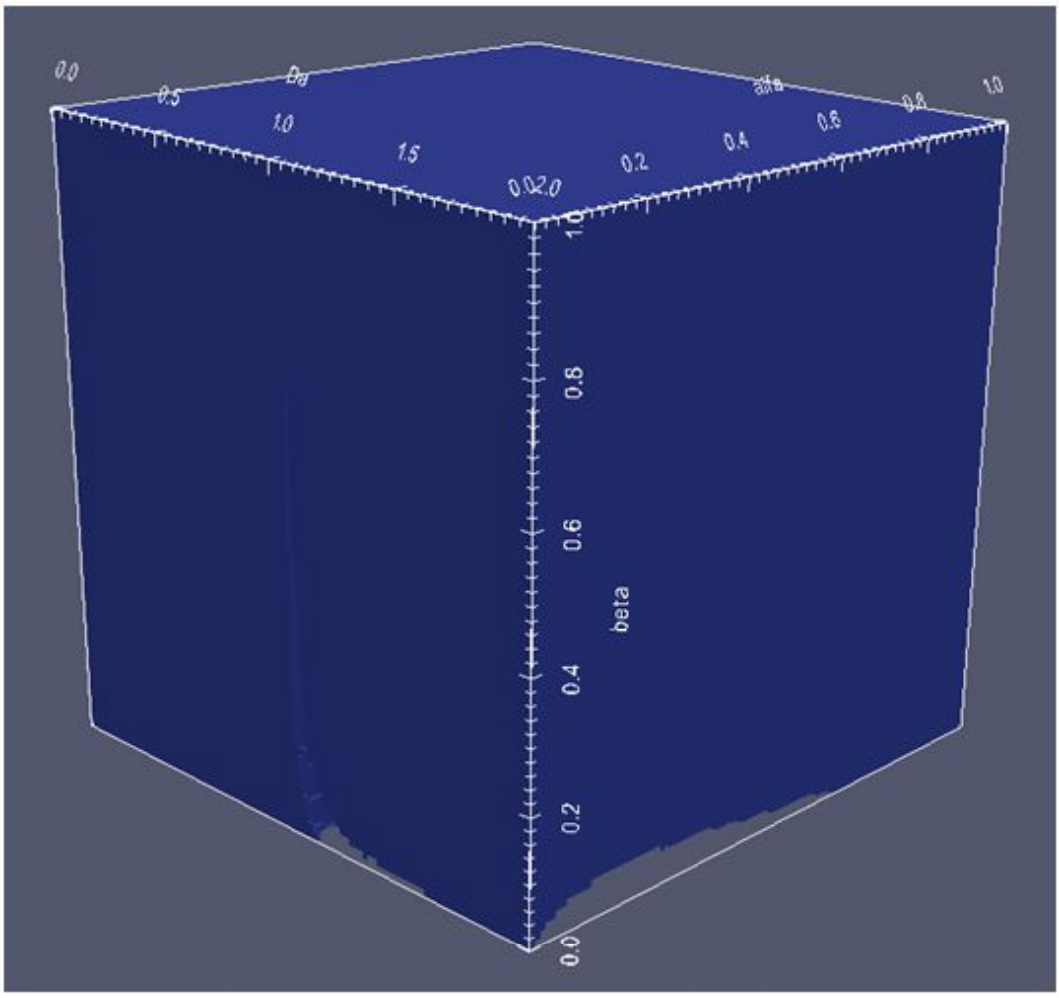


Figura 4.13: Soluções estacionárias do biorreator usando detecçao por picos.

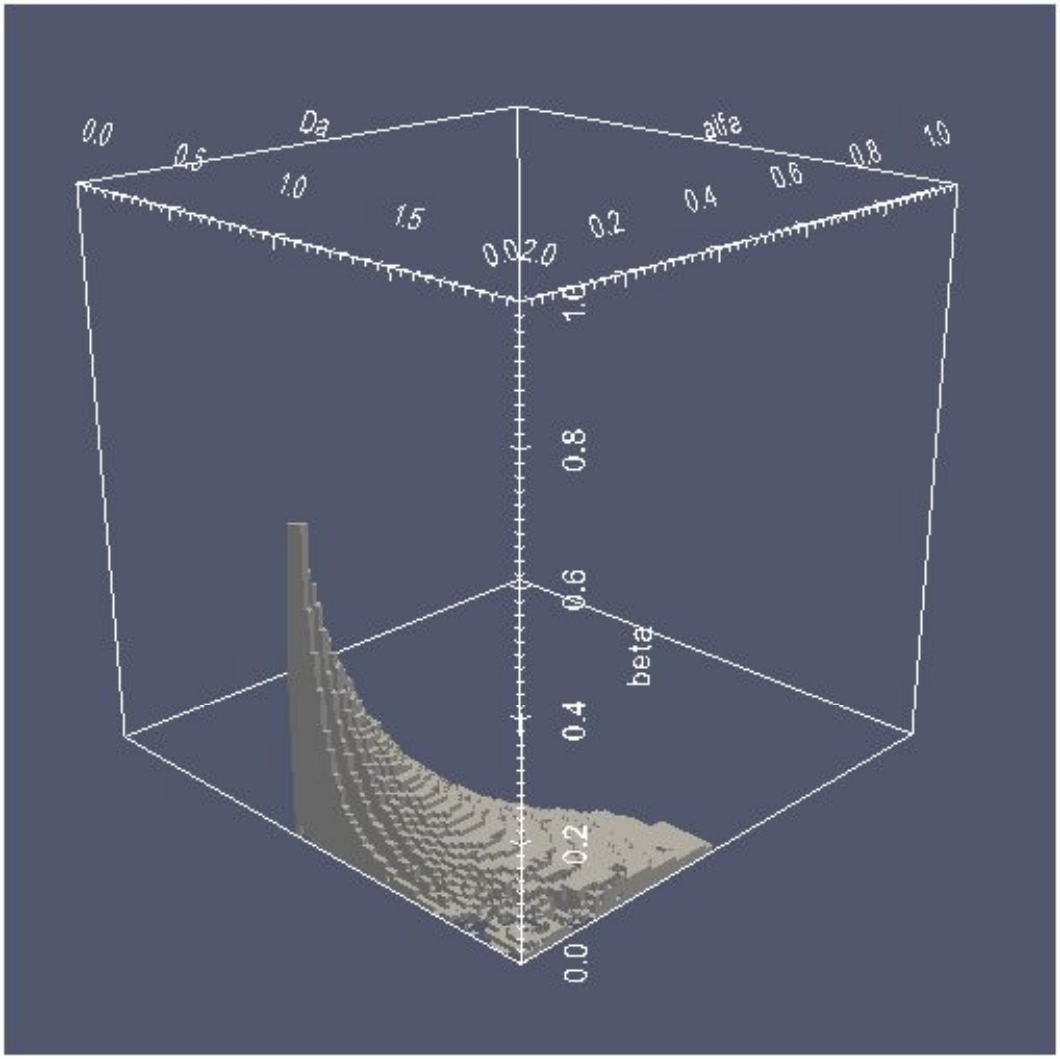


Figura 4.14: Soluções periódicas do biorreator usando o expoente de Lyapunov.

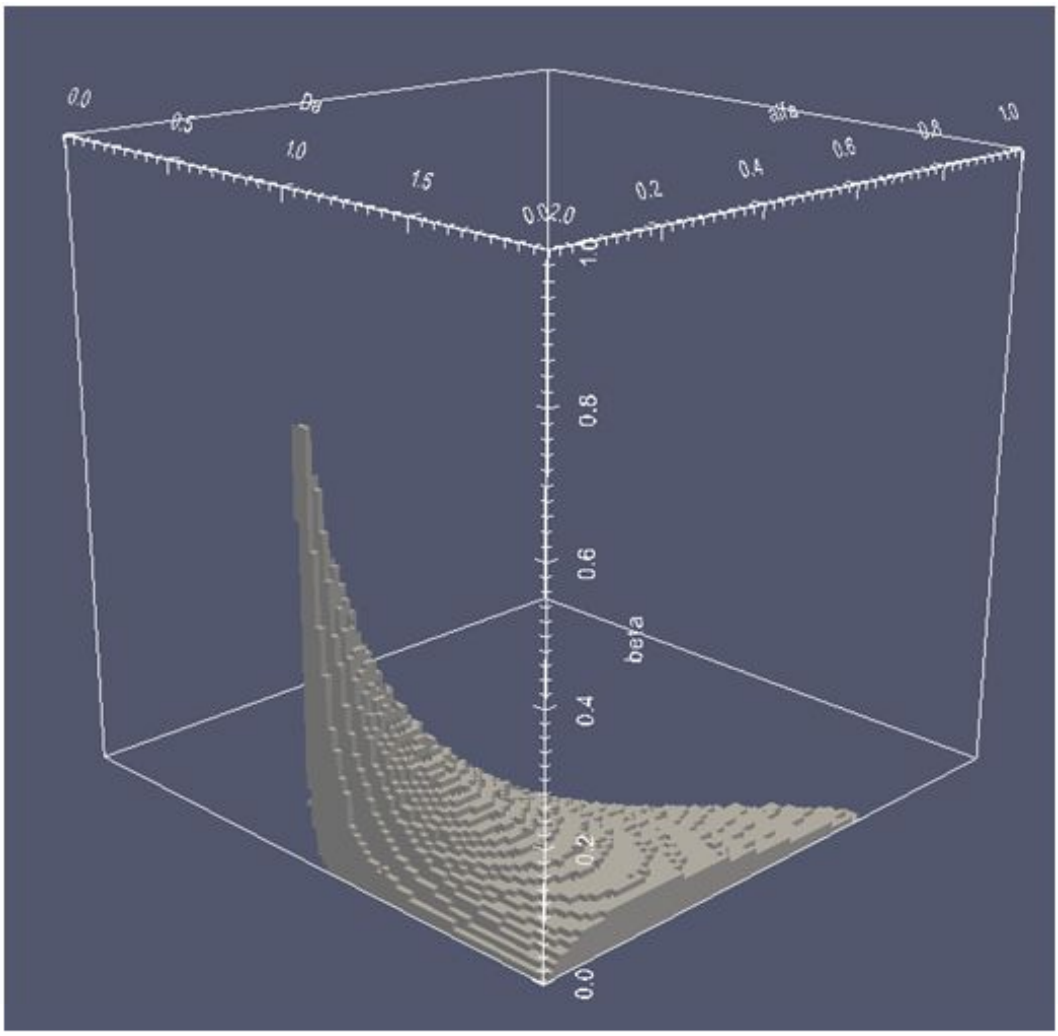


Figura 4.15: Soluções periódicas do biorreator usando detecçao por picos.

Nas Figuras 4.16 e 4.17 são apresentados os planos de fases para a combinação dos parâmetros $\alpha = 0,01$; $\beta = 0,6$; $Da = 1,05$ e $\alpha = 0,0035$; $\beta = 0,6$; $Da = 1,026$, caracterizadas como solução estacionária e periódica, respectivamente.

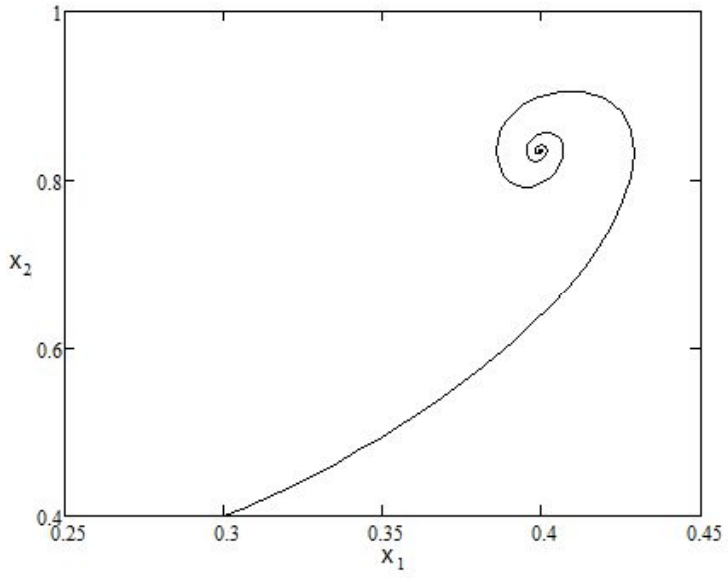


Figura 4.16: Plano de fase do bioreator para $\alpha = 0,01$; $\beta = 0,6$; $Da = 1,05$.

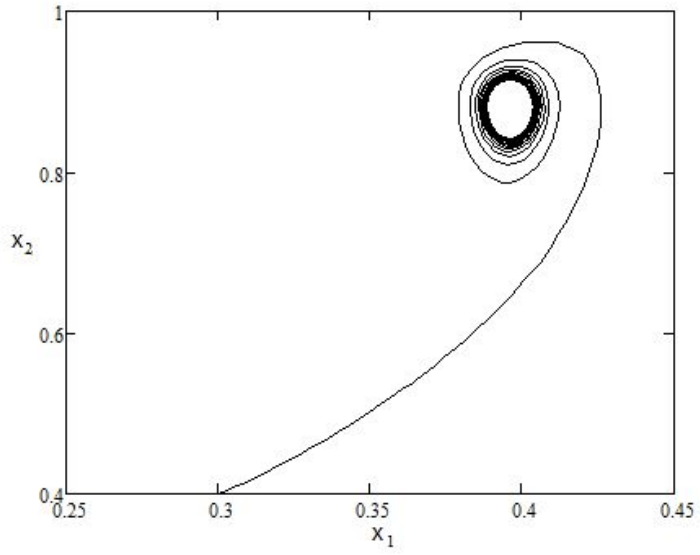


Figura 4.17: Plano de fase do bioreator para $\alpha = 0,0035$; $\beta = 0,6$; $Da = 1,026$.

4.1.4 Resultados de Processamento Paralelo

A Tabela 4.2 apresenta o tempo necessário para criar um mapa em duas dimensões para o caso do sistema de Lorenz com $b = 8/3$, $0 < r < 300$ e $0 < s < 30$, nas mesmas condições de tolerância e número de avaliações. Apresenta também o *Speedup* ($Sp(N)$) e a eficiência ($Ef(N)$), em porcentagem, de cada simulação em função do número de núcleos de processamento (N) e do tempo de processamento ($T(N)$).

Tabela 4.2: Tempo de processamento, *Speedup* e eficiência

Número de núcleos (N)	$T(N)$ (h)	$Sp(N)$	$Ef(N)$
6	1,262518	4,9426	82,38
5	1,468763	4,2486	84,97
4	1,846697	3,3798	84,48
3	2,469766	2,5266	84,22
2	3,610512	1,7283	86,42
1	6,240143	1,0000	100,00

As Figuras 4.18 e 4.19 apresentam os gráficos do *Speedup* ($Sp(N)$) e da eficiência ($Ef(N)$), respectivamente.

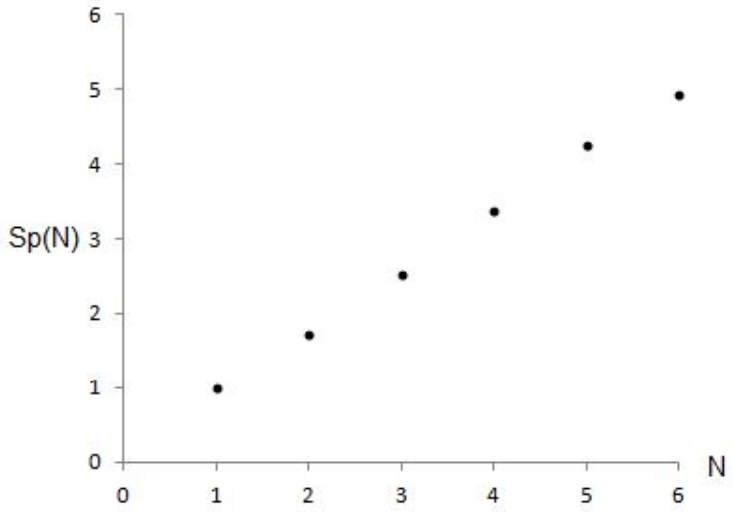


Figura 4.18: *Speedup versus* número de processadores.

É possível observar que o *Speedup* é quase linear e a eficiência cai apenas 2%, devido ao uso de núcleos de processamento lógico, quando passa de dois para três processadores, depois permanece praticamente constante. Demonstrando que o processo é bem escalável.

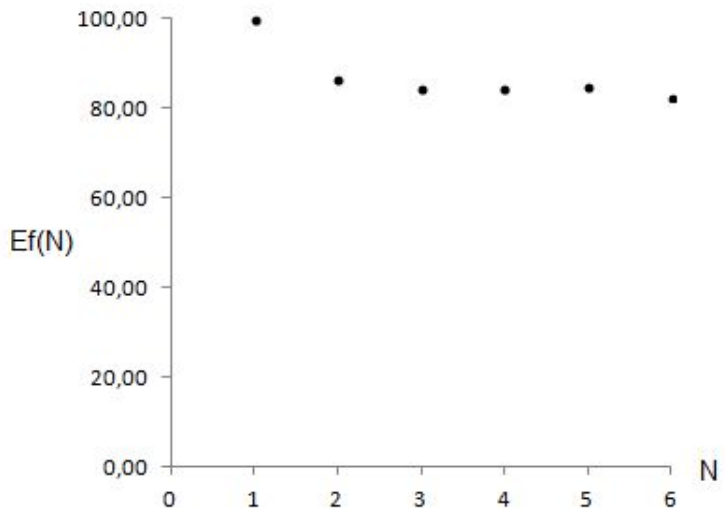


Figura 4.19: Eficiência *versus* número de processadores.

4.2 Discussão

A otimização usando o algoritmo DIRECT apresentou resultados satisfatórios no que diz respeito ao mapeamento do espaço de parâmetros de um sistema. Por ser um método determinístico, é de fácil reprodução e não necessita de múltiplas análises para verificar a resposta obtida quando comparado com métodos não determinísticos.

Outras metodologias usam técnicas como o enxame de partículas e o algoritmo genético. Entretanto, elas realizam cálculos entre os pontos para avaliar a região ótima, isso pode implicar em um tempo maior de processamento, haja vista que em cada etapa esse procedimento deve ser feito. Essa característica não é observada na metodologia proposta. Considerando também que existe pouca relação entre os centros dos hiperretângulos, isso favorece também o emprego de processamento paralelo.

Quanto à função objetivo, também foi testada uma função contínua que tinha como variável o próprio expoente de Lyapunov, porém dessa forma o método buscava apenas dentro de uma região e perdia o aspecto da busca global. Por isso, buscou-se detalhar onde houvesse hiperretângulos vizinhos com comportamentos dinâmicos diferentes. Assim, foi possível delimitar cada comportamento encontrado em todo o espaço paramétrico estudado.

Tanto a caracterização feita pelos expoentes de Lyapunov quanto a feita pela detecção de picos, apresentaram o mesmo mapa, todavia a segunda se mostrou mais sensível às tolerâncias e passos de integração visto que, dependendo da tolerância adotada, uma solução, por exemplo, periódica pode ser confundida como caótica.

Com relação à resolução das fronteiras, a caracterização feita pelos expoentes de

Lyapunov proporciona um mapa mais preciso quando comparado com a caracterização feita pela detecção de picos. Como exemplo, cita-se o caso do reator exotérmico clássico; um ponto no mapa gerado pela caracterização feita pelos expoentes de Lyapunov está na borda da região de soluções periódicas, enquanto que no mapa gerado pela caracterização feita por detecção de picos o mesmo ponto está no interior na região. A Figura 4.20 apresenta a simulação desse caso e a Figura 4.21 ilustra um corte do mapa gerado no caso 2 com a localização deste ponto.

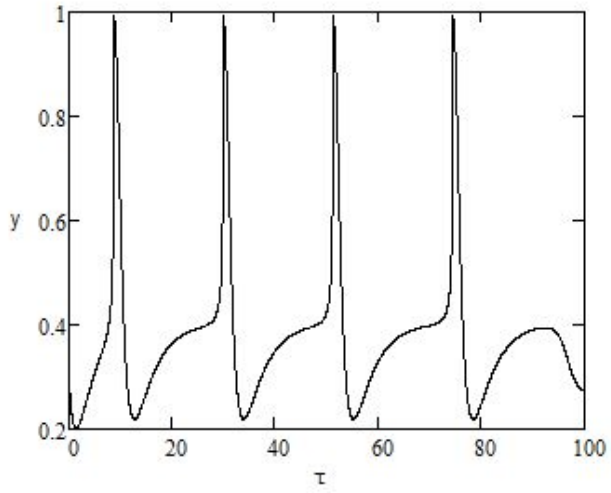


Figura 4.20: Simulaçao do modelo do reator CSTR para B = 15,668 β = 2,264 Da = 0,107.

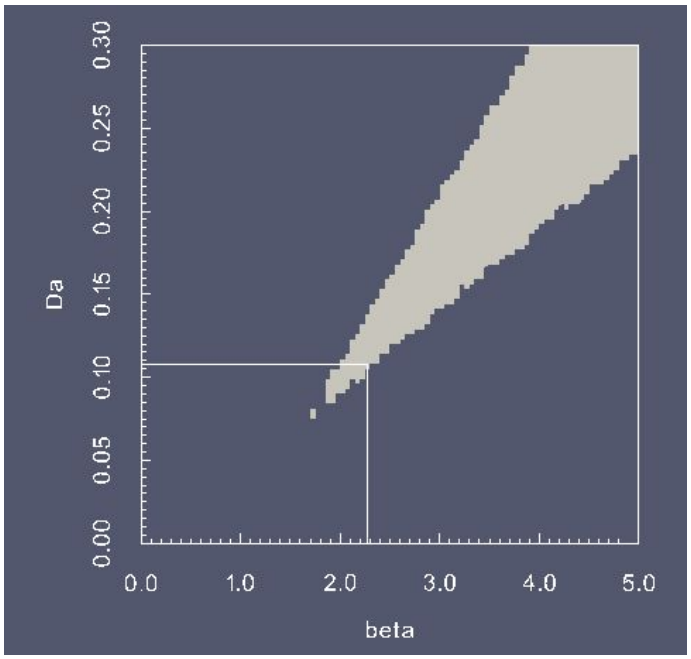


Figura 4.21: Corte do mapa de soluções periódicas do reator CSTR para B = 15,668.

Então, pode-se observar na Figura 4.20 que o sistema demora a atingir o estado

estacionário, e durante esse período ele aparenta ser periódico. De fato, o sistema está na iminência de apresentar um comportamento periódico, portanto, ainda que classificado como periódico em ambas as metodologias, o fato desse ponto estar na borda é mais coerente.

Nas Figuras 4.4a, 4.4b e 4.5 é possível observar que os diagramas de bifurcação são apenas cortes dos mapas gerados pela metodologia proposta nesse trabalho. Entretanto, os diagramas de bifurcação apresentam informações, como pontos limite, pontos de bifurcação, órbitas instáveis entre outras. Estes pontos também poderiam ser aproximadamente localizados pela metodologia proposta, pois estão na fronteiras fronteiras entre comportamentos distintos.

Logo, este é um algoritmo para análise dinâmica prévia e complementar. Para estudos posteriores de bifurcação e refinamento dos diagramas, são recomendados os pacotes computacionais apresentados pelo Capítulo 2.

Capítulo 5

Conclusões e Sugestões

Neste trabalho foi demonstrado que é possível fazer a caracterização preliminar da estrutura dinâmica de um sistema combinando algoritmos de otimização e expoentes de Lyapunov, sem que seja necessária qualquer informação preliminar sobre o comportamento dinâmico do sistema estudado.

Para delinear os contornos do mapa de soluções especiais, foi utilizada o algoritmo DIRECT para a otimização dos parâmetros de cada modelo e buscar regiões próximas a pontos nos quais há mudança de comportamento dinâmico. Os comportamentos que se propõe classificar são as soluções estacionárias, soluções periódicas e soluções caóticas, desde que apresentem estabilidade.

A função objetivo proposta permite delimitar as regiões do espaço de parâmetros que apresentam os comportamentos dinâmicos propostos. Ainda é possível trocar essa função para que sejam encontrados tipos de atratores ou uma região com um comportamento específico.

Apesar da simplicidade matemática dos modelos empregados para ilustrar os resultados, eles apresentam soluções dinâmicas complexas e o algoritmo proposto consegue mapear em pouco tempo a região paramétrica. A acurácia dos resultados depende dos critérios de tolerância adotados pelo usuário, impactando também no tempo necessário para uma análise dinâmica de um sistema desconhecido. Sabendo as regiões mais promissoras para o sistema apresentar um dado comportamento dinâmico, é possível direcionar a análise de bifurcações nos pacotes computacionais usuais de continuação paramétrica.

Portanto, o algoritmo proposto neste trabalho é recomendado quando se deseja realizar um estudo do comportamento dinâmico de um sistema sem o conhecimento prévio de onde ocorrem os fenômenos que se deseja investigar.

Para trabalhos futuros, sugere-se a comparação do desempenho e do mapa gerado entre outros métodos de otimização e o DIRECT. Neste trabalho o tempo final de simulção é arbitrado pelo usuário, entretanto, quando a solução é estacionária, é possível determinar esse tempo calculando o módulo do inverso do menor valor

característico da matriz Jacobiana do sistema vezes cinco, para garantir que a dinâmica esteja estabelecida e evitar que o método caracterize erroneamente a solução estacionária. Ou seja, quando o sistema se aproxima de pontos de bifurcação, um valor característico se aproxima de zero e o tempo necessário para atingir o estado estacionário aumenta.

Capítulo 6

Referências Bibliográficas

AGRAWAL, P., LEE, C., UM, H. C., RAMKRISHNA, D., 1982, "Theoretical Investigations of Dynamic Behavior of Isothermal Continuous Stirred Tank Biological Reactors", Chem. Engng. Sci., v. 37, pp. 453-462.

ARGYRIS, J., FAUST,G., HAASE, M., 1994, An Exploration of Chaos, Texts on Computational Mechanics - Volume VII, 1ª ed. The Netherlands, North-Holland, Elsevier Science B. V.

ARNOL'D, V. I., 1978, Mathematical Methods of Classical Mechanics, 1ª ed. New York, Springer-Verlag.

ARNOL'D, V. I., 1988, Geometrical Methods in the Theory of Ordinary Differential Equations, 2ª ed. Berlim, Springer.

BALAKOTAIAH, V. and LUSS, D., 1983, Multiplicity features of reacting systems - dependence of the steady-states of a CSTR on the residence time, Chem. Engng. Sei., 38, p. 1709-1721. Journal of Loss Prevention in the Process Industries, v. 17, n.5, p. 355–364, 2004.

BILOUS, O., AMUNDSON, N. R., 1955, Chemical Reactor Stability and Sensitivity, AIChE J., v. 1 , p. 513-521.

CHOW, S-N., HALE, J. K., 1982, Methods of Bifureation Theory, 1ª ed. New York, Springer-Verlag.

CRAWFORD, J. D., OMOHUNDRO, S. M., 1984, On the Global Structure of Period Doubling Flows, Physica 13D 1 & 2, p. 161 - 180.

DERCOLE, F. e RINALDI, S., 2011, Dynamical Systems and Their Bifurcations, p. 291-325. In: Advanced Methods of Biomedical Signal Processing, eds. Cerutti, S. e Marchesi, C., IEEE-Wiley Press, New York, NY.

DOEDEL, E. J., HEINEMANN, R. F., 1983, Numerical Computation of Periodic Solution Branches and Oscillatory Dynamics of the Stirred Tank Reactor with A $\rightarrow$ B $\rightarrow$ Reactions, Chem. Engng. Sci., v. 38, pp. 1493-1499.

DOEDEL, E. J., KERNEVEZ, J. P., 1989, Auto 86 - Software for Continuation and Bifurcation Problems in Ordinary Differential Equations - Including User Manual, Passadena - CA, Center for Research on Parallel Computing - California Institute of Technology.

DOEDEL, E. J., WANG, X. I., FAIRGRIEVE, T. F., 1994, Auto 94 - Software for Continuation and Bifurcation Problems in Ordinary Differential Equations, Pasadena - CA, Center for Research on Parallel Computing - California Institute of Technology.

FREIRE, E.,PIZARRO, L.,RODRÍGUEZ-LUIZ, A. J. e FERNANDEZ-SÁNCHEZ, F., 2005, Multiparametric Bifurcations in na Enzyme-Catalysed Reaction Model. International Journal of Bifurcation and Chaos, v. 15, No. 3, p. 905-947.

FREITAS FILHO, I. P., BISCAIA JR., E. C., PINTO, J. C., 1994, Steady-State Multiplicity in Continuous Bulk Polymerization Reactors - A Generic Approach, Chem. Engng. Sci., v. 49, p. 3745-3755.

FREITAS FILHO, I. P., 1993, Multiplicidade de Estados Estacionários em Reatores Contínuos de Polimerização em Massa - Análise do Caso Geral. Tese de M.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

FRIEDLY, J.C., 1972, Dynamic Behavior of Processes, Prentice Hall.

GLENDINNING, P., 1995, Stability, Instability and Chaos: An Introduction to the Theory of Nonlinear Differential Equations, Cambridge Texts in Applied Mathematics, 1ª ed. Cambridge, Cambridge Press.

GOLUBITSKY, M., KEYFITZ, B. L., 1980, A Quantitative Study of the Steady State Solutions for a Continuous Flow Stirred Tank Chemical Reactor, SIAM 3. Math. Anal., v. 11, p. 316-339.

IOSS. G., JOSEPH, D. D., 1980, Elementary Stability and Bifurcation Theory, 1ª ed. New York, Springer-Verlag.

JACKSON, E. A., 1991, Perspective of Nonlinear Dynamics - Volume 1, 1ª ed. Cambridge, Cambridge University Press.

JONES, D. R., PERTTUNEN, C. D. e STUCKMAN, B. E., 1993, Lipschitzian Optimization Without the Lipschitz Constant. Journal od Optimization Theory

and Application, v. 79, No. 1, p. 157-181

KUBICEK, M., MAREK, M., 1983, Computational Methods in Bifurcation Theory and Dissipative Structures, 1ª ed. New York, Springer-Verlag.

MALLORY, K. e VAN GORDER, R. A., 2015. Competitive Modes for the Detection of Chaotic Parameter Regimes in the General Chaotic Bilinear System of Lorenz Type,. International Journal of Bifurcation and Chaos, V.25, No 4, 1530012 (32 pages)

MARDSEN, I. E., MCCRACKEN, M., 1976, The Hopf Bifurcation and Its Applications - Applied Mathematical Sciences (19), 1ª ed. New York, Springer-Verlag.

MELO, P. A., 2000. Dinâmica e Estabilidade de Reatores Tubulares de Polimerização com Reciclo. Tese de D.Sc., COPPE/UFRJ, Rio de Janeiro, Brasil.

MELO, P. A., BISCAIA Jr., E. C., PINTO, J. C., 2003. "The bifurcation behavior of continuous free-radical solution loop polymerizaction reactors". Chemical Engineering Science, v. 58, pp. 2805-2821.

MELO, P. A., PINTO, J. C., 2008, Introdução à Modelagem Matemática e Dinâmica Não-Linear de Processos Químicos, Escola Piloto Virtual Giulio Massarani, Programa de Engenharia Química da COPPE, UFRJ, RJ.

NELE, M., PINTO, J. C., 1997, Dynamic Behavior of a Continuous Autothermal Isobutylene Polymerization Reactor, 11. Appl. Polym. Sci., v. 65, p. 1403-1413.

OECHSLER B. F., 2011, Análise de Bifurcações de Problemas de Micromistura em Reatores de Polimerização em Solução. Dissertação de M.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

OURIQUE, C. O., 2000, Métodos Alternativos Para Análise Dinâmica em Processos Químicos. Tese de D.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

OURIQUE, C. O., BISCAIA JR., E. C., & PINTO, J. C., 2002. The Use of Particle Swarm Optimization for Dynamical Analysis in Chemical Process. Computers and Chemical Engineering, v. 26, p. 1783-1793.

PINTO, J. C., 1990, Dynamic Behavior of Continuous Vinyl-Chloride Suspension Polymerization Reactors - A Simple Model Analysis, Polym. Engng. Sci., v. 30, p. 291-302.

PINTO, J. C., 1991, Uma Revisao sobre Sistemas Dinamicos Nao Lineares, Teoria de Bifurcações e Comportamento Dinamico de Sistemas da Engenharia Quimica,

Rev. Bras. Eng. - RBE, v. 8, p. 5-65.

RABINOWITZ, P., 1977, Applications of Bifurcation Theory, 1ª ed. New York, Academic Press Inc.

RAMMINGER, G. O., SECCHI, A. R., 2007, Integração do software AUTO com o simulador de processos EMSO, Anais do VII COBEC - IC, São Paulo, SP.

RODRIGUES, K. K., 2011, Comportamento Caótico em Reatores Contínuos de Polimerização em Solução via Radicais Livres. Dissertação de M.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

ROSA, I. S., 2013, Análise Dinâmica e de Estabilidade de Reatores Tubulares de Polimerização de Propeno do Tipo Loop. Tese de D.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

RUELLE, D . ,1989, Elements of Differentiable Dynamics and Bifurcation Theory, 1ª ed. New York, Academic Press Inc.

SALAU, N. P. G., NEUMANN, G. A., TRIERWEILER, J. O., A. R. SECCHI, 2009, Multivariable Control Strategy Based on Bifurcation Analysis of an Industrial Gas-Phase Polymerization Reactor. Journal of Process Control, v. 19, p. 530-538

SAVI, M. A., 2006. Dinâmica Não-Linear e Caos. Rio de Janeiro, e-papers.

SEYDEL, R., 2010, Practical Bifurcation and Stability Analysis From Equilibrium to Chaos. In: Interdisciplinary Applied Mathematics, v. 5, 3ª ed. New York, Springer.

SMITH, C. B., KUSZTA, B., LYBERATOS, G. & BAILEY, J. E., 1983, "Period Doubling and Complex Dynamics in an Isothermical Chemical Reaction System", Chem. Engng. Sci., v. 38, p. 425-430.

TANENBAUM, A. S., 2009, Sistemas operacionais modernos, 3ª ed. São Paulo, Pearson Education

THOMPSON, J. M. T., STEWART, H. B., 1986, Nonlinear Dynamics na Chaos, 1ª ed. New York, John Wiley & Sons.

UPPAL, A., RAY, W. H., & POORE, A. B., 1974. On the dynamic behaviour of continuous stirred tank reactors. Chemical Engineering Science, v. 29, pp. 967-985.

VAN HEERDEN, C., 1953, Autothermic Process, Ind. Engng. Chem., v. 45, p. 1242-1247.

WIGGINS, S., 1990, Introduction to Applied Nonlinear Dynamical Systems and Chaos - Texts in Applied Mathematics (2), 1 ed. New York, Springer-Verlag.

WOLF, A.; SWIFT, J.B.; SWINNEY, H.L. & VASTANO, J.A., 1985, Determining Lyapunov Expoents from a Time Series, Physica 16D, p.285-317.

Capítulo 7

Apêndice

```
unit uSurface;

interface

  uses
    uBase, uEvaluator, System.Generics.Collections,
        System.Generics.Defaults, System.SyncObjs;

  const
    MAX_DIV = 8;
    MAX_DELTA = 4;
    MAX_REFINE = 100;

  type
    TSurfaceNode = class;
    TChildrenList = array [0 .. 2] of TSurfaceNode;

    TSurfaceConfig = record
      eval: TEvaluator;
      param0, param1: TVectorf;
      constParams: TVectori;
      multithreaded: Boolean;
      constructor Create(eval: TEvaluator; param0, param1: TVectorf;
        constParams: TVectori; multithreaded: Boolean);
    end;

    TSurfaceNodeIterator = record
```

```
    root, it: TSurfaceNode;
    constructor Create(root: TSurfaceNode);
    procedure Reset;
    function HasNext: Boolean;
    function GetNext: TSurfaceNode;
  private
    function FirstChild(node: TSurfaceNode): TSurfaceNode;
    function NextNode(node: TSurfaceNode): TSurfaceNode;
  end;

  PRefineCandidate = ^TRefineCandidate;

  TRefineCandidate = record
    node: TSurfaceNode;
    dir: Integer;
    constructor Create(node: TSurfaceNode; direction: Integer);
    procedure Refine;
  end;

  TSurfaceNode = class
  public
    eval: TEvalResult;
    changed: Boolean;

    dim: Integer;
    coord: TVectorf;
    size: TVectorf;
    ndiv: TVectori;
    center: TVectorf;

    parent: TSurfaceNode;
    index: Integer;
    dir: Integer;
    children: TChildrenList;


    constructor Create(parent: TSurfaceNode; d: Integer;
        index: Integer);
    constructor CreateRoot(dimensions: Integer);
```

```
  destructor Destroy;
  function IsSquare: Boolean;
  function LongestDimension: Integer;
  function ShortestDimension: Integer;
  function LongestDimensionIndex: Integer;
  function ShortstDimensionIndex: Integer;
  function DeltaDiv(var min, max: Integer): Integer;
  procedure Divide(direction: Integer);
  function DescendentOf(node: TSurfaceNode): Boolean;
  function GetBoundaryList(dir: Integer): TList<TSurfaceNode>;
  function GetBoundaryNode(direction: Integer): TSurfaceNode;
  class function EvalSize(divisions: Integer): Double;
private
  procedure SetSize;
  procedure Shrink(dimension: Integer);
  function CanReduce(boundsDir, refDir: Integer): Boolean;
  function GetBrother(direction: Integer): TSurfaceNode;
  function Reduce(boundDir, refDir, refIndex: Integer):
      TSurfaceNode;
end;

TSurfaceTree = class
public
  config: TSurfaceConfig;
  dim: Integer;
  isConst: array of Boolean;
  leafNodes: Integer;
  root: TSurfaceNode;

  minNode: Integer;
  maxNode: Integer;
  bndTarget: Integer;
  refineList: TList<TRefineCandidate>;
  refineType: String;

  updateNodes: TQueue<TSurfaceNode>;
  updateSemaphore: Cardinal;
  procedure BeginUpdate;
  procedure EndUpdate;
```

```
    function hasNextUpdate: Boolean;
    function nextUpdate: TSurfaceNode;

    constructor Create(config: TSurfaceConfig); overload;
    destructor Destroy; override;
    function GetIterator: TSurfaceNodeIterator;
    function NodeAt(coord: TVectorf): TSurfaceNode;

    function GetCandidates(sort: Boolean):TList<TRefineCandidate>;
    procedure Step(n: Integer; sort: Boolean); overload;
    procedure Step(candidate: TRefineCandidate); overload;
    function Evaluate(coord: TVectorf): TEvalResult; overload;
    function Evaluate(node: TSurfaceNode): TEvalResult; overload;
    function GetParamAt(coord: TVectorf): TVectorf;
  private
    critical: TCriticalSection;
    procedure SelectCandidates(sort: Boolean);
    procedure SelectMinCandidates;
    procedure SelectBoundaryCandidates;
    function IsCandidate(node: TSurfaceNode): Boolean;
    procedure InitStats;
    procedure UpdateStats(node: TSurfaceNode);
  end;

implementation

  uses
    Math, SysUtils, Winapi.Windows;

  { TSurfaceNode }

  function TSurfaceNode.CanReduce(boundsDir, refDir: Integer):
  Boolean;
  begin
    if (dir = refDir) or (dir = Abs(boundsDir) - 1) then
      Result := True
    else
      Result := False;
  end;
```

```
constructor TSurfaceNode.Create(parent: TSurfaceNode; d, index:
Integer);
begin
  Self.parent := parent;
  Self.index := index;

  dim := parent.dim;
  size := vCopy(parent.size);
  ndiv := vCopy(parent.ndiv);
  coord := vCopy(parent.coord);
  center := vZerosf(dim);

  Shrink(d);
  coord[d] := coord[d] + index * size[d];
  vMla(center, size, 1 / 2, coord);

  dir := -1;
  eval := nil;
  changed := True;
end;

constructor TSurfaceNode.CreateRoot;
begin
  dim := dimensions;
  coord := vZerosf(dim);
  size := vOnesf(dim);
  ndiv := vZerosi(dim);
  center := vCopy(size);
  vDiv(center, 2);

  parent := nil;
  index := -1;
  dir := -1;
  eval := nil;
  changed := True;
end;

function TSurfaceNode.DeltaDiv(var min, max: Integer): Integer;
```

```
var
  i: Integer;
begin
  min := ndiv[0];
  max := ndiv[0];
  for i := 1 to dim - 1 do
    if ndiv[i] < min then
      min := ndiv[i]
    else if ndiv[i] > max then
      max := ndiv[i];
  Result := max - min;
end;

function TSurfaceNode.DescendentOf(node: TSurfaceNode): Boolean;
var
  n: TSurfaceNode;
begin
  n := parent;
  while (n <> nil) and (n <> node) do
    n := n.parent;
  Result := (n <> nil);
end;

destructor TSurfaceNode.Destroy;
var
  n: TSurfaceNode;
begin
  if dir < 0 then
    eval.Free
  else
    for n in children do
      n.Destroy;
  inherited Destroy;
end;

procedure TSurfaceNode.Divide(direction: Integer);
var
  i: Integer;
begin
```

```
  if dir >= 0 then
    raise Exception.Create('Node already divided');

  dir := direction;
  for i := 0 to 2 do
    children[i] := TSurfaceNode.Create(Self, dir, i);
  children[1].eval := Self.eval;
end;

class function TSurfaceNode.EvalSize;
begin
  Result := Power(3, -divisions);
end;

function TSurfaceNode.GetBoundaryList(dir: Integer):
TList<TSurfaceNode>;
var
  l, t: TList<TSurfaceNode>;
  n: TSurfaceNode;
begin
  if Self.dir < 0 then begin
    l := TList<TSurfaceNode>.Create;
    l.Add(Self);
  end
  else if (Self.dir = Abs(dir) - 1) then
    l := children[Sign(dir) + 1].GetBoundaryList(dir)
  else begin
    l := TList<TSurfaceNode>.Create;
    for n in children do begin
      t := n.GetBoundaryList(dir);
      l.AddRange(t);
      t.Free;
    end;
  end;
  Result := l;
end;

function TSurfaceNode.GetBoundaryNode(direction: Integer):
TSurfaceNode;
```

```
var
  brother, uncle: TSurfaceNode;
begin
  brother := GetBrother(direction);
  if brother <> nil then
    Result := brother.Reduce(direction, parent.dir, index)
  else if parent = nil then
    Result := nil
  else begin
    uncle := parent.GetBoundaryNode(direction);
    if uncle = nil then
      Result := nil
    else if (uncle.dir < 0) or not uncle.CanReduce(direction,
        parent.dir) then
      Result := uncle
    else
      Result := uncle.Reduce(direction, parent.dir, index);
  end;
end;

function TSurfaceNode.GetBrother(direction: Integer):
TSurfaceNode;
var
  next: Integer;
begin
  if (parent = nil) or (parent.dir <> Abs(direction) - 1) then
    Result := nil
  else begin
    next := index + Sign(direction);
    if (next < 0) or (next > 2) then
      Result := nil
    else
      Result := parent.children[next];
  end;
end;

function TSurfaceNode.IsSquare: Boolean;
var
  i, d: Integer;
```

```
begin
  d := ndiv[0];
  i := 1;
  while (i < dim) and (ndiv[i] = d) do
    Inc(i);
  if i = dim then
    Result := True
  else
    Result := False;
end;

function TSurfaceNode.LongestDimension: Integer;
begin
  Result := ndiv[LongestDimensionIndex];
end;

function TSurfaceNode.LongestDimensionIndex: Integer;
var
  i, min: Integer;
begin
  min := 0;
  for i := 1 to dim - 1 do
    if ndiv[i] < ndiv[min] then
      min := i;
  Result := min;
end;

function TSurfaceNode.Reduce(boundDir, refDir, refIndex: Integer)
  : TSurfaceNode;
begin
  if dir = Abs(boundDir) - 1 then
    Result := children[Sign(-boundDir) + 1].Reduce(boundDir,
        refDir, refIndex)
  else if dir = refDir then
    Result := children[refIndex].Reduce(boundDir, -2, refIndex)
  else
    Result := Self;
end;
```

```
procedure TSurfaceNode.SetSize;
var
  i: Integer;
begin
  for i := 0 to dim - 1 do
    size[i] := EvalSize(ndiv[i]);
end;

function TSurfaceNode.ShortestDimension: Integer;
begin
  Result := ndiv[ShortstDimensionIndex];
end;

function TSurfaceNode.ShortstDimensionIndex: Integer;
var
  i, max: Integer;
begin
  max := 0;
  for i := 1 to dim - 1 do
    if ndiv[i] > ndiv[max] then
      max := i;
  Result := max;
end;

procedure TSurfaceNode.Shrink(dimension: Integer);
begin
  Inc(ndiv[dimension]);
  size[dimension] := EvalSize(ndiv[dimension]);
end;
{ TSurfaceTree }

procedure TSurfaceTree.BeginUpdate;
begin
  WaitForSingleObject(updateSemaphore, INFINITE);
end;

procedure TSurfaceTree.EndUpdate;
begin
  ReleaseSemaphore(updateSemaphore, 1, nil);
```

```
end;

function TSurfaceTree.hasNextUpdate: Boolean;
begin
  Result := (updateNodes.Count > 0);
end;

function TSurfaceTree.nextUpdate: TSurfaceNode;
begin
  Result := updateNodes.Dequeue;
end;

constructor TSurfaceTree.Create(config: TSurfaceConfig);
var
  i, j: Integer;
begin
  critical := TCriticalSection.Create();
  updateSemaphore := CreateSemaphore(nil, 1, 1, 'updateNodes');
  updateNodes := TQueue<TSurfaceNode>.Create;

  Self.config := config;

  if (Length(config.param0) <> Length(config.param1)) then
    raise Exception.Create('Parameter input must have
          then same dimension');
  dim := Length(config.param0) - Length(config.constParams);
  vSort(config.constParams);
  SetLength(isConst, Length(config.param0));
  for i := 0 to Length(config.param0) - 1 do
    isConst[i] := False;
  for i := 0 to Length(config.constParams) - 1 do
    isConst[config.constParams[i]] := True;

  root := TSurfaceNode.CreateRoot(dim);
  root.eval := Evaluate(root.center);

  refineList := TList<TRefineCandidate>.Create;
  InitStats;
end;
```

```
destructor TSurfaceTree.Destroy;
begin
  root.Destroy;
  refineList.Free;
  critical.Free;
  CloseHandle(updateSemaphore);
  updateNodes.Destroy;
  inherited Destroy;
end;

function TSurfaceTree.Evaluate(node: TSurfaceNode): TEvalResult;
begin
  node.eval := Evaluate(node.center);
end;

function TSurfaceTree.Evaluate(coord: TVectorf): TEvalResult;
var
  pout: TVectorf;
  i, icoord, npar, iconst, nconst: Integer;
begin

  pout := GetParamAt(coord);
  Result := config.eval.Evaluate(pout);
end;

function TSurfaceTree.GetIterator: TSurfaceNodeIterator;
begin
  Result := TSurfaceNodeIterator.Create(root);
end;

function TSurfaceTree.GetParamAt(coord: TVectorf): TVectorf;
var
  i, npar, icoord: Integer;
  p0, p1: TVectorf;
begin
  p0 := config.param0;
  p1 := config.param1;
  npar := Length(p0);
```

```
  Result := vZerosf(npar);
  icoord := 0;
  for i := 0 to npar - 1 do
    if isConst[i] then
      Result[i] := p0[i]
    else begin
      Result[i] := p0[i] + (p1[i] - p0[i]) * coord[icoord];
      Inc(icoord);
    end;
end;

procedure TSurfaceTree.InitStats;
begin
  minNode := 0;
  maxNode := 0;
  bndTarget := 1;
  leafNodes := 1;
end;

function TSurfaceTree.IsCandidate(node: TSurfaceNode): Boolean;
var
  c: TRefineCandidate;
begin
  for c in refineList do
    if c.node = node then begin
      Result := True;
      Exit;
    end;
  Result := False;
end;

function TSurfaceTree.NodeAt(coord: TVectorf): TSurfaceNode;
var
  it: TSurfaceNodeIterator;
  n: TSurfaceNode;
  found: Boolean;
begin
  it := GetIterator;
```

```
  found := False;
  while it.HasNext and not found do begin
    n := it.GetNext;
  end;
  if found then
    Result := n
  else
    Result := nil;
end;

procedure TSurfaceTree.SelectCandidates;
begin
  refineList.Clear;
  SelectMinCandidates;
  if refineList.Count = 0 then
    SelectBoundaryCandidates;

  if sort and (refineList.Count > 0) then begin
    refineList.sort(TDelegatedComparer<TRefineCandidate>.Construct(
      function(const Left, Right: TRefineCandidate): Integer
      begin
        Result := Sign(Left.node.LongestDimension -
          Right.node.LongestDimension);
      end));
    if refineList.Count > MAX_REFINE then
      refineList.DeleteRange(MAX_REFINE,
              refineList.Count - MAX_REFINE);
  end;
end;

procedure TSurfaceTree.SelectBoundaryCandidates;
var
  min, max, delta: Integer;
  it: TSurfaceNodeIterator;
  i, dir, long, minBnd: Integer;
  hasBounds: Boolean;
  ev: EvalType;
  n, bnd: TSurfaceNode;
begin
```

```
hasBounds := False;
minBnd := 1000;
it := GetIterator;
while it.HasNext do begin
  n := it.GetNext;
  ev := n.eval.tpe;
  long := n.LongestDimension;
  for i := 0 to dim * 2 - 1 do begin
    dir := ((i shr 1) + 1) * ((i and 1) * 2 - 1);
    bnd := n.GetBoundaryNode(dir);
    if (bnd <> nil) and (bnd.dir < 0) and (bnd.eval.tpe <> ev)
          then begin
      hasBounds := True;
      delta := bnd.DeltaDiv(min, max);
      if (delta = 0) and (min < minBnd) then
        minBnd := min;
      if ((delta <> 0) or (max < bndTarget)) and not
            IsCandidate(bnd) then
        refineList.Add(TRefineCandidate.Create(bnd,
          bnd.LongestDimensionIndex));
    end;
  end;
end;
if hasBounds and (minBnd < 1000) and (minBnd >= bndTarget) and
  (bndTarget < MAX_DIV) then begin
  bndTarget := minBnd + 1;
  if refineList.Count = 0 then
    SelectBoundaryCandidates;
end
else if refineList.Count > 0 then
  refineType := 'Boundary'
else begin
  refineType := 'All';
  it.Reset;
  while it.HasNext do begin
    n := it.GetNext;
    if n.LongestDimension < MAX_DIV then
      refineList.Add(TRefineCandidate.Create(n,
            n.LongestDimensionIndex));
```

```
    end;
  end;
end;

procedure TSurfaceTree.SelectMinCandidates;
var
  it: TSurfaceNodeIterator;
  n: TSurfaceNode;
  delta, min, max: Integer;
begin
  it := GetIterator;
  minNode := 1000;
  maxNode := -1;
  while it.HasNext do begin
    n := it.GetNext;
    if n.dir < 0 then begin
      delta := n.DeltaDiv(min, max);
      if min < minNode then
        minNode := min;
      if max > maxNode then
        maxNode := max;
    end;
  end;

  if maxNode - minNode >= MAX_DELTA then begin
    it.Reset;
    while it.HasNext do begin
      n := it.GetNext;
      delta := n.DeltaDiv(min, max);
      if min <= minNode then
        refineList.Add(TRefineCandidate.Create(n,
                n.LongestDimensionIndex));
      if refineList.Count <> 0 then
        refineType := 'Min';
    end;
  end;
end;

function TSurfaceTree.GetCandidates;
```

```
var
  cand: TRefineCandidate;
begin
  if refineList.Count = 0 then
    SelectCandidates(sort);
  Result := refineList;
end;

procedure TSurfaceTree.Step(n: Integer; sort: Boolean);
var
  i: Integer;
  cand: TRefineCandidate;
begin
  for i := 0 to n - 1 do begin
    if refineList.Count = 0 then
      SelectCandidates(sort);
    if refineList.Count > 0 then begin
      cand := refineList.First;
      refineList.Delete(0);

      cand.Refine;
      Evaluate(cand.node.children[0]);
      Evaluate(cand.node.children[2]);
      UpdateStats(cand.node);
      Inc(leafNodes, 2);
    end;
  end;
end;

procedure TSurfaceTree.Step(candidate: TRefineCandidate);
begin
  candidate.Refine;
  Evaluate(candidate.node.children[0]);
  Evaluate(candidate.node.children[2]);
  critical.Acquire;
  UpdateStats(candidate.node);
  Inc(leafNodes, 2);
  critical.Release;
```

```
    WaitForSingleObject(updateSemaphore, INFINITE);
    updateNodes.Enqueue(candidate.node.children[0]);
    updateNodes.Enqueue(candidate.node.children[1]);
    updateNodes.Enqueue(candidate.node.children[2]);
    ReleaseSemaphore(updateSemaphore, 1, nil);

end;

procedure TSurfaceTree.UpdateStats(node: TSurfaceNode);
var
  n: TSurfaceNode;
  min, max, delta: Integer;
begin
  for n in node.children do begin
    delta := n.DeltaDiv(min, max);
    if max > maxNode then
      maxNode := max;
  end;
end;

{ TSurfaceNodeIterator }

constructor TSurfaceNodeIterator.Create(root: TSurfaceNode);
begin
  Self.root := root;
  Reset;
end;

function TSurfaceNodeIterator.FirstChild(node: TSurfaceNode):
TSurfaceNode;
begin
  if node.dir < 0 then
    Result := node
  else
    Result := FirstChild(node.children[0]);
end;

function TSurfaceNodeIterator.GetNext: TSurfaceNode;
```

```
begin
  Result := it;
  it := NextNode(it);
end;

function TSurfaceNodeIterator.HasNext: Boolean;
begin
  Result := (it <> nil);
end;

function TSurfaceNodeIterator.NextNode(node: TSurfaceNode):
TSurfaceNode;
begin
  if node.parent = nil then
    Result := nil
  else if node.index < 2 then
    Result := FirstChild(node.parent.children[node.index + 1])
  else
    Result := NextNode(node.parent);
end;

procedure TSurfaceNodeIterator.Reset;
begin
  it := FirstChild(root);
end;

{ TSurfaceConfig }

constructor TSurfaceConfig.Create(eval: TEvaluator; param0,
param1: TVectorf;
constParams: TVectori; multithreaded: Boolean);
begin
  Self.eval := eval;
  Self.param0 := param0;
  Self.param1 := param1;
  Self.constParams := constParams;
  Self.multithreaded := multithreaded;
end;
```

```
  { TRefineCandidate }

  constructor TRefineCandidate.Create(node: TSurfaceNode;
  direction: Integer);
  begin
    Self.node := node;
    dir := direction;
  end;

  procedure TRefineCandidate.Refine;
  begin
    node.Divide(dir);
  end;

end.

unit uBase;

interface

  type
    PDoubleArray = ^TDoubleArray;
    TDoubleArray = array [0 .. 32767] of Double;

    PVectorf = ^TVectorf;
    TVectorf = array of Double;
    TVectori = array of Integer;

  function vZerosf(n: Integer): TVectorf;
  function vZerosi(n: Integer): TVectori;
  function vOnesf(n: Integer): TVectorf;
  function vArrayf(const n: array of Double): TVectorf; overload;
  function vArrayi(const n: array of Integer): TVectori; overload;

  function vCopy(v: TVectorf): TVectorf; overload;
  function vCopy(v: TVectori): TVectori; overload;
  procedure vCopy(var vTo: TVectorf; vFrom: TVectorf); overload;
  procedure vCopy(var vTo: TVectori; vFrom: TVectori); overload;
```

```
  function vLen(v: TVectorf): Integer; overload;
  function vLen(v: TVectori): Integer; overload;

  procedure vAdd(v1, v2: TVectorf); overload;
  procedure vSub(v1, v2: TVectorf); overload;
  procedure vMul(v: TVectorf; k: Double); overload;
  procedure vDiv(v: TVectorf; k: Double); overload;

  procedure vSub(var dst: TVectorf; v1, v2: TVectorf); overload;
  procedure vMul(var dst: TVectorf; v: TVectorf; k: Double);overload;

  procedure vMla(dst, mulv: TVectorf; mulk: Double; addv: TVectorf); over

  function vNorm(v: TVectorf): Double; overload;
  function vNormSqr(v: TVectorf): Double; overload;
  function vMax(v: TVectorf): Double; overload;
  function vMin(v: TVectorf): Double; overload;
  procedure vSort(var v: TVectori); overload;

implementation

  uses
    SysUtils, System.Generics.Collections;

  function vZerosf(n: Integer): TVectorf;
  var
    i: Integer;
  begin
    SetLength(Result, n);
    for i := 0 to n - 1 do
      Result[i] := 0;
  end;

  function vZerosi(n: Integer): TVectori;
  var
    i: Integer;
  begin
    SetLength(Result, n);
    for i := 0 to n - 1 do
```

```
    Result[i] := 0;
end;

function vOnesf(n: Integer): TVectorf;
var
  i: Integer;
begin
  SetLength(Result, n);
  for i := 0 to n - 1 do
    Result[i] := 1;
end;

function vArrayf(const n: array of Double): TVectorf;
var
  i, l: Integer;
begin
  l := Length(n);
  SetLength(Result, l);
  for i := 0 to l - 1 do
    Result[i] := n[i];
end;

function vArrayi(const n: array of Integer): TVectori;
var
  i, l: Integer;
begin
  l := Length(n);
  SetLength(Result, l);
  for i := 0 to l - 1 do
    Result[i] := n[i];
end;

function vCopy(v: TVectorf): TVectorf;
begin
  SetLength(Result, Length(v));
  vCopy(Result, v);
end;

function vCopy(v: TVectori): TVectori;
```

```
begin
  SetLength(Result, Length(v));
  vCopy(Result, v);
end;

procedure vCopy(var vTo: TVectorf; vFrom: TVectorf);
var
  i: Integer;
begin
  for i := 0 to Length(vTo) - 1 do
    vTo[i] := vFrom[i];
end;

procedure vCopy(var vTo: TVectori; vFrom: TVectori);
var
  i: Integer;
begin
  for i := 0 to Length(vTo) - 1 do
    vTo[i] := vFrom[i];
end;

function vLen(v: TVectorf): Integer;
begin
  Result := Length(v);
end;

function vLen(v: TVectori): Integer;
begin
  Result := Length(v);
end;

procedure vAdd(v1, v2: TVectorf);
var
  i: Integer;
begin
  for i := 0 to Length(v1) - 1 do
    v1[i] := v1[i] + v2[i];
end;
```

```
procedure vSub(v1, v2: TVectorf);
var
  i: Integer;
begin
  for i := 0 to Length(v1) - 1 do
    v1[i] := v1[i] - v2[i];
end;

procedure vMul(v: TVectorf; k: Double);
var
  i: Integer;
begin
  for i := 0 to Length(v) - 1 do
    v[i] := v[i] * k;
end;

procedure vDiv(v: TVectorf; k: Double);
var
  i: Integer;
begin
  for i := 0 to Length(v) - 1 do
    v[i] := v[i] / k;
end;

procedure vSub(var dst: TVectorf; v1, v2: TVectorf);
var
  i: Integer;
begin
  for i := 0 to Length(v1) - 1 do
    dst[i] := v1[i] - v2[i];
end;

procedure vMul(var dst: TVectorf; v: TVectorf; k: Double);
var
  i: Integer;
begin
  for i := 0 to Length(dst) - 1 do
    dst[i] := v[i] * k;
end;
```

```
procedure vMla(dst, mulv: TVectorf; mulk: Double; addv: TVectorf); over
var
  i: Integer;
begin
  for i := 0 to Length(dst) - 1 do
    dst[i] := addv[i] + mulv[i] * mulk;
end;

function vNorm(v: TVectorf): Double;
begin
  Result := Sqrt(vNormSqr(v));
end;

function vNormSqr(v: TVectorf): Double;
var
  i: Integer;
  m: Double;
begin
  m := 0;
  for i := 0 to Length(v) - 1 do
    m := m + v[i] * v[i];
  Result := m;
end;

function vMax(v: TVectorf): Double;
var
  i: Integer;
  Max: Double;
begin
  Max := v[0];
  for i := 1 to Length(v) - 1 do
    if v[i] > Max then
      Max := v[i];
  Result := Max;
end;

function vMin(v: TVectorf): Double;
var
```

```
    i: Integer;
    Min: Double;
  begin
    Min := v[0];
    for i := 1 to Length(v) - 1 do
      if v[i] < Min then
        Min := v[i];
    Result := Min;
  end;

  procedure vSort(var v: TVectori);
  var
    i, j, k, n: Integer;
  begin
    n := Length(v);
    for i := 0 to n - 2 do begin

      k := i;
      for j := i + 1 to n - 1 do
        if v[j] < v[k] then
          k := j;

      j := v[k];
      v[k] := v[i];
      v[i] := j;
    end;
  end;

end.

unit uEquation;

interface

  uses
    uBase, Math;

  const
    NUM_LIN_DELTA = 1E-8;
```

```
type
  TEquation = class
  public
    hasLinearization, useLinearization: Boolean;
    startCondition: TVectorf;
    stateDim, parameterDim: Integer;


    constructor Create(stateDim, parameterDim: Integer;
      useLinearization: Boolean; startCondition: TVectorf);

    procedure Derive(var state, params, derivs: TVectorf;
      useLinearization: Boolean); virtual; abstract;
    procedure NumericLinearization(var state, params, derivs,
        tmp1, tmp2, tmp3, tmp4: TVectorf);
  end;

  TLorenz = class(TEquation)
    constructor Create(useLinearization: Boolean);
    procedure Derive(var state, params, derivs: TVectorf;
      useLinearization: Boolean); override;
  end;

  TLorenzLin = class(TEquation)
    constructor Create;
    procedure Derive(var state, params, derivs: TVectorf;
      useLinearization: Boolean); override;
  end;

  TExothermicReactor = class(TEquation)
    constructor Create(useLinearization: Boolean);
    procedure Derive(var state, params, derivs: TVectorf;
      useLinearization: Boolean); override;
  end;

  TBiologicalReactor = class(TEquation)
    constructor Create(useLinearization: Boolean);
    procedure Derive(var state, params, derivs: TVectorf;
```

```
      useLinearization: Boolean); override;
  end;

  TPolimerization = class(TEquation)
    constructor Create(useLinearization: Boolean);
    procedure Derive(var state, params, derivs: TVectorf;
      useLinearization: Boolean); override;
  end;

implementation

  uses
    SysUtils;

  { TEquation }

  constructor TEquation.Create(stateDim, parameterDim: Integer;
    useLinearization: Boolean; startCondition: TVectorf);
  begin
    if not useLinearization then
      Self.stateDim := stateDim
    else
      Self.stateDim := stateDim + stateDim * stateDim;

    Self.parameterDim := parameterDim;
    Self.startCondition := startCondition;
    Self.useLinearization := useLinearization;
    hasLinearization := False;
  end;

  procedure TEquation.NumericLinearization(var state, params,
  derivs, tmp1, tmp2, tmp3, tmp4: TVectorf);
  var
    i, j, k, index, n, max: Integer;
    delta, t: Double;
  begin
    n := Length(startCondition);
    Derive(state, params, derivs, False);
    max := 8;
```

```
delta := NUM_LIN_DELTA;

for i := 0 to n - 1 do
  for j := 0 to n - 1 do
    derivs[(j + 1) * n + i] := 0;

while max > 0 do
  try

    for i := 0 to n - 1 do begin

      {
        for j := 0 to n - 1 do
        tmp1[j] := state[j];
        tmp1[i] := tmp1[i] + delta;
        Derive(tmp1, params, tmp3, False);
        for j := 0 to n - 1 do
        tmp2[j] := (tmp3[j] - derivs[j]) / delta;

      for j := 0 to n - 1 do begin
        tmp1[j] := state[j];
        tmp2[j] := state[j];
      end;
      tmp1[i] := tmp1[i] - delta;
      tmp2[i] := tmp2[i] + delta;

      Derive(tmp1, params, tmp3, False);
      Derive(tmp2, params, tmp4, False);

      for j := 0 to n - 1 do
        tmp2[j] := (tmp4[j] - tmp3[j]) / (2 * delta);
      }
      for j := 0 to n - 1 do
        for k := 0 to n - 1 do
          derivs[(j + 1) * n + k] := derivs[(j + 1) * n + k] +
            state[(i + 1) * n + k] * tmp2[j];
    end;

    for j := 0 to n - 1 do
```

```
          for k := 0 to n - 1 do begin
            t := derivs[(j + 1) * n + k];
            if Abs(t) > 1E8 then

              derivs[(j + 1) * n + k] := Sign(t) * 1E8;
          end;

        Break;
      except
        delta := delta / 10;
        Dec(max);
      end;
    if max = 0 then
      raise Exception.Create('Could not find suitable step');
end;

{ TLorenz }

constructor TLorenz.Create;
begin
  inherited Create(3, 3, useLinearization, vArrayf([1, 1, 1]));
end;

procedure TLorenz.Derive(var state, params, derivs: TVectorf;
  useLinearization: Boolean);
begin

  derivs[0] := params[0] * (state[1] - state[0]);

  derivs[1] := state[0] * (params[1] - state[2]) - state[1];

  derivs[2] := state[0] * state[1] - params[2] * state[2];
end;

{ TPolimer }

constructor TExothermicReactor.Create;
begin
  inherited Create(2, 5, useLinearization, vArrayf([0.3, 4]));
```

```
end;

procedure TExothermicReactor.Derive(var state, params,
derivs: TVectorf; useLinearization: Boolean);
var
  k: Double;
begin

  k := (1 - state[0]) * Exp(state[1]/(1 + state[1]/params[3]));

  derivs[0] := -state[0] + params[2] * k;
  derivs[1] := -state[1] + params[2] * params[0] * k -
      params[1] * (state[1] - params[4]);


end;

{ TBiologicalReactorReactor }

constructor TBiologicalReactor.Create;
begin
  inherited Create(2, 3, useLinearization, vArrayf([0.24, 75]));
end;

procedure TBiologicalReactor.Derive(var state, params,
derivs: TVectorf; useLinearization: Boolean);
var
  k, a, b, da, x1, x2: Double;
begin
  x1 := state[0];
  x2 := state[1];

  a := params[0];
  b := params[1];
  da := params[2];

  k := 1 + a - x2;

  derivs[0] := -x1 + da * ((1 + a) * (1 - x2) * x1) / k;
```

```
  derivs[1] := -x2 + da * ((1 + a) * (1 + b) * (1 - x2) * x1) /
    (k * (1 + b - x2));
end;

{ TPolimerization }

constructor TPolimerization.Create;
begin
  inherited Create(3, 10, useLinearization, vArrayf([0.2, 0.8,
      25 + 273.15]));
end;

procedure TPolimerization.Derive(var state, params,
derivs: TVectorf; useLinearization: Boolean);
var
  D, alfa, a, Lambda, DE1, DE2, B1, G, T0, e, Bo: Double;
  m, x0, x1, x2: Double;
  R0, R1, Q, g_, Teb: Double;

begin

  D := params[0];
  alfa := params[1];
  a := params[2];
  Lambda := params[3];
  DE1 := params[4];
  DE2 := params[5];
  B1 := params[6];
  G := params[7];
  T0 := params[8];
  e := params[9];
  Bo := -B1 * G * Exp(D);


  x0 := state[0];
  x1 := state[1];
  x2 := state[2];
```

```
  if (x1 < 0) then
    g_ := 1E10 - 5
  else
    g_ := Power((1 - (1 - x1) * alfa), 0.5 * Lambda);

  if (x0 < 0) then
    R0 := 0
  else
    R0 := x0 * Exp(D - DE2 / x2);

  if (x0 < 0) or (x1 < 0) then
    R1 := 0
  else
    R1 := x1 * Sqrt(x0) * Exp(D + a - DE1 / x2) / g_;


  Teb := 125 + 273.15;
  if (x2 > Teb) then
    Q := 1E4 * Abs(Teb - x2)
  else
    Q := 0;


  m := (1 - x1);
  derivs[0] := 1 - (1 - e * m) * x0 - R0;
  derivs[1] := 1 - (1 - e * m) * x1 - R1;
  derivs[2] := ((1 + Bo) * (T0 - x2) - alfa * G * R1) *
      (1 - e * m) - Q;
end;

{ TLorenzLin }

constructor TLorenzLin.Create;
begin
  inherited Create(3, 3, True, vArrayf([1, 1, 1]));
  hasLinearization := True;
end;
```

```
  procedure TLorenzLin.Derive(var state, params,
  derivs: TVectorf; useLinearization: Boolean);
  var
    i: Integer;
    eta1, eta2, eta3: Double;
  begin

    derivs[0] := params[0] * (state[1] - state[0]);

    derivs[1] := state[0] * (params[1] - state[2]) - state[1];

    derivs[2] := state[0] * state[1] - params[2] * state[2];

    if useLinearization then
      for i := 0 to 2 do begin
        eta1 := state[3 + i];
        eta2 := state[6 + i];
        eta3 := state[9 + i];
        derivs[3 + i] := params[0] * (eta2 - eta1);
        derivs[6 + i] := ((params[1] - state[2]) * eta1 - eta2 -
                eta3 * state[0]);
        derivs[9 + i] := (state[1] * eta1 + state[0] * eta2 -
                params[2] * eta3);
      end;
  end;

end.

unit uIntegrator;

interface

  uses
    uEquation, uBase;

  type
    TIntegratorInstance = class
      valid: Boolean;
      equation: TEquation;
```

```
    useLinearization: Boolean;
    t: Double;
    tries:Integer;
    state, params: TVectorf;
    workAreaf: array of TVectorf;
    workAreai: array of TVectori;
    tmp1, tmp2, tmp3, tmp4: TVectorf;
  end;

  TIntegrator = class
    discreteStep: Boolean;
    useLinearization: Boolean;

    procedure CopyInstanceState(iFrom, iTo: TIntegratorInstance);
    function CreateInstance(equation: TEquation; params: TVectorf)
      : TIntegratorInstance; overload; virtual;
    function CreateInstance(ref: TIntegratorInstance):
        TIntegratorInstance;
      overload; virtual;
    procedure IntegrateTime(var inst: TIntegratorInstance;
        tout: Double);
      virtual; abstract;
    procedure IntegrateStep(var inst: TIntegratorInstance);
        virtual; abstract;
  end;

  TRungeKutta = class(TIntegrator)
    delta: Double;
    constructor Create(delta: Double; useLinearization: Boolean);

    function CreateInstance(equation: TEquation; params: TVectorf)
      : TIntegratorInstance; override;
    procedure IntegrateTime(var inst: TIntegratorInstance;
      tout: Double); override;
    procedure IntegrateStep(var inst: TIntegratorInstance);
        override;

  end;
```

```
implementation

  uses
    SysUtils, Winapi.Windows;

  { TRungeKutta }

  constructor TRungeKutta.Create;
  begin
    Self.delta := delta;
    Self.useLinearization := False;
    discreteStep := True;
  end;

  function TRungeKutta.CreateInstance(equation: TEquation;
  params: TVectorf) : TIntegratorInstance;
  var
    i: Integer;
    inst: TIntegratorInstance;
  begin
    inst := inherited CreateInstance(equation, params);

    SetLength(inst.workAreai, 0);
    SetLength(inst.workAreaf, 5);
    for i := 0 to 4 do
      inst.workAreaf[i] := vZerosf(vLen(inst.state));

    Result := inst;
  end;

  procedure TRungeKutta.IntegrateStep(var inst:
  TIntegratorInstance);
  var
    i, n: Integer;
    equ: TEquation;
    st, tmp, k1, k2, k3, k4, pars: TVectorf;
  begin
    with inst do begin
      equ := equation;
```

```
    st := state;
    n := Length(state) - 1;
    pars := inst.params;
    k1 := workAreaf[0];
    k2 := workAreaf[1];
    k3 := workAreaf[2];
    k4 := workAreaf[3];
    tmp := workAreaf[4];
  end;

  for i := 0 to n do
    tmp[i] := st[i];
  equ.Derive(tmp, pars, k1, inst.useLinearization);

  for i := 0 to n do
    tmp[i] := st[i] + delta * k1[i] / 2;
  equ.Derive(tmp, pars, k2, inst.useLinearization);

  for i := 0 to n do
    tmp[i] := st[i] + delta * k2[i] / 2;
  equ.Derive(tmp, pars, k3, inst.useLinearization);

  for i := 0 to n do
    tmp[i] := st[i] + delta * k3[i];
  equ.Derive(tmp, pars, k4, inst.useLinearization);

  inst.t := inst.t + delta;
  for i := 0 to n do
    st[i] := st[i] + delta * (k1[i] + 2 * k2[i] + 2 * k3[i]
        + k4[i]) / 6;
end;

procedure TRungeKutta.IntegrateTime(var inst:
TIntegratorInstance; tout: Double);
begin
  while inst.t < tout do
    IntegrateStep(inst);
end;
```

```
{ TIntegrator }

procedure TIntegrator.CopyInstanceState(iFrom, iTo:
TIntegratorInstance);
var
  i, n: Integer;
begin
  iTo.t := iFrom.t;
  vCopy(iTo.state, iFrom.state);
end;

function TIntegrator.CreateInstance(ref: TIntegratorInstance)
  : TIntegratorInstance;
var
  inst: TIntegratorInstance;
  i, n: Integer;
begin
  inst := TIntegratorInstance.Create;

  inst.valid := ref.valid;
  inst.equation := ref.equation;
  inst.useLinearization := ref.useLinearization;
  inst.t := ref.t;
  inst.state := vCopy(ref.state);
  inst.params := vCopy(ref.params);

  n := Length(ref.workAreaf);
  SetLength(inst.workAreaf, n);
  for i := 0 to n - 1 do
    inst.workAreaf[i] := vCopy(ref.workAreaf[i]);

  n := Length(ref.workAreai);
  SetLength(inst.workAreai, n);
  for i := 0 to n - 1 do
    inst.workAreai[i] := vCopy(ref.workAreai[i]);

  Result := inst;
end;
```

```
  function TIntegrator.CreateInstance(equation: TEquation;
  params: TVectorf) : TIntegratorInstance;
  var
    i, j, n: Integer;
    inst: TIntegratorInstance;
  begin
    if (vLen(params) <> equation.parameterDim) then
      raise Exception.Create('Invalid state or parameter size');

    inst := TIntegratorInstance.Create;
    inst.valid := True;
    inst.equation := equation;
    inst.useLinearization := equation.useLinearization;
    inst.t := 0;
    if (not equation.useLinearization) then
      inst.state := vCopy(equation.startCondition)
    else begin
      n := Length(equation.startCondition);
      inst.state := vZerosf(equation.stateDim);
      for j := 0 to n - 1 do
        inst.state[j] := equation.startCondition[j];
      for j := 0 to n - 1 do
        inst.state[(j + 1) * n + j] := 1;
      inst.tmp1 := vZerosf(n);
      inst.tmp2 := vZerosf(n);
      inst.tmp3 := vZerosf(n);
      inst.tmp4 := vZerosf(n);
    end;
    inst.params := vCopy(params);

    Result := inst;
  end;

end.

unit uDassl;

interface
```

```
uses
  uBase, uIntegrator, uEquation;

type

  TDassl = class(TIntegrator)
    delta, rtol, atol: Double;
    constructor Create(delta, error: Double;
        useLinearization: Boolean);
    procedure IntegrateTime(var inst: TIntegratorInstance;
      tout: Double); override;
    procedure IntegrateStep(var inst: TIntegratorInstance);
        override;
    function CreateInstance(equation: TEquation;
        params: TVectorf) : TIntegratorInstance; override;
    function CreateInstance(ref: TIntegratorInstance)
      : TIntegrator
  end;

  SystemRes = procedure(
    var t: Double;
    y: PDoubleArray;
    yprime: PDoubleArray;
    delta: PDoubleArray;
    var ires: Integer;
    rpar: PDoubleArray;
    ipar: PIntegerArray
    ); cdecl;

  Jacobian = procedure(
    var t: Double;
    y: PDoubleArray;
    yprime: PDoubleArray;
    pd: PDouble;
    var cj: Double;
    rpar: PDoubleArray;
    ipar: PIntegerArray
    ); cdecl;
```

```
  procedure ort_vet_(
    var nd: Integer;
    v_y: PDoubleArray;
    v_x: PDoubleArray
    ); cdecl; external 'dassl.dll';

  procedure ddassl_(
    res: SystemRes;
    { }
    var neq: Integer;
    var t: Double;
    { }
    y: PDoubleArray;
    yprime: PDoubleArray;
    { }
    var tout: Double;
    info: PIntegerArray;
    var rtol: Double;
    var atol: Double;
    var idid: Integer;
    { }
    rwork: PDoubleArray;
    var lrw: Integer;
    iwork: PIntegerArray;
    var liw: Integer;
    { }
    rpar: PDoubleArray;
    ipar: PIntegerArray;
    { }
    jac: Jacobian
    ); cdecl; external 'dassl.dll';

implementation

  uses
    System.SysUtils;

  procedure res(
    var t: Double;
```

```
  y: PDoubleArray;
  yprime: PDoubleArray;
  delta: PDoubleArray;
  var ires: Integer;
  { }
  rpar: PDoubleArray;
  ipar: PIntegerArray
  ); cdecl;
var
  i, err: Integer;
  equ: TEquation;
  inst: TIntegratorInstance;
begin
  inst := TIntegratorInstance(Pointer(ipar[0]));
  equ := inst.equation;

  err := 0;

  equ.Derive(TVectorf(y), TVectorf(rpar), TVectorf(delta),
    inst.useLinearization);

  if inst.useLinearization and not equ.hasLinearization then
    equ.NumericLinearization(TVectorf(y), TVectorf(rpar),
        TVectorf(delta), inst.tmp1, inst.tmp2, inst.tmp3, inst.tmp4);

  ires := 0;
  for i := 0 to equ.stateDim - 1 do begin
    delta[i] := yprime[i] - delta[i];

  end;

end;

procedure jac(
  var t: Double;
  y: PDoubleArray;
  yprime: PDoubleArray;
  pd: PDouble;
  var cj: Double;
```

```
    rpar: PDoubleArray;
    ipar: PIntegerArray
    ); cdecl;
begin

end;

{ TDassl }

constructor TDassl.Create(delta, error: Double;
useLinearization: Boolean);
begin
    Self.delta := delta;
    Self.useLinearization := True;
    discreteStep := False;

    rtol := error;
    atol := error;
end;

function TDassl.CreateInstance(equation: TEquation;
params: TVectorf) : TIntegratorInstance;
var
    i, n, ires: Integer;
    inst: TIntegratorInstance;
    delta: TVectorf;
    lin: Boolean;
begin
    inst := inherited CreateInstance(equation, params);
    n := equation.stateDim;

    SetLength(inst.workAreai, 3);
    inst.workAreai[0] := vArrayi([Integer(Pointer(inst))]);
    inst.workAreai[1] := vZerosi(15);
    inst.workAreai[2] := vZerosi(n + 20);

    SetLength(inst.params, vLen(params));

    SetLength(inst.workAreaf, 2);
```

```
  inst.workAreaf[0] := vZerosf(n);
  inst.workAreaf[1] := vZerosf(40 + 9 * n + n * n);

  inst.useLinearization := False;
  delta := vZerosf(n);
  res(inst.t, @inst.state[0], @inst.workAreaf[0][0], @delta[0],
      ires,
    @inst.params[0], @inst.workAreai[0][0]);

  for i := 0 to n - 1 do
    inst.workAreaf[0][i] := -delta[i];

  res(inst.t, @inst.state[0], @inst.workAreaf[0][0], @delta[0],
      ires,
    @inst.params[0], @inst.workAreai[0][0]);

  Result := inst;
end;

function TDassl.CreateInstance(ref: TIntegratorInstance):
TIntegratorInstance;
var
  inst: TIntegratorInstance;
begin
  inst := inherited CreateInstance(ref);
  inst.workAreai[0][0] := Integer(Pointer(inst));
  Result := inst;
end;

procedure TDassl.IntegrateStep(var inst: TIntegratorInstance);
begin
  IntegrateTime(inst, inst.t + delta);
end;

procedure TDassl.IntegrateTime(var inst: TIntegratorInstance;
tout: Double);
var
  idid, n, neq, liw, lrw, i: Integer;
  ti: Double;
```

```
    equ: TEquation;
  begin
    equ := inst.equation;
    n := vLen(inst.state);
    neq := vLen(equ.startCondition);
    lrw := vLen(inst.workAreaf[1]);
    liw := vLen(inst.workAreai[2]);
    ti := inst.t;

    with inst do begin
      for i := 0 to 14 do
        workAreai[1][i] := 0;
    end;

    ddassl_(
      res,
      n,
      inst.t,
      @inst.state[0],
      @inst.workAreaf[0][0],
      tout,
      @inst.workAreai[1][0],
      rtol, atol, idid,
      @inst.workAreaf[1][0],
      lrw,
      @inst.workAreai[2][0],
      liw,
      @inst.params[0],
      @inst.workAreai[0][0],
      jac);

  end;

end.

unit SurfaceThread;

interface
```

```
  uses
    uSurface, System.Classes, System.Generics.Collections,
    System.Generics.Defaults, EvaluationThread, uThreadMessages;

  const
    QUEUE_SIZE = 10;

  type
    TSurfaceThread = class(TThread)
    private
      { Private declarations }
    public
      halted: Boolean;
      evalCount: Integer;
      evalArray: array of TEvaluationThread;
      actives: Integer;
      list: TList<TRefineCandidate>;
      procedure Execute; override;
      procedure StartCycle;
      procedure SendData(thr: TEvaluationThread);
    public
      surface: TSurfaceTree;
      constructor Create(config: TSurfaceConfig);
      procedure AskTerminate;
      function LockSurface: TSurfaceTree;
      procedure ReleaseSurface;

      class function CpuCores: Integer;
    end;

implementation

  uses
    Winapi.Windows;

  { TSurfaceThread }

  constructor TSurfaceThread.Create(config: TSurfaceConfig);
  var
```

```
  i: Integer;
  msg: tagMsg;
begin
  inherited Create(False);
  PeekMessage(msg, 0, 0, 0, PM_NOREMOVE);

  halted := False;
  surface := TSurfaceTree.Create(config);
  evalCount := CpuCores();
end;

procedure TSurfaceThread.Execute;
var
  i, n: Integer;
  halting: Integer;
  thr: TEvaluationThread;
  msg: tagMsg;
  cand: TRefineCandidate;
  done: Boolean;
begin
  SetLength(evalArray, evalCount);
  for i := 0 to evalCount - 1 do begin
    evalArray[i] := TEvaluationThread.Create(i, ThreadID);
    evalArray[i].surface := surface;
    SetThreadAffinityMask(evalArray[i].Handle, 1 shl i);
  end;
  actives := evalCount;
  while actives > 0 do begin
    GetMessage(msg, 0, 0, 0);
    if ThreadMessage(msg.message) = TM_READY then
      Dec(actives);
  end;
  halting := -1;
  list := nil;
  StartCycle;
  while GetMessage(msg, 0, 0, 0) do begin
    case ThreadMessage(msg.message) of
      TM_DONE: begin
        Dec(actives);
```

```
        if halting < 0 then begin
          thr := evalArray[msg.lParam];
          if (list.Count > 0) then
            SendData(thr)
          else begin
            if (actives = 0) then
              StartCycle;
          end;
        end;
      end;
      TM_HALT: begin
        halting := evalCount;
        for i := 0 to evalCount - 1 do
          PostThreadMessage(evalArray[i].ThreadID,
                      Integer(TM_HALT), 0, 0);
      end;
      TM_HALTED: begin
        Dec(halting);
        if (halting = 0) then
          halted := True;
      end;
      TM_CONTINUE: begin
        halted := False;
        halting := -1;
        n := list.Count;
        StartCycle;
      end;
      TM_TERMINATE: break;
    end;
  end;

  for i := 0 to evalCount - 1 do
    PostThreadMessage(evalArray[i].ThreadID,
        Integer(TM_TERMINATE), 0, 0);
  for i := 0 to evalCount - 1 do begin
    evalArray[i].WaitFor;
    evalArray[i].Destroy;
  end;
```

```
  surface.Free;
end;

procedure TSurfaceThread.SendData(thr: TEvaluationThread);
var
  candidate: PRefineCandidate;
begin
  Inc(actives);
  New(candidate);
  candidate^ := list.First;
  list.Delete(0);
  PostThreadMessage(thr.ThreadID, Integer(TM_DATA),
      Cardinal(candidate), 0);
end;

procedure TSurfaceThread.StartCycle;
var
  i, n: Integer;
begin
  if (list = nil) or (list.Count = 0) then
    list := surface.GetCandidates(False);
  n := list.Count;
  if (n > evalCount * QUEUE_SIZE) then
    n := evalCount * QUEUE_SIZE;

  for i := 0 to n - 1 do
    SendData(evalArray[i mod evalCount]);
end;

procedure TSurfaceThread.AskTerminate;
begin
  PostThreadMessage(ThreadID, Integer(TM_TERMINATE), 0, 0);
end;

function TSurfaceThread.LockSurface: TSurfaceTree;
begin
  PostThreadMessage(ThreadID, Integer(TM_HALT), 0, 0);
  while not halted do
    Sleep(10);
```

```
    Result := surface;
  end;

  procedure TSurfaceThread.ReleaseSurface;
  begin
    PostThreadMessage(ThreadID, Integer(TM_CONTINUE), 0, 0);
  end;

  class function TSurfaceThread.CpuCores: Integer;
  var
    SysInfo: TSystemInfo;
    HyperThread: Integer;
  begin
    GetSystemInfo(SysInfo);
    HyperThread := 0;
    if SysInfo.dwNumberOfProcessors > 1 then begin
      asm
        CPUID
        test edx, $800000
        jz @@Not_HT_Support
        mov HyperThread , 1
      @@Not_HT_Support:
      end;
    end;
    Result := SysInfo.dwNumberOfProcessors;
    if Boolean(HyperThread) then
      Result := Result div 2;
  end;

end.

unit EvaluationThread;

interface

  uses
    uBase, System.Classes, uEvaluator, Winapi.Windows,
        Winapi.Messages, uThreadMessages, uSurface;
```

```
type

  TEvaluationThread = class(TThread)
  private
    { Private declarations }
  protected
    procedure Execute; override;
  public
    index: Cardinal;
    parent: Cardinal;
    surface: TSurfaceTree;
  public
    constructor Create(index, parentId: Cardinal);
  end;

implementation

  { EvaluationThread }

  procedure TEvaluationThread.Execute;
  var
    msg: tagMsg;
    candidate: PRefineCandidate;
  begin
    PostThreadMessage(parent, Cardinal(TM_READY), 0, index);
    while GetMessage(msg, 0, 0, 0) do
      case ThreadMessage(msg.message) of
        TM_TERMINATE: break;
        TM_HALT:
          PostThreadMessage(parent, Cardinal(TM_HALTED), 0, index);
        TM_DATA: begin
          candidate := Pointer(msg.wParam);
          surface.Step(candidate^);
          Dispose(candidate);
          PostThreadMessage(parent, Cardinal(TM_DONE), 0, index);
        end;
      end;
  end;
```

```
  constructor TEvaluationThread.Create;
  var
    msg: tagMsg;
  begin
    inherited Create(False);
    PeekMessage(msg, 0, 0, 0, PM_NOREMOVE);

    Self.index := index;
    parent := parentId;
  end;

end.
```