Universidade do Minho
Escola de Engenharia

Orlando Ricardo Nunes Rocha

Desenvolvimento de ferramentas computacionais para a optimização de processos de fermentação em Biotecnologia

Setembro de 2009

Sistemas Espaciais de Apoio à Decisão – Suporte às Actividades de Negócio
Orlando Ricardo Nunes Rocha
UMinho | 2009

Universidade do Minho
Escola de Engenharia

Orlando Ricardo Nunes Rocha

Desenvolvimento de ferramentas computacionais para a optimização de processos de fermentação em Biotecnologia

Tese de Mestrado em Informática

Trabalho efectuado sob a orientação do
Doutor Miguel Rocha
e da
Doutora Isabel Rocha

Setembro de 2009

Agradecimentos

Após concluir esta tese não posso deixar de agradecer a todos aqueles que me incentivaram e apoiaram ao longo deste trabalho.

Primeiro queria agradecer aos meus pais que sempre me apoiaram em todas as minhas decisões e estão sempre presentes nos momentos difíceis.

Aos meus orientadores o Doutor Miguel Rocha e a Doutora Isabel Rocha pela disponibilidade e apoios prestados ao longo da elaboração desta dissertação.

Ao Nuno Faria e à Sónia, pelo contínuo incentivo durante a realização deste trabalho.

A todos os meus amigos em geral que directamente ou indirectamente me apoiaram em todas as situações e permitiram a conclusão deste trabalho.

Sumário

Actualmente, uma larga variedade de produtos tais como antibióticos, proteínas, vacinas e outros compostos químicos são produzidos através de processos fermentativos. Devido à subida dos preços do petróleo e aos fortes incentivos por parte das instituições para substituir os produtos derivados de petróleo por "produtos verdes", muitos dos processos tradicionais têm vindo a ser substituídos por bioprocessos. Consequentemente, tem existido um esforço para melhorar a produtividade dos processos biológicos. A optimização destes processos pode ser realizada em duas etapas: primeiramente, faz-se uma selecção e uma melhoria genética do microrganismo e num segundo passo são identificadas as melhores condições para realizar o processo fermentativo. Nesta etapa, normalmente são realizados estudos experimentais através de tentativa-erro para obter as condições ambientais que propiciem o melhor crescimento e produtividade do microrganismo, manipulando as concentrações iniciais dos nutrientes, os perfis de alimentação de substrato ao reactor, os modos de operação, bem como a temperatura e o pH.

Nos últimos anos, têm sido desenvolvidas várias ferramentas informáticas para simulação e optimização de bioprocessos. Porém, a maioria destas ferramentas está direccionada para estudar as vias metabólicas de um microrganismo de modo a optimizar a produtividade de determinado produto. Numa fase posterior, é efectuada uma optimização genética do microrganismo. Apesar de existir uma grande variedade de ferramentas informáticas verifica-se que nenhuma delas está desenhada especificamente para a optimização e simulação de processos fermentativos. Assim, o objectivo deste trabalho foi desenvolver de raiz uma ferramenta direccionada para simulação, optimização e estimação de parâmetros de processos fermentativos.

A aplicação *OptFerm* foi desenvolvida sobre uma plataforma denominada *AIBench*, tendo-se utilizado a linguagem Java como linguagem de programação. O *OptFerm* foi então desenvolvido de modo a ser uma ferramenta de fácil uso, extensível e que pudesse funcionar em qualquer sistema operativo, estando disponível como software livre em http://darwin.di.uminho.pt/optferm/. A aplicação foi desenhada de modo a que o utilizador pudesse realizar várias tarefas de simulação, optimização e estimação de parâmetros com diferentes condições no que se refere a variáveis de estado, parâmetros, perfis de alimentação, etc.. As tarefas de optimização foram focadas na determinação do melhor perfil de alimentação de uma corrente de substrato a alimentar ao reactor, dos melhores valores das variáveis de estado para iniciar uma fermentação e do tempo óptimo de duração para uma fermentação.

Foram realizados alguns estudos de optimização e estimação de parâmetros com o objectivo de verificar se a aplicação era suficientemente robusta. Os estudos foram baseados na repetição das experiências por 30 vezes para obter significância estatística. Após os estudos, verificou-se que as operações foram realizadas levando a resultados coerentes, não tendo sido detectados erros relevantes.

Abstract

Nowadays, several products such as antibiotics, proteins, vaccines, amino-acids and other chemicals are produced using fermentation processes. Due to the rise of petroleum prices and the strong incentive to replace petroleum derivatives by “green products”, many traditional processes have been replaced by new biotechnological ones. Consequently, an effort to improve biotechnological techniques has been undertaken. In order to optimize the productivity of a biological process, in the majority of the cases, two different steps have to be addressed: firstly, a selection and genetic improvement of the microbial strain is accomplished; in a second step, the best conditions for the fermentation process are identified, such as the initial nutrient concentrations, operating modes, feeding profiles for fed-batch fermentations, temperature and pH.

Over the last few years, several tools have been developed for the simulation and optimization of biological processes. However, the majority of these tools was designed specifically to study metabolic pathways with the aim of increasing the productivity of a certain product. In a subsequent stage, a genetic optimization of the organism is conducted. Although there are several tools available to study simulate and optimize cellular pathways, there is still a clear lack of specific tools to perform the optimization of fermentation processes. Therefore, the aim of this work was to develop a specific computational tool to perform simulation and optimization of fermentation processes and estimation of unknown parameters.

The *OptFerm* software was developed using the Java programming language, with the aim of being a user-friendly, extensible and platform-independent computational tool. OptFerm is freely available in http://darwin.di.uminho.pt/optferm/. The tool was designed in order to allow the user to evaluate and compare several different methods for the tasks of simulation, optimization and parameter estimation, in the context of fermentation processes. The aim is to allow users to improve process productivity, achieving better results in reduced times. The optimization tasks available include the optimization of a substrate feeding trajectory, of the feeding trajectory plus initial conditions or of the feeding trajectory plus the final fermentation time.

After developing the OptFerm tool, some studies on optimization and parameter estimation were performed, with the aim of verifying if the tool is sufficiently robust. The studies were based on repeating 30 times each type of experiment. It was verified that the operations were performed with coherent results and no relevant errors have been detected.

Índice

Lista de tabelas

Lista de figuras

Lista de símbolos

Abreviaturas

OCDE	Organização para a Cooperação e Desenvolvimento Económico
MVC	*Model-View-Controller*
STB	*Stirred Tank Bioreactor*
SBML	*System Biology Markup Language*
XML	*Extensible Markup Language*
JRE	*Java Runtime Environment*
IMEX	Método Implícito-Explícito de Runge-Kutta
ERK	Método de Runge-Kutta de passo linear constante
GUI	*Graphical user interface*
CellML	*Cell Modelling Markup Language*

Letras Latinas

C_n	variável de estado *n* (concentração do componente n)	
D	taxa de diluição	(T^{-1})
F	vector de caudais volumétricos de entradas líquidas no reactor	$(MT^{-1}L^{-3})$
F_i	factor de interpolação	
F_{in}	caudal volumétrico de alimentação	$(M\ T^{-1})$
F_{out}	caudal volumétrico de saída do reactor	$(M\ T^{-1})$
K_s	parâmetro de saturação	
Q	vector de caudais mássicos de saídas gasosas do reactor	$(M\ L^{-3}T^{-1})$
S	concentração de substrato	$(M\ L^{-3})$
S_{in}	concentração de substrato na alimentação	$(M\ L^{-3})$
W	peso ou volume dentro do reactor	(M); (L)
X	concentração de biomassa	$(M\ L^{-3})$
Z	igual a *-1* se o componente ξ_i é consumido ou 1 se for produzido	
k_{ij}	coeficientes de rendimento ou coeficientes estequiométricos do componente *i* na reacção *j*;	
k_i	coeficientes estequiométricos ou de rendimento	
k_d	taxa específica de morte do microrganismo	(T^{-1})
n	tamanho da população	

Letras gregas

ξ_i	variável de estado (concentração do componente *i*)	(ML^{-3})
φ_j	taxa cinética da reacção *j*	
μ_{max}	taxa específica máxima de crescimento	(T^{-1})
μ_b	taxas cinéticas das reacções	
μ	taxa específica de crescimento do microrganismo	(T^{-1})

Capítulo 1

Introdução

Neste capítulo, faz-se o enquadramento deste trabalho relativamente aos diferentes campos da Biotecnologia. É feita uma pequena exposição do estado da arte, relativamente ao uso de ferramentas informáticas em Biotecnologia. São ainda apresentados os objectivos na realização deste trabalho.

O capítulo está dividido da seguinte forma:

1.1. Contexto e motivação

Ao longo da história, os microrganismos têm tido um enorme impacto na alimentação e no sistema económico da humanidade. Na antiguidade, muitos produtos eram produzidos através de bioprocessos com total desconhecimento dos nossos antepassados, sendo os casos mais conhecidos a produção de vinho e o fabrico de pão. Porém, só por volta do século XIX é que a humanidade começou a entender os mecanismos biológicos que justificavam estes processos. Desde então, o Homem começou a controlar vários processos mediados por microrganismos e a desenvolver técnicas para melhorar a sua rentabilidade.

Em meados dos anos 1970, com o desenvolvimento de novas técnicas de manipulação do DNA, surgiram novos ramos associados à Biotecnologia. Em 1992, na conferência de Ambiente e Desenvolvimento das Nações Unidas, a Biotecnologia foi designada como "Um conjunto de técnicas que utilizam sistemas biológicos, organismos vivos, ou derivados com o objectivo de produzir ou modificar produtos ou processos para um uso específico" [1]. Dez anos mais tarde, a OCDE (Organização para a Cooperação e Desenvolvimento Económico) definiu a Biotecnologia como "a aplicação da Ciência e Tecnologia em organismos vivos, bem como a produtos de forma a alterar matéria viva ou produtos derivados, com o objectivo de produzir conhecimento, bens e serviços". Esta definição cobre todas as áreas da Biotecnologia moderna e a OCDE recomenda que esta definição seja acompanhada por uma lista de técnicas biotecnológicas com o intuito de a complementar [2]. A definição, tal como a lista complementar, podem ser consultadas no site da OCDE (http://www.oecd.org).

Actualmente, a aplicação e o desenvolvimento de novos processos biotecnológicos tornou-se importante na indústria química, alimentar e farmacêutica, devido às preocupações ambientais e energéticas. Com o aumento do valor do petróleo e os incentivos para desenvolver novas tecnologias "verdes", com menor impacto ambiental, as empresas e as instituições académicas têm vindo a procurar substituir e criar novas tecnologias com recurso a processos biológicos. É expectável que em 2010, 20% dos produtos químicos a nível mundial sejam produzidos através de processos biológicos [3]. Ao utilizar novas técnicas biotecnológicas como o DNA recombinante, é possível desenvolver novos microrganismos com características especificas, de forma a aumentar os rendimentos de produção, bem como produzir um produto mais puro. Porém, o crescimento destes microrganismos e a respectiva produção são efectuados em reactores em condições controladas. A determinação das condições ambientais óptimas é de extrema importância para obter o máximo de produtividade. Deste modo, para optimizar um processo biológico, é necessário realizar 2 etapas, na maioria das vezes: uma primeira etapa aonde é realizado um melhoramento genético do organismo e uma segunda etapa na qual se tentam obter as condições óptimas do processo de produção, tais como as concentrações iniciais de nutrientes, perfis de alimentação, temperatura, pH. A última etapa é geralmente realizada a nível industrial através de experiências por tentativa-erro [4].

Nos anos recentes, os avanços verificados a nível da capacidade de processamento computacional permitiram o aparecimento de novas ferramentas informáticas para análise de dados bioquímicos, sendo possível compreender de

forma mais célere a estrutura e a dinâmica dos sistemas biológicos. Assim, foi possível estabelecer novas teorias e novas técnicas experimentais para resolver problemas na Biotecnologia e na Medicina [5]. Várias destas aplicações informáticas foram desenhadas e implementadas para modelação e simulação das vias metabólicas, incluindo por exemplo a análise de fluxos existentes no interior das células. Algumas destas ferramentas como Copasi [6], CellDesigner [7], Systems Biology Workbench [8], e Systems Biology Toolbox [9], apresentam funcionalidades semelhantes, apesar de todas elas diferirem em termos de usabilidade, aplicabilidade e na rapidez de executar as tarefas.

Embora haja uma grande variedade de ferramentas para realizar optimização, simulação e estimação de parâmetros em sistemas intracelulares, verifica-se uma falta de ferramentas com funcionalidades semelhantes para processos fermentativos. Faltam ferramentas desenhadas especificamente para a optimização de processos fermentativos em biorreactores, tais como: optimização das concentrações iniciais de nutrientes, de perfis de alimentação ou para a estimação de parâmetros a partir de dados experimentais obtidos de fermentações.

1.2. Objectivos

O principal objectivo desta tese foi o de desenvolver uma ferramenta computacional de fácil utilização para optimização e simulação de processos fermentativos. Esta ferramenta foi implementada em código aberto, de modo a poder ser usada e estendida por qualquer utilizador, com a mais-valia de ser funcional em qualquer sistema operativo. Foi desenvolvido também um módulo para a estimação de parâmetros fermentativos com recurso a dados experimentais obtidos de fermentações realizadas em sistema fechado ou em semi-continuo.

A ferramenta foi implementada na linguagem de programação Java, devido a ser uma linguagem multi-plataforma, simples, robusta e de elevada performance, além de ser também uma linguagem utilizada por um elevado número de programadores. A ferramenta foi desenvolvida usando uma plataforma base designada por *AIBench (http://www.aibench.org)*. Esta aplicação segue o paradigma MVC (*Model-View-Controller*), um padrão utilizado na Engenharia de Software no qual as várias tarefas de controlo, visualização e acesso aos dados estão logicamente separadas. Foram desenvolvidas várias classes de forma a prover a ferramenta com uma interface gráfica amigável e de fácil compreensão para o utilizador e para executar as tarefas de simulação, optimização e de estimação de parâmetros. Os algoritmos de optimização implementados nesta ferramenta foram desenvolvidos anteriormente no grupo de investigação onde o trabalho se integra e constam de: Algoritmos Evolucionários [10-12], Evolução Diferencial [11] e *Simulated Annealing* [13].

Após a conclusão da ferramenta foram realizados alguns estudos para verificar o desempenho, fiabilidade e repetibilidade da aplicação nos vários âmbitos descritos anteriormente.

1.3. Estrutura da tese

A tese está organizada em seis capítulos distintos. O presente capítulo inclui uma pequena descrição do estado da arte relativamente ao uso de ferramentas informáticas em Biotecnologia e uma breve explicação do motivo da realização deste trabalho.

Nos restantes capítulos faz-se a seguinte exposição:

Capitulo 2 – É apresentada uma breve descrição teórica dos processos fermentativos (os tipos de reactores e modos de operação). De seguida, faz-se uma exposição dos conceitos gerais utilizados na modelação e optimização de bioprocessos e a sua aplicabilidade em processos realizados em modo semi-contínuo ou em sistema fechado. Por fim, faz-se uma pequena descrição das ferramentas informáticas existentes para simulação e optimização de processos biológicos.

Capitulo 3 – São expostas pormenorizadamente todas as funcionalidades que foram implementadas na ferramenta informática *OptFerm* e os requisitos mínimos necessários para a utilização desta ferramenta.

Capitulo 4 – Descreve como foi implementada a ferramenta *OptFerm*, e como foram criados e incorporados (no caso de módulos externos) os diferentes módulos na aplicação. É ainda relatado como estes módulos se interligam.

Capitulo 5 – Descreve um caso de estudo com o intuito de mostrar visualmente as diversas funcionalidades e como se podem realizar as várias operações disponíveis na aplicação *OptFerm*. No final, é exposto um pequeno estudo que foi realizado com o objectivo de verificar a robustez da aplicação através de *30* repetições de uma mesma operação.

Capitulo 6 - São apresentadas as conclusões finais e faz-se uma pequena exposição das funcionalidades a acrescentar em trabalho futuro.

Capítulo 2

Introdução teórica

No presente capítulo, faz-se uma exposição teórica dos campos mais importantes relacionados com o trabalho desenvolvido nesta tese. O capítulo está organizado da seguinte forma:

2.1. Processos biotecnológicos

A OCDE considerou os processos biotecnológicos como um ramo da biotecnologia, integrando-os na lista de suporte à definição de biotecnologia [2]. São processos que utilizam organismos (bactérias, leveduras, fungos, etc.) ou enzimas para produção de um determinado produto.

2.1.1. Fermentação

O termo "Fermentação" é utilizado por microbiólogos e bioquímicos para descrever qualquer processo de produção que envolva cultura de microrganismos (crescimento). Actualmente, os processos fermentativos são largamente utilizados a nível industrial para produção de uma vasta gama de diferentes produtos, os quais se podem dividir em sete categorias como se pode observar na Tabela 1 [14,15]:

Tabela 1: As diferentes categorias de produtos produzidos através de processos fermentativos, Fonte [14].

Categoria do produto	Produto
Biomassa	Levedura de padeiro Bactérias lácteas Células microbianas (single cell protein)
Metabolitos primários	Vinho Etanol Acido láctico Acido cítrico Glutamato etc.…
Metabolitos secundários	Penicilinas Cefalosporinas Estatinas
Proteínas recombinantes	Insulina Hormonas de crescimento Vacinas Anticorpos
Enzimas	Enzimas para detergentes Enzimas para a indústria do amido
Polímeros	*Polihidroxialcanoatos (PHAs)* Goma Xantana
DNA	Vacinas Terapia genética

Os produtos são originados durante a fase de crescimento celular (fase exponencial) ou na fase estacionária após o crescimento celular. Durante a fase exponencial produzem-se os metabolitos primários, enzimas e proteínas, essenciais ao crescimento; na fase estacionária são produzidos os metabolitos secundários que aparentemente não têm um impacto directo no crescimento celular. Não obstante,

muitos destes metabolitos secundários apresentam funções antimicrobianas e de inibição enzimática.

2.1.2. Biorreactores

Todos os produtos apresentados anteriormente são produzidos, a nível industrial, em biorreactores ou fermentadores. Estes são contentores fechados, que permitem assegurar um ambiente controlado de forma a garantir as condições óptimas para o desenvolvimento dos microrganismos ou das reacções catalisadas por enzimas. As condições ambientais normalmente controladas são: a temperatura, o pH, a concentração de oxigénio e a composição do meio fermentativo (inicial ou alimentado ao reactor). Para este efeito, existem vários tipos de biorreactores, sendo os 3 tipos base os seguintes [16-18]:

- Tanque agitado (***S****tirred* ***T****ank* ***B****ioreactor*) – É o tipo de reactor mais utilizado na realização de fermentações. Este reactor pode ser utilizado em sistema fechado, semi-continuo ou continuo. O principal factor que o diferencia de todos os outros reactores é o seu sistema de agitação mecânico, provido de um veio central com várias pás, permitindo atingir um sistema próximo do perfeitamente agitado. A entrada de gases é efectuada no fundo do reactor, usualmente através de compressão. A combinação de agitação com o caudal de gás de entrada no reactor, permite aumentar tanto a velocidade de dissolução do gás como a concentração de gás em solução. Este reactor tem uma camisa de aquecimento de forma a manter a temperatura constante. Devido ao seu sistema de agitação, é o reactor que apresenta mais gastos energéticos por unidade de volume.
- Reactor com circulação por arejamento (*Airlift*) – Neste tipo de reactor, a agitação é promovida pela circulação do líquido no seu interior. O *airlift* é constituído por dois tubos, ligados no topo e no fundo. Num dos tubos (ascendente), o ar é injectado pelo fundo, e no outro tubo (descendente) a pressão de ar é praticamente inexistente. Esta configuração faz com que o líquido circule por diferença de densidades entre o tubo ascendente e descendente. Assim, a velocidade de agitação pode ser gerida pelo caudal de gás alimentado ao reactor. A transferência de oxigénio que se consegue com este tipo de reactor é geralmente menor do que a que se consegue obter num reactor *STB*.
- Reactor empacotado (*Packed-bed e fluidized bed bioreactor*) – É constituído por um tubo cheio com partículas de (bio)catalisador, no qual a alimentação pode ser realizada pela extremidade inferior ou superior. Em alguns casos, o meio tem que ser várias vezes recirculado pela coluna, para aumentar a conversão, requerendo um tanque de armazenamento.

2.1.3. Modos de operação

Existem três modos de operação de biorreactores: em sistema descontínuo, em semi-continuo e em contínuo.

No sistema descontínuo, o reactor é cheio inicialmente com o meio fermentativo (substratos, biocatalizadores) e não é adicionado qualquer tipo de substrato no decorrer da fermentação. Apenas se adiciona ácido ou base para manter o pH constante. A fermentação decorre até que um substrato seja totalmente consumido.

No sistema em semi-continuo, no início da fermentação o reactor é cheio parcialmente com o meio fermentativo, e no decorrer da fermentação vai-se adicionando o substrato limitante, de forma contínua ou parcial até que se atinja o volume máximo útil do reactor ou então até se atingir um ponto desejado para parar a fermentação (produtividade máxima) [16]. Normalmente, a alimentação tem início quando a fonte limitante de carbono decresce para um nível mínimo crítico. No final de cada ciclo fermentativo, pode-se deixar uma porção de meio e começar novamente um novo ciclo de fermentação [14]. A utilização de sistemas em semi-continuo é importante quando existem efeitos repressivos de fontes de carbono rapidamente utilizáveis ou então quando existe um efeito tóxico de um componente no meio.

Nos sistemas em contínuo, o volume de fermentação é mantido constante, havendo entradas e saídas no reactor. O estado estacionário é atingido quando a velocidade de crescimento de uma cultura se mantém constante, designando-se de quimiostato. Este sistema permite alcançar uma produtividade bastante maior e apresenta ainda a vantagem de os equipamentos serem de menor dimensão comparativamente aos utilizados nos sistemas em descontínuo ou em semi-continuo. Porém, tem a desvantagem de apresentar um risco elevado de contaminação [16]. Este é um dos principais factores que impede o seu uso generalizado, além de que, em muitos processos se pretendem obter metabolitos secundários, que na maioria dos casos, só são produzidos após a fase de crescimento exponencial. Tal como os sistemas em semi-continuo, a produtividade pode ser optimizada pelo caudal de alimentação.

2.2. Modelação

Segundo Swetz e Hartzler [19], um "modelo matemático de um objecto ou de um fenómeno real é um conjunto de regras ou leis, de natureza matemática, que representam adequadamente o objecto ou o fenómeno". Um modelo matemático de qualquer sistema biológico não é mais do que uma representação formal do nosso conhecimento acerca das interacções e leis presentes nesse sistema [20]. Deste modo, a modelação matemática tornou-se uma ferramenta essencial na engenharia de processos, tanto na indústria química como na Biotecnologia, pois ajuda a interpretar e a prever fenómenos naturais e experimentais. Além do mais, permite testar pressupostos através de simulações, de modo a verificar o comportamento dos sistemas, permitindo optimizar determinado processo. Pode ser estabelecido mais do

que um modelo para o mesmo fenómeno, dependendo dos objectivos do modelo e dos dados experimentais disponíveis para a caracterização desse fenómeno.

2.2.1. Modelação de processos biológicos

A modelação de processos biológicos ou reacções enzimáticas é bastante difícil, e na maioria dos casos é necessário transitar várias etapas durante o desenvolvimento de um modelo até obter um modelo minimamente credível [21]. Na Figura 1, estão descritos os diferentes passos a executar até que se obtenha um modelo válido. Os modelos biológicos são uma simplificação da realidade, pois devido à enorme complexidade do metabolismo celular e ao comportamento não linear das reacções enzimáticas, é impossível caracterizar totalmente um sistema biológico. A principal dificuldade em criar um modelo é identificar quais os factores que influenciam o crescimento do microrganismo e os que afectam a produção de determinado produto desejado. Para a maioria dos microrganismos, existe um total desconhecimento ou um conhecimento parcial de como funcionam os seus mecanismos metabólicos, o que aumenta a dificuldade de construir um modelo biológico.

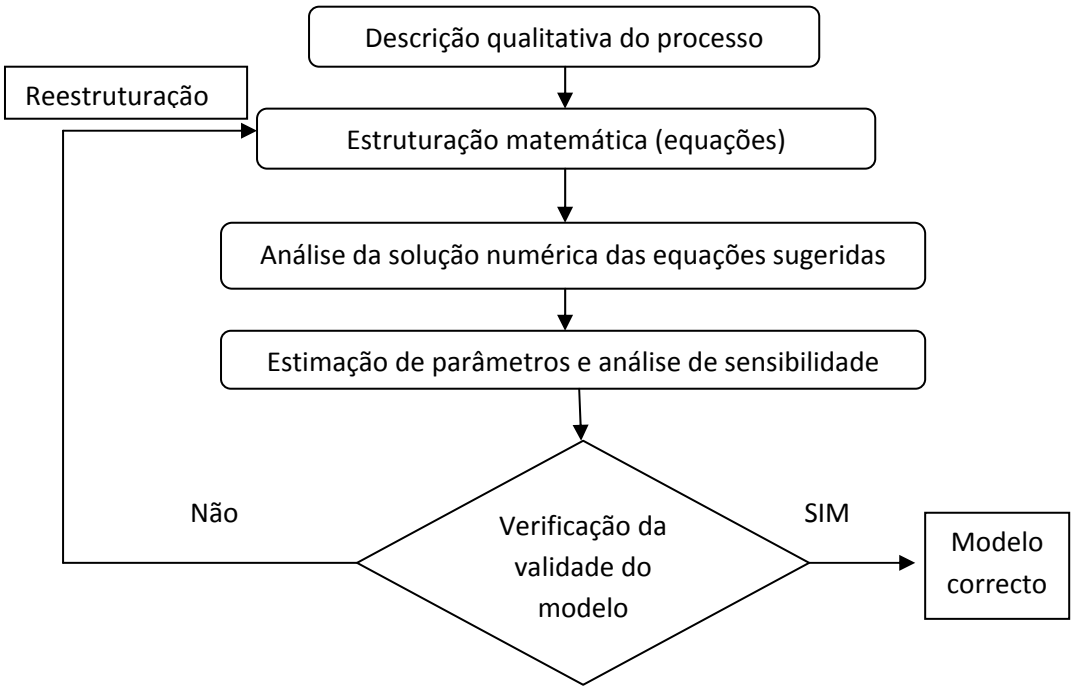


Figura 1 : Esquema exemplificando os diferentes passos realizados até obter um modelo que descreva correctamente um processo.

Os modelos podem ser classificados de acordo com 2 critérios distintos, modelos estruturados ou não estruturados ou modelos segregados ou não segregados.

- Modelo não estruturado – Neste tipo de modelos, considera-se a célula como uma única entidade, a qual é vista como um sistema de "caixa negra". Sabe-se

o que a célula consome e o que produz, mas não são considerados os processos envolvidos no seu interior. O modelo só descreve as cinéticas de crescimento celular, de consumo e de produção. Devido à simplicidade na concepção destes modelos, este podem ser utilizados para descrever o comportamento de fermentações, possibilitando estratégias de controlo, simulações e optimizações facilitando o *scale-up* dos processos fermentativos [22,23].

- Modelo estruturado – A célula é vista como uma entidade independente, e na formulação dos modelos tem-se em consideração as vias metabólicas, os metabolitos e as enzimas existentes no interior da célula. Também são descritos os mecanismos regulatórios e cinéticos presentes na célula, os mecanismos de transporte para o interior da célula e os mecanismos físico-químicos no exterior da célula [24]. Estes modelos oferecem, à partida, resultados mais próximos dos processos biológicos [25]. Contudo, devido à complexidade dos mecanismos envolvidos e ao grande número de reacções e vias metabólicas presentes, torna-se difícil estabelecer um modelo completo de modo a descrever o metabolismo celular como um todo [26]. Além disso, para muitos dos microrganismos, os mecanismos regulatórios não foram ainda esclarecidos.

- Modelo não segregado – A população celular é considerada como uma única variável no modelo (Biomassa por unidade de volume) e as células são encaradas como um conjunto e não como células independentes [21].

- Modelo segregado – os modelos são construídos com base numa heterogeneidade celular, em que as células podem estar em estado fisiológicos diferentes num mesmo instante [21].

A maioria dos modelos biológicos são construídos conjugando estas duas categorias. Assim, pode-se construir um modelo de acordo com uma das quatro combinações mostradas na Figura 2.

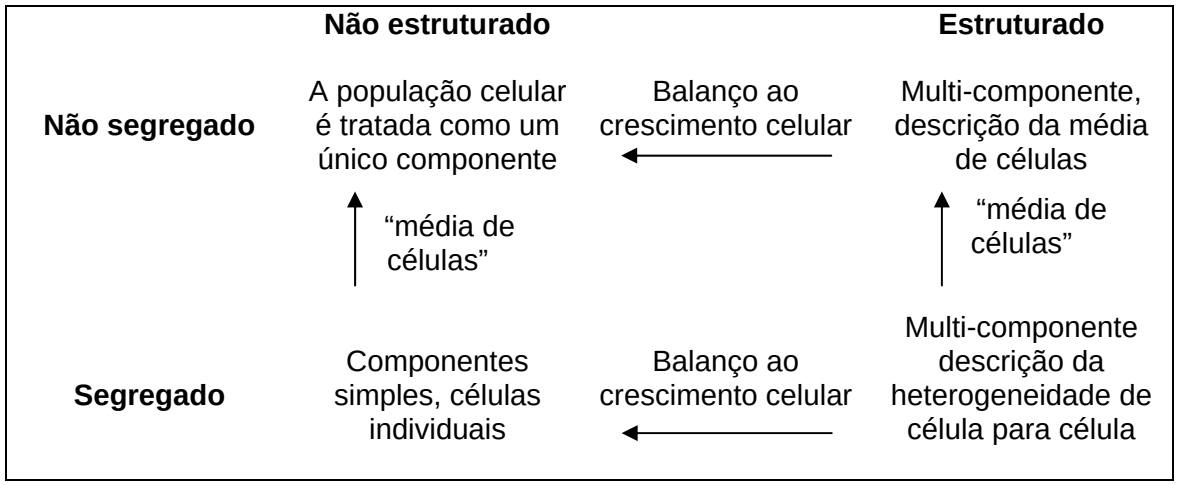


Figura 2: Classificação da representação matemática dos modelos biológicos de acordo com Bailey [27].

Os modelos representativos dos processos fermentativos são considerados modelos macroscópicos, e descrevem o biorreactor como um sistema. Este sistema pode estar dividido em dois submodelos, um modelo relacionado com os fenómenos de transporte, onde são consideradas as transferências de massa e de calor e um outro modelo relacionado com o crescimento celular, o qual incorpora todas as funções relacionadas com o crescimento celular e as respectivas cinéticas de consumo e produção [28].

2.2.2. Modelo dinâmico geral de reactores biológicos

Os reactores do tipo *tanque agitado (STB)* são os que se utilizam mais frequentemente na indústria biotecnológica. Usualmente, é utilizado um modelo dinâmico para caracterizar este tipo de reactor, no qual se faz o balanço de massas aos componentes envolvidos no esquema reaccional. A equação genérica que caracteriza estes balanços é:

$$Acumulação = conversão + (entrada - saida) \quad (1)$$

A incorporação dos factores de entrada e de saída depende do modo de operação da fermentação. Num sistema fechado, não existe nem entrada nem saída de componentes, em semi-continuo não existe saída de componentes e em contínuo existe entrada e saída de componentes. Em 1990, Bastin e Dochain [29] estabeleceram um Modelo Dinâmico Geral de Reactores Biológicos (reactores perfeitamente agitados), através de balanços de massa aos seus componentes, sendo este descrito pela seguinte equação:

$$\frac{d\xi_i}{dt} = \sum zk_{ij}\varphi_j(\xi_i, t) - D\xi_i + F - Q \quad (2)$$

$$i, j \in \mathbb{N}$$

onde:

ξ_i variável de estado (concentração do componente *i*);

k_{ij} coeficientes de rendimento ou coeficientes estequiométricos do componente *i* na reacção *j*;

φ_j taxa cinética da reacção *j*;

D taxa de diluição definida pela razão entre o caudal volumétrico de alimentação (F_{in}) e o volume de líquido dentro do reactor (V), $D = \frac{F_{in}}{V}$;

F caudal mássico da entrada do liquido no reactor;

Q caudal mássico de saída do componente (na forma gasosa) do reactor;

Z toma o valor igual a *-1* se o componente ξ_i é consumido ou 1 se for produzido;

Quando se utiliza o modelo dinâmico geral de reactores biológicos para representar um sistema em contínuo ou semi-contínuo, é necessário adicionar uma equação representando a variação de volume no interior do reactor, conforme apresentado a seguir:

$$\frac{dV}{dt} = F_{in} - F_{out} \quad (3)$$

onde,

F_{in} caudal volumétrico de entrada

F_{out} caudal volumétrico de saída do reactor. Para um sistema em semi-continuo o parâmetro F_{out} não é considerado.

2.2.3. Cinética das reacções

O maior desafio no desenvolvimento de um modelo é descrever correctamente as cinéticas que caracterizam os processos de conversão em função das concentrações dos componentes relevantes (biomassa, substratos e produtos). A determinação dos parâmetros cinéticos pode ser realizada através de dois métodos diferentes [30]:

- Medindo as velocidades iniciais das reacções para condições reaccionais diferentes.
- Através de experiências em sistema fechado.

A actividade enzimática que está na base das cinéticas de crescimento e de consumo de substrato e produção depende de inúmeros factores, tanto ambientais (e,g, temperatura e pH) como das concentrações de reagentes ou da concentração de substâncias inibidoras. Os fenómenos de transferência de massa também podem influenciar grandemente na actividade enzimática, pois influenciam directamente as concentrações dos reagentes [14]. A junção de todos estes fenómenos dificulta o estabelecimento de um mecanismo reaccional que descreva adequadamente a actividade enzimática observada para uma larga gama de temperaturas, valores de pH e concentração de inibidores. Por vezes, só é possível caracterizar a actividade enzimática para determinadas condições específicas de temperatura e pH.

Uma das equações cinéticas mais conhecida e frequentemente utilizada na caracterização da cinética do crescimento microbiano é a cinética de Monod, baseada na equação de Michaelis-Menten [31]:

$$\mu(S) = \frac{\mu_{max} . S}{K_S + S} \quad (4)$$

onde:

μ taxa específica de crescimento,

μ_{max} taxa específica máxima de crescimento,

S concentração de substrato,

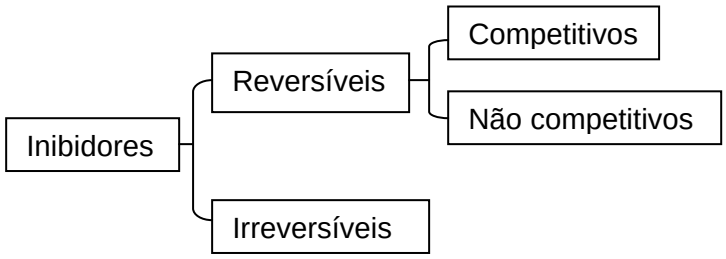
K_s parâmetro de saturação correspondente à concentração de substrato necessária para obter metade da taxa específica máxima de crescimento.

Existem várias outras equações relacionadas com o crescimento microbiano e com as cinéticas de consumo de substrato, sendo algumas referidas na Tabela 2 juntamente com os seus autores.

Tabela 2: Compilação de diferentes equações cinéticas

Autor	Expressão
Teissier [32]	$\mu = \mu_{max}\left(1 - exp\left(\frac{-S}{K_S}\right)\right)$
Moser [32]	$\mu = \mu_{max}\left(1 + K_S . S^{-\lambda_i}\right)^{-1}$
Contois e Fujimoto [32]	$\mu = \mu_{max}\frac{S}{K_S . X + S}$

A actividade de uma enzima pode ser afectada por substâncias denominadas de "inibidores". Quando estão presentes, estas podem reduzir significativamente a velocidade de uma reacção enzimática. Estas substâncias podem ser classificadas em:

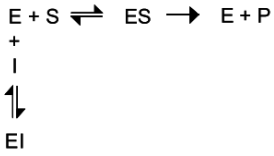
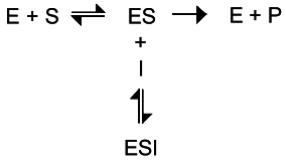
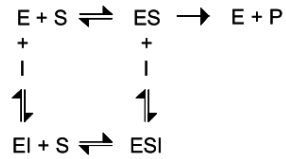


- **Inibição reversível** – A actividade enzimática é recuperada logo após a remoção do inibidor.

- **Inibição irreversível** – Um inibidor combina irreversivelmente com uma enzima; a velocidade enzimática diminui à medida que as enzimas vão realizando as ligações com o inibidor.
- **Inibição competitiva** – O substrato e a substância inibidora competem pelo mesmo sítio activo da enzima. O inibidor pode ser uma substância análoga, um derivado do substrato ou um produto da reacção. Logo que o inibidor se liga à enzima, existe uma modificação na conformação, impedindo a ligação do substrato.
- **Inibição não competitiva** – O inibidor não tem efeito directo na ligação do substrato à enzima. O inibidor e o substrato ligam-se reversivelmente, e o complexo enzima-subtrato-inibidor é cataliticamente inactivo, pois o inibidor impede o posicionamento correcto do substrato no centro activo.

Na Tabela 3, são apresentados os diferentes tipos de inibição, os respectivos esquemas referentes aos complexos enzima-subtrato e enzima-inibidor e as equações cinéticas que descrevem a actividade enzimática na presença de uma substância inibidora.

Tabela 3: Compilação dos vários tipos de inibição cinética e respectivas equações e esquemas reaccionais (fonte [30]).

Tipo de inibição	Esquema	Equação
Competitiva		$v = V_{max}\frac{[S]}{K_M\left(1+\frac{[I]}{K_I}\right)+[S]}$
Produto	Semelhante à anterior mas I=P	$v = V_{max}\frac{[S]}{K_M\left(1+\frac{[P]}{K_I}\right)+[S]}$
Não competitiva		$v = V_{max}\frac{[S]}{K_M+\left(1+\frac{[I]}{K_I}\right)\cdot[S]}$
Substrato	Semelhante à anterior mas I=S	$v = V_{max}\frac{[S]}{K_M+[S]+\frac{[S]^2}{K_I}}$
Mista		$v = V_{max}\frac{[S]}{\left(1+\frac{[I]}{K_I}\right)(K_M+[S])}$

2.3. Optimização de Bioprocessos

A utilização de técnicas de optimização teve tal impacto em diferentes áreas da indústria, que aquelas se tornaram uma peça fundamental na engenharia de processos [33]. Desde que G.B Dantzig apresentou em 1947 o algoritmo Simplex [34], para programação linear, até aos dias de hoje, foram desenvolvidas inúmeras técnicas e algoritmos de optimização sempre com o objectivo de optimizar problemas de difícil resolução (modelos não lineares) e de larga escala. Os processos de optimização são baseados em três princípios, a codificação do problema, a função objectivo que se pretende minimizar ou maximizar e o espaço de soluções associado. Basicamente, trata-se de encontrar os valores das variáveis do processo que produzem uma melhor rentabilidade ou o valor de menor custo, dependendo do critério da optimização [35]. A forma geral dos problemas de optimização é a seguinte [36]:

$$\min_{x \in R^n,} f(x) \tag{5}$$

em que x é uma variável de decisão, $f(x)$ é a função objectivo. Se o problema de optimização apresentar restrições, a forma geral é a seguinte [36]:

$$\begin{aligned} &\min_{x \in R^n,} f(x) \\ &c_i(x) = 0,\ i \in E \\ &c_i(x) \geq 0,\ i \in I \end{aligned} \tag{6}$$

onde *E* e *I* são o conjunto de índices de condições de igualdade e condições de desigualdade, $c_i(x), (i = 1, \ldots, m \in E \cup I)$ são as funções de restrições. Quando as funções objectivo e as funções das restrições são lineares, é considerado um problema de programação linear; de outra forma é considerado um problema de programação não linear.

A aplicação de métodos de optimização em processos biotecnológicos veio possibilitar tanto o aumento das produtividades (em biomassa e/ou produtos) como o melhoramento de processos (favorecimento de determinadas vias metabólicas num microrganismo de modo a aumentar a produção de determinado produto de maior valor acrescentado).

Outro campo que obteve grande atenção na área dos bioprocessos foi a aplicação de optimização dinâmica a fermentações em modo semi-continuo. Ao aplicar este tipo de optimização, foi possível determinar os melhores perfis de alimentação de modo a maximizar a produtividade ou um índice económico derivado das concentrações finais [37]. A maioria dos bioprocessos apresentam comportamentos

não lineares, e frequentemente existe a necessidade de definir restrições às variáveis de estado e de controlo. Deste modo, é importante utilizar técnicas de optimização dinâmica robustas. Além do mais, na maioria das vezes, estas têm de operar sob modelos complexos e lidar com a presença de ruídos (erros) nos dados experimentais [38]. Vários métodos de optimização têm vindo a ser utilizados tais como:

- Métodos Indirectos – baseados na transformação do problema de controlo original num problema com dois pontos fronteira, utilizando as condições necessárias de Pontryagin [37].
- Métodos Determinísticos - baseados na transformação do problema de controlo original em problemas de programação não linear utilizando vectores de parametrização [37].
- Métodos estocásticos – são considerados métodos adaptativos bastante robustos para problemas difíceis; porém, requerem grande capacidade computacional. Alguns destes métodos são: os Algoritmos Genéticos/ Evolucionários [10,39,40,38], Evolução Diferencial [10,41], *Simulated Annealing* [42] e método dinâmico de *Hill Climbing* [40].

Os algoritmos de optimização mais rápidos procuram apenas uma solução local, um ponto em que a função objectivo apresenta o seu valor mínimo. Porém, nem sempre é encontrado o melhor ponto mínimo designado de solução óptima global, pois habitualmente este ponto é difícil de identificar e localizar, dada a complexidade inerente a muitos problemas reais de optimização. A maioria dos algoritmos que conseguem realizar a optimização global fazem-no através da resolução sequencial de problemas de optimização local. Todos os bons algoritmos de optimização devem apresentar as seguintes propriedades:

- Robustez – Devem resolver uma grande variedade de problemas, independentemente da escolha dos valores iniciais das variáveis.
- Eficiência – Não devem requerer um tempo de computação demasiadamente elevado.
- Certeza – Devem ter a capacidade de identificar a solução com precisão, e devem ser insensíveis a erros nos dados ou a erros devido a arredondamentos aritméticos.

Dos vários algoritmos de optimização apresentados, os algoritmos estocásticos têm merecido uma maior atenção devido a apresentarem uma maior robustez face a problemas mais complexos e uma melhor eficiência quando se utilizam modelos com um grande número de variáveis. Actualmente, os algoritmos de base evolucionária baseados em mecanismos de evolução biológica são um dos algoritmos mais utilizados, sendo os mais conhecidos os Algoritmos Genéticos/ Evolucionários e os algoritmos de Evolução Diferencial.

2.3.1. Algoritmos Evolucionários

Os Algoritmos Genéticos foram desenvolvidos inicialmente por John Holland em 1975 [43]. Desde então, têm sido desenvolvidas várias variações destes algoritmos, apresentando no entanto sempre os mesmos conceitos, tendo a designação evoluído para o nome mais genérico de Algoritmos Evolucionários. Os AEs são inspirados na teoria de evolução de Darwin, sendo os problemas resolvidos através de um processo em que uma população de soluções evolui ao longo de várias gerações, sendo que as melhores soluções "sobrevivem" com uma maior probabilidade e podem originar novas soluções através da aplicação de operadores de reprodução.

As soluções potenciais para um problema específico são codificadas numa estrutura semelhante a um genoma, tipicamente de forma linear, i.e. como um vector de elementos (genes). Diversas alternativas são possíveis, desde a codificação binária (onde cada um dos elementos do genoma pode ter dois valores possíveis, 0 e 1), à codificação real usada neste trabalho (onde cada elemento é um valor numérico real), passando por outras alternativas (valores inteiros, conjuntos, permutações, etc.).

Sobre esta estrutura, e com o fim de criar novas soluções, são aplicados operadores de reprodução: cruzamento (*crossover*) e mutação de modo a gerar novas entidades (indivíduos ou soluções) num determinado espaço de busca. Em cada geração, é avaliada a adaptação de cada indivíduo na população e os mais aptos têm maiores probabilidades de serem escolhidos como progenitores e permanecerem na nova população [44,40,45], como demonstrado na Figura 3. Os diferentes passos de uma iteração de AEs são descritos em mais detalhe a seguir:

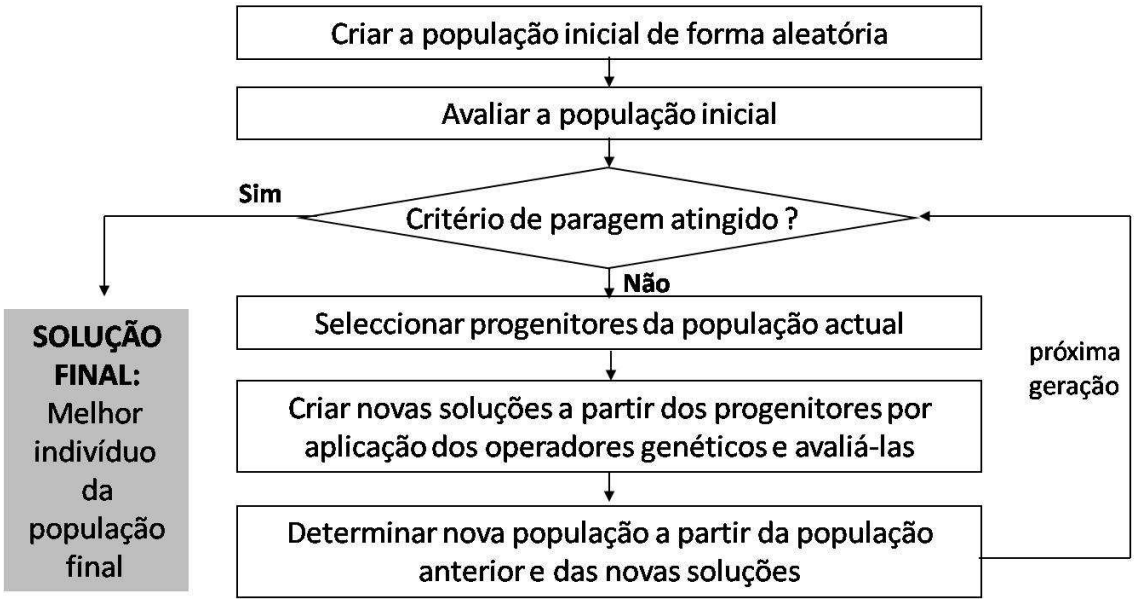


Figura 3: Estrutura geral de um Algoritmo Evolucionário

- **Inicialização** - No início, uma população de *n* indivíduos é gerada aleatoriamente, na qual cada indivíduo representa uma possível solução para o problema. O tamanho da população é importante, pois pode afectar a qualidade da solução final bem como o tempo de processamento. Numa população pequena, o processamento é rápido mas as soluções geradas podem não ser as melhores, enquanto numa população grande a probabilidade de produzir soluções melhores é maior, mas o tempo de processamento é maior.

- **Cálculo de aptidão** – é efectuado com base na função objectivo, sendo realizada uma análise de desempenho a cada indivíduo (cada indivíduo recebe um valor numérico, proporcional à sua qualidade).

- **Selecção** – Os indivíduos mais aptos em cada geração são utilizados para gerar uma nova população através de cruzamento e mutação. A probabilidade de cada indivíduo ser seleccionado é proporcional à sua aptidão. Existem dois métodos de selecção mais usados:

    - o método de selecção por roleta - cada secção da roleta é proporcional ao valor de adaptação de cada indivíduo; quanto maior o valor, maior o tamanho da secção. Contudo, quando as diferenças de aptidão são ínfimas este método degrada-se geração após geração.
    - o método de selecção por classificação - o indivíduo é seleccionado de acordo com o seu posicionamento dentro da população através da seguinte fórmula [46]:

$$p(i) = \frac{(n+1) - ordem(i)}{\sum_{j=1}^{n} j} \tag{7}$$

    onde *n* representa o tamanho da população e *ordem(i)* é a função de ordem que dá o posicionamento de cada indivíduo na população.

Cruzamento – dois indivíduos (progenitores) seleccionados na etapa anterior são cruzados originando novos indivíduos (descendência). Em representações lineares existem três tipos de cruzamento mais usados: cruzamento de um e dois pontos e cruzamento uniforme. No cruzamento de um ponto, os indivíduos a serem cruzados são divididos aleatoriamente num ponto, e existe uma permuta da metade inicial de um individuo com a metade final do outro individuo e vice-versa. No cruzamento de dois pontos as operações são semelhantes, mas existem dois pontos de corte. De seguida faz-se o emparelhamento das diferentes partes de um indivíduo com as do outro indivíduo de modo aleatório. No cruzamento uniforme é gerada aleatoriamente uma "máscara de cruzamento" que vai servir como mecanismo de decisão, para designar qual dos progenitores vai fornecer o gene ao descendente. O mecanismo funciona da seguinte forma:

1. Como exemplo é gerado uma máscara de cruzamento aleatório 0100111.
2. Seguindo o encadeamento binário da máscara, caso o valor seja igual a 1, o gene do progenitor 1 contido nessa posição é copiado para o descendente, se for 0 é copiado o gene do progenitor 2 dessa posição.

Mutação – As operações de mutação consistem em alterações localizadas de genes no indivíduo, sendo realizadas para evitar que o algoritmo convirja prematuramente para mínimos locais. A frequência com que este operador é aplicado é chamada a taxa de mutação, que normalmente é baixa para não tornar o processo demasiadamente aleatório [46].

2.3.2. Algoritmos de evolução diferencial

O algoritmo de Evolução Diferencial é um descendente do Algoritmo Genético, mas foi desenhado para realizar as optimizações mais rapidamente, em casos onde a representação das soluções pode ser realizada com base em variáveis numéricas (números reais). Foi formulado por Kenneth Price e Rainer Storn para resolver um problema de aproximação polinomial de Chebychev, em que os autores tiveram a ideia de utilizar um vector de diferenças para perturbar a população [41]. Este algoritmo também utiliza alguns operadores semelhantes aos anteriores mas com algumas distinções.

A grande diferença está na inicialização do algoritmo e na forma como são realizadas as diferentes etapas durante as operações de optimização. Inicialmente, são gerados aleatoriamente *p* vectores de dimensão *n*, que vão formar a população. A população vai sendo manipulada até ser encontrado um critério de paragem, que pode ser um determinado número de avaliações. O algoritmo foi implementado da seguinte forma [47]:

1. Inicializar a população;
2. Avaliar a população;
3. Gerar uma nova população em que cada indivíduo *i* é gerado em paralelo de acordo com os seguintes critérios:
   i. Seleccionar aleatoriamente 3 indivíduos distintos r_1, r_2, r_3 da população diferentes de *i;*
   ii. Gerar um vector experimental baseado no esquema utilizado (explicado abaixo);
   iii. Aplicar o cruzamento entre o vector experimental e o vector do indivíduo corrente.
   iv. Se o candidato não for válido, alterar as suas coordenadas inválidas, redefinindo-as para o limite mais próximo;
   v. Avaliação do candidato;
   vi. Utilizar o candidato na próxima geração só se for melhor que o indivíduo original;

vii. Alterar o indivíduo original pelo candidato se este for mais apto.

4. Voltar ao passo 3 se o critério de terminação não tiver sido ainda atingido.

As estratégias aplicadas na evolução diferencial podem variar de acordo com o tipo de indivíduo a ser modificado (o indivíduo corrente ou o indivíduo melhor adaptado) e com o número de indivíduos utilizados para a perturbação. A conjugação destes factores podem formar um esquema do tipo DE/a/b/ em que pode ser alterado da seguinte forma:

a – especifica o vector a ser perturbado, podendo ser "rand" caso se queira utilizar um indivíduo corrente escolhido aleatoriamente ou "best" para se utilizar o indivíduo melhor adaptado.

b - especifica o número de diferenças ponderadas usadas para a perturbação de a.

Alguns dos esquemas utilizados são os seguintes [48]:

DE/rand/1 $$\vec{t} = \vec{x}_{r_1} + F\left(\vec{x}_{r_2} - \vec{x}_{r_3}\right) \quad (8)$$

DE/best/1 $$\vec{t} = \vec{x}_{best} + F\left(\vec{x}_{r_2} - \vec{x}_{r_3}\right) \quad (9)$$

DE/best/2 $$\vec{t} = \vec{x}_{best} + F\left(\vec{x}_{r_1} + \vec{x}_{r_2} - \vec{x}_{r_3} + \vec{x}_{r_4}\right) \quad (10)$$

onde x_{r_j}, 2 ≤ j ≤ 3 representa os diferentes indivíduos seleccionados aleatoriamente da população, sendo estes diferentes do individuo x_{r_i}; x_{best} é o indivíduo mais apto e F representa o parâmetro de escala (valores tipicamente entre 0 e 2).

2.4. Estimação de parâmetros

Os modelos biológicos podem conter um grande número de parâmetros, dependendo da extensão e da complexidade desses mesmos modelos. Normalmente, os valores desses parâmetros são obtidos através de experiências ou recorrendo a valores descritos na bibliografia com origem em estudos de outros cientistas. Quando os valores dos parâmetros são desconhecidos devido a dificuldades em obter esse valor através de experiências ou à inexistência de informação na bibliografia, pode ser realizada uma estimação dos parâmetros através da aproximação dos dados simulados aos dados experimentais.

A estimação consiste na procura dos valores dos parâmetros θ descritos no modelo, que minimizem a distância entre os valores experimentais y e os valores simulados $f(x,\theta)$. O método dos mínimos quadrados é o método de regressão mais utilizado na estimação de parâmetros, em que as somas do quadrado das distâncias são minimizadas utilizando a seguinte equação [49]:

$$\min_{\theta} \sum_{n=1}^{N} \left(y_n - f(x, \theta) \right)^2 \quad (11)$$

Supondo que se tinha uma série de medições y_n recolhidas para diferentes valores x_n, o objectivo é determinar o conjunto de valores θ que originem o mínimo da função. Normalmente, são utilizados os métodos de Newton para resolver os problemas de mínimos quadrados de sistemas não lineares. Porém, na maioria das vezes, trata-se de um processo moroso e as equações tendem a ser mal condicionadas [36].

No entanto, podem ser utilizados métodos alternativos na estimação de parâmetros. Assim, algoritmos meta-heuristicos e heurísticos como os algoritmos evolucionários [10,12], evolução diferencial [10] e simulated anneling [13] são cada vez mais utilizados na realização destas tarefas devido à sua robustez quando se trabalha com um grande número de parâmetros ou quando se trabalha com modelos complexos. O conjunto de parâmetros do modelo é codificado num cromossoma, sobre o qual vão sendo aplicados os operadores de reprodução (selecção, cruzamento, mutação). A estimação processa-se de forma semelhante ao processo de optimização exemplificado na Figura 3, com uma diferença que o cálculo de aptidão é realizado com base na equação 11, que avalia a distância dos valores simulados com os valores experimentais. Os cromossomas que originarem a minimização dessa função serão seleccionados para produzir as futuras novas gerações.

2.5. Ferramentas de simulação e optimização

Nos últimos anos, várias ferramentas informáticas foram desenvolvidas para modelação e simulação de vias metabólicas e transdução de sinais presentes nas células. Nestas ferramentas, as reacções celulares são descritas por equações diferenciais ou por equações algébricas simples. As simulações são realizadas através de integração numérica, presumindo que todos os parâmetros têm valores predefinidos [50]. Algumas das ferramentas mais conhecidas na modelação e simulação de redes bioquímicas são o Copasi [6], o CellDesigner [7], o Systems Biology Workbench [51] (uma plataforma que engloba vários módulos entre os quais o JDesigner e o Jarnac) e o Systems Biology toolbox [9].

As Tabelas 4 e 5 apresentam um estudo comparativo em termos de funcionalidades, aplicabilidades, simplicidade de uso e compatibilidade, entre algumas das ferramentas mais utilizadas actualmente. Na Tabela 4, são apresentadas as características das aplicações e as plataformas onde se podem executá-las. É referida outra característica que é a capacidade de utilizar ficheiros SBML (*Systems Biology Markup Language*), que é um formato baseado em XML (*extensible Markup Language*) para representar modelos biológicos. A vantagem de utilizar este formato é que se pode utilizar um mesmo ficheiro (contendo um modelo) em qualquer ferramenta que trabalhe com este formato.

Tabela 4: Características técnicas e requisitos das várias ferramentas de software disponíveis (Fonte [6-9,52,53,51,54]).

Tipo da aplicação	Programa	Sistema operativo	Requisitos	Portabilidade dos modelos		Ferramenta de Biologia de sistemas	Código livre
				Importação	Exportação		
Simuladores	CellDesigner	Windows, Linux,OS X		SBML	SBML	•	
	CellWare	Windows, Linux,OS X	JRE ≥ 1.4	SBML	SBML		
	Copasi	Windows, Linux,OS X		SBML, Gepasi	SBML		
	Dizzy	Windows, Linux,OS X	JRE ≥ 1.4	SBML	SBML	•	•
	Dynetica	Windows, Linux,OS X	JRE ≥ 1.3				
	Gepasi	Windows		SBML	SBML		
	Pasadena Twain	Windows					•
	Plas	Windows					
	Systems Biology Toolbox	Windows, Linux	Matlab	SBML	SBML	•	•
Plataformas de Modelação	Systems Biology Workbench (JDesigner/Jarnac)	Windows					
	JSim	Windows	JRE ≥ 1.3				
Simuladores alojados em servidores de internet	Basis	NA					
	Virtual Cell	Java-enabled platforms	JRE ≥ 1.4 Web browser	SBML, CellML, Matlab	SBML, CellML, Matlab		

NA – Não aplicável, pois o acesso ao servidor apenas requer um Web browser.

Na Tabela 5 estão descritas as diferentes funcionalidades e aplicabilidade das aplicações, sendo estas assinaladas com uma bola preta. A maioria dos programas só suporta trabalhar com equações diferenciais ordinárias, e apenas cinco é que permitem realizar a estimação de parâmetros. Apesar de existir uma grande variedade de aplicações, todas elas foram desenhadas para estudar especificamente vias metabólicas, estando direccionadas para o estudo dos sistemas celulares.

Tabela 5: Análise às capacidades das aplicações (Fonte [6-9,52,53,51,54])

Tipo da aplicação	Programa	Equações diferenciais ordinárias	Equações diferenciais parciais	Estocástico	Compartimentação automática	Cálculo directo do estado estacionário	Análise de sensibilidade em estado estacionário	Análise de sensibilidade ao longo do tempo	Averiguação automática dos parâmetros	Perturbação ao longo do tempo	Estimação de parâmetros / Optimização
Simuladores	CellDesigner	•			•	•				•	
	CellWare	•		•	•						•
	Copasi	•		•	•	•	•	•		•	•
	Dizzy	•		•	•					•	
	Dynetica	•		•							
	Gepasi	•			•	•	•a	•			•
	Pasadena Twain	•									
	Plas	•				•	•b	•b		•	
	Systems Biology Toolbox 2	•		•		•	•	•	•	•	•
Plataformas de Modelação	JDesigner/Jarnac			•	•	•	•	•	•c	•	
	JSim	•	•				•c		•	•	•
Simuladores alojados em servidores de internet	Basis			•	d					•	
	Virtual Cell	•	•		•				•		

a – Implementação incorrecta;
b – Para sistemas representados pelo formalismo das *power laws*;
c – Requer programação por parte do utilizador;
d – Permite apenas compartimentos unitários.

Capítulo 3

Requisitos e Funcionalidades da Ferramenta *OptFerm*

No presente capítulo, apresentam-se as funcionalidades existentes na aplicação *OptFerm*, bem como os requisitos necessários para que seja possível utilizar esta aplicação. Este capítulo está estruturado do seguinte modo:

Ao desenvolver esta ferramenta teve-se como principal objectivo proporcionar uma aplicação que possibilitasse testar diferentes condições nas operações descritas em cima, de forma simples e célere.

3.1. Requisitos Mínimos e disponibilidade

O OptFerm foi desenvolvido como software livre, estando disponível no sitio http://darwin.di.uminho.pt/optferm/. Esta ferramenta foi desenvolvida em linguagem JAVA e pode ser executada em qualquer sistema operativo necessitando apenas do *Java Runtime Environment* (JRE) versão 6 ou superior. O *JRE* pode ser adquirido gratuitamente no site **java.sun.com**. Os requisitos mínimos aconselhados em termos de *hardware* são: processador superior a 2 Ghz, 1 gigabyte de RAM.

3.2. Modelos

Os modelos são o elemento base de todas as operações presentes no *OptFerm*, visto que todas as operações são executadas sobre eles. Os modelos são caracterizados por equações diferenciais, representando os balanços de massa das variáveis de estado em função do tempo. Poderão ser utilizados quaisquer tipos de modelos, estruturados ou não estruturados, desde que sejam representados através das respectivas equações diferenciais.

Cada modelo é caracterizado por duas classes Java, uma classe *Processo* e uma outra classe *Função*, sendo a estruturação de cada uma das classes sempre igual para qualquer modelo. Após a definição das classes supracitadas, estas são compiladas e os valores inicialmente estabelecidos para as variáveis de estado e os parâmetros não podem ser modificados (ficando como valores padrão). Porém, podem ser criados novos grupos de valores iniciais das variáveis, parâmetros e os respectivos limites (inferior e superior) numa fase posterior.

3.2.1. Classe Processo

Nesta classe, são descritos os seguintes parâmetros:

- Nomes das variáveis de estado;
- Valores iniciais das variáveis de estado;
- Limites inferiores e superiores das variáveis de estado;
- Função objectivo referente à optimização.

Estes parâmetros, com excepção da função objectivo, são agrupados de forma a criar dois tipos diferentes de objectos, após a compilação do código Java:

- Nomes e valores iniciais das variáveis de estado, para criar um objecto “variáveis de estado”.
- Limites inferiores e superiores das variáveis de estado, com o intuito de criar um objecto “limites das variáveis de estado”

Numa fase posterior, estes conjuntos de dados (objectos) podem ser substituídos por outros conjuntos de dados contendo valores diferentes.

A função objectivo é a função principal no processo de optimização, pois todos os cálculos efectuados para determinação do respectivo perfil de alimentação têm como base a minimização ou maximização do valor desta função. Assim, o utilizador tem que ser cuidadoso ao definir esta equação, devendo verificar sempre qual ou quais as variáveis de estado a maximizar ou minimizar. A função tem que ser definida obrigatoriamente num método denominado *productivity*. Todavia, pode ser definido qualquer tipo de equação de acordo com o caso em estudo.

3.2.2. Classe Função

Nesta classe são definidos os seguintes parâmetros e equações:

- Nome dos parâmetros do modelo;
- Valores iniciais dos parâmetros;
- Limites inferiores e superiores dos parâmetros;
- Equações diferenciais;
- Equações cinéticas .

De forma similar ao apresentado na classe *Processo*, os nomes e valores dos parâmetros são agrupados num só objecto, tal como os limites inferiores e superiores, permitindo assim a criação e utilização de novos objectos com diferentes valores numa fase posterior. As equações diferenciais e cinéticas são definidas em métodos diferentes.

Equações diferenciais

A dinâmica do processo é caracterizado pelas equações diferenciais, seguindo o paradigma descrito por *Bastin e Dochain* [29] conforme apresentado na equação 2. Contudo, o vector de caudais mássicos referente às saídas gasosas poderá ser desprezado, dependendo do modelo aplicado e da descrição respiratória que se fizer do microrganismo. Por exemplo, se se verificar com recurso a dados experimentais que a perda de massa devido à saída de gases do biorreactor é mínima, então este dado pode ser desprezado.

Pode ser definido qualquer tipo de modelo, desde que se siga o paradigma descrito na equação 2. Em todos os modelos é obrigatório definir tanto o parâmetro referente ao caudal de alimentação de um substrato (F_{in}). Além disso, o factor de diluição deve estar incorporado em todas as equações diferenciais referentes a cada variável de estado.

Equações cinéticas

As equações cinéticas são definidas na classe *Função* num método separado das equações diferenciais. Os valores das equações cinéticas são calculados sempre que requerido pelo método das equações diferenciais e são passados a este através de um vector. Podem-se utilizar quaisquer tipos de equações cinéticas, nas quais se podem aplicar restrições, de modo a impor ou condicionar uma determinada reacção quando determinado valor de uma variável de estado ou reacção cinética for atingido. Estas restrições são descritas através de estruturas *if-then-else* ou utilizando estruturas *while* ou *do-while*.

A construção *if* tem a forma:

```
if (expressão controle 1) {
declaração 1
 }
else if (expressão controle 2) {
declaração 2
 }
else {
declaração 3
}
end
```

enquanto a construção *while* e *do-while* tem a forma:

```
while (expressão controle) {
      declaração
}
end
```

ou

```
do {
   declaração
}
while (expressão controle)
```

Esta metodologia requer alguns conhecimentos por parte do utilizador de como funcionam as estruturas de controlo em Java. Todavia este procedimento permite uma enorme flexibilidade na definição de um modelo reaccional.

3.3. Simulação

Esta operação permite a visualização do comportamento das variáveis de estado ao longo do tempo. Os utilizadores têm a possibilidade de realizar diversas simulações de um processo com diferentes combinações de valores iniciais das variáveis de estado, parâmetros e perfis de alimentação. É também possível utilizar conjuntos de valores obtidos nas optimizações do perfil de alimentação, bem como valores de parâmetros resultantes da estimação destes conforme ilustrado na Figura 4.

À medida que estas operações vão sendo realizadas, os resultados ficam logo disponíveis para empregar nas simulações. Contudo, só se pode utilizar um perfil de alimentação em cada simulação, seja este proveniente de uma optimização ou definido pelo utilizador. O mesmo acontece na utilização dos parâmetros: ou se utiliza um conjunto pré-definido pelo utilizador ou um conjunto proveniente da estimação de parâmetros. Deste modo, podem-se realizar diversas simulações com diferentes condições, acelerando assim o processo de estudo para verificação das condições mais adequadas para adoptar num processo fermentativo.

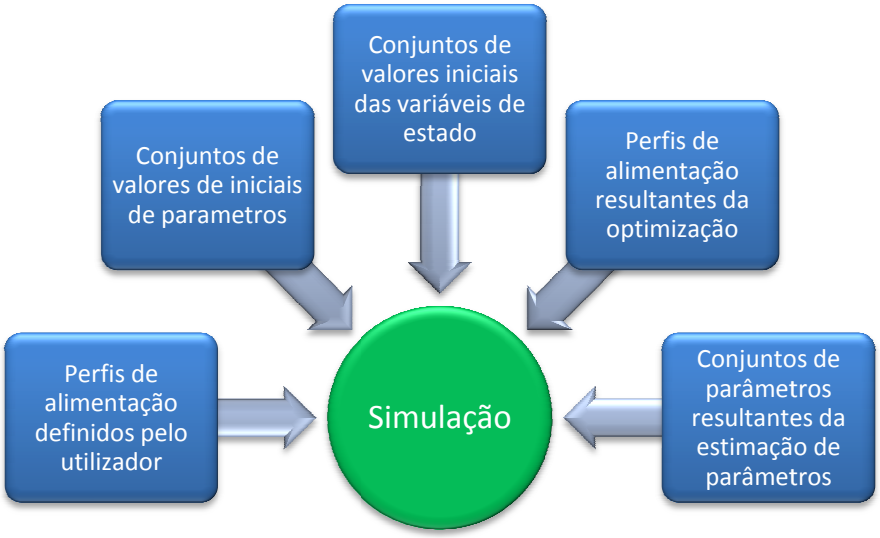


Figura 4: Tipos de dados que poderão ser utilizados numa simulação

Os resultados das simulações são apresentados ao utilizador através de gráficos, onde se pode visualizar a variação das variáveis de estado ou das cinéticas ao longo do tempo. É facultado ao utilizador a possibilidade de visualizar cada uma das variáveis separadamente ou então de todas em conjunto.

Os dados das simulações podem ser comparados com dados experimentais, de modo a verificar a sua aproximação, permitindo consequentemente a validação dos valores utilizados nas simulações. É possível comparar uma simulação com diferentes conjuntos de dados experimentais ou vice-versa. Após a execução de uma simulação, o utilizador tem ainda ao seu dispor uma funcionalidade que permite somar duas ou mais variáveis de estado, resultando num único gráfico referente à soma dessas variáveis ao longo do tempo. Esta funcionalidade pode ser útil quando se pretende

optimizar simultaneamente mais do que uma variável de estado, sendo assim possível verificar o total das partes obtidas.

A simulação é realizada através da integração numérica do modelo utilizado. A integração numérica pode ser realizada através de dois algoritmos diferentes, ambos baseados em esquemas de Runge-Kutta. Um utiliza um método Implícito-Explícito de Runge-Kutta de passo linear múltiplo (*IMEX*), e um outro o método de Runge-Kutta de passo linear constante (*ERK*), ambos os algoritmos estão incluídos no módulo OdeToJava [55] .

3.4. Optimização

A ferramenta *OptFerm* permite realizar três tipos de optimização distintos:

- Perfil de alimentação de uma corrente de substrato a alimentar ao reactor;
- Perfil de alimentação de uma corrente de substrato simultaneamente com a determinação dos melhores valores das variáveis de estado para iniciar uma fermentação.
- Perfil de alimentação de uma corrente de substrato simultaneamente com a determinação do tempo óptimo para finalizar uma fermentação.

Na segunda opção, o utilizador pode seleccionar quais as variáveis de estado, a serem alvo da optimização. Na terceira opção, o utilizador tem que definir um intervalo de tempo, sobre o qual vai ser determinado o melhor tempo (ponto) de paragem. Tal como apresentado na simulação, podem ser realizadas várias optimizações aplicando várias conjugações com diferentes conjuntos de valores iniciais das variáveis de estado e conjuntos de valores de parâmetros, tal como ilustrado na Figura 5.

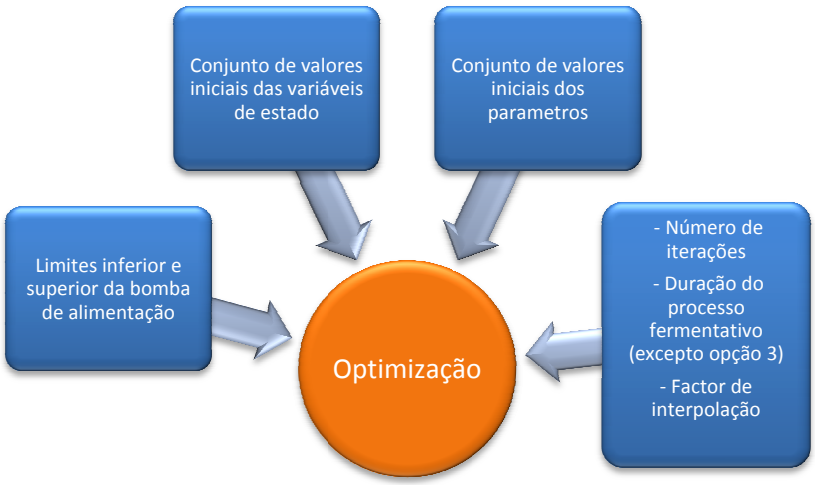


Figura 5: Dados que deverão ser definidos para realizar as tarefas de optimização

A alimentação de substrato ao reactor é, na maioria dos casos, realizada através de uma bomba (normalmente peristáltica), tornando-se muitas vezes um factor limitante devido ao caudal máximo que as bombas conseguem debitar para o interior de um reactor. Assim, o utilizador tem a possibilidade de definir limites mínimo e máximo referentes ao caudal da bomba, servindo estes como restrições nas operações de cálculo do perfil de alimentação. Existe um outro parâmetro, denominado de factor de interpolação (F_i) que também pode ser manipulado. Como o tamanho da solução final é calculado com base na relação entre o tempo total de fermentação (T_f) e o intervalo de discretização $\left(\frac{Tf}{dt}\right)$, o que daria uma solução extensa, foi adicionado o parâmetro F_i que, ao multiplicar pelo intervalo de discretização $\left(\frac{Tf}{dt \cdot F_i}\right)$, estabelece um intervalo de tamanho fixo, originando uma solução com $S = \left(\frac{Tf}{dt \cdot F_i}\right) + 1$ pontos .

Os resultados das optimizações podem ser visualizados através de um gráfico ou de uma tabela, onde é apresentado o caudal de substrato a alimentar ao longo do tempo. Todas as informações referentes às condições utilizadas em cada optimização são associadas internamente a essas optimizações, de modo a que numa fase posterior se possa ter acesso a essas informações. Pode ainda ser gerado um relatório final em formato PDF ou ASCII no qual são descritas as condições utilizadas e os resultados obtidos.

3.5. Estimação de Parâmetros

Tal como nas operações de simulação e optimização, o utilizador pode conjugar diferentes conjuntos de valores iniciais de variáveis de estado, valores de parâmetros, perfis de alimentação e dados experimentais para realizar a estimação, como exposto na Figura 6. O utilizador pode optar por utilizar ou não um perfil de alimentação previamente definido.

Figura 6: Dados que deverão ser definidos para proceder às tarefas de estimação de parâmetros.

É dada ao utilizador a opção de fixar cada um dos parâmetros, de modo a excluí-los das operações de estimação, permanecendo sempre com o valor constante durante as operações. Um procedimento semelhante pode ser aplicado às variáveis de estado, mas neste caso exclui-se uma variável ou várias variáveis de estado das operações de estimação. Ao excluir uma variável, esta não vai ser associada aos cálculos da função de custo total (equação 12). Esta função permite verificar a aproximação dos dados simulados aos dados experimentais, com a alteração dos vários parâmetros no decorrer da estimação. A aproximação destes dados é realizada minimizando o custo total, como apresentado a seguir:

$$Total\ Cost = \sum_{i=1}^{n}\left(\frac{1}{N_p}\sum_{j=1}^{p}\left(\frac{\xi_{sim,ij}-\xi_{\exp,ij}}{\overline{\xi}_{\exp,ij}}\right)^2\right) \tag{12}$$

onde

$\xi_{sim,ij}$ representa os dados simulados

$\xi_{\exp,ij}$ representa os dados experimentais, referentes a cada variável ξ (*N* é o numero de variáveis de estado) em cada ponto (*p* é o numero total de pontos).

A diferença é dividida pelo valor médio $\overline{\xi}_{\exp,ij}$ de cada variável ξ com o propósito de atribuir a mesma importância a todas as variáveis de estado.

Os resultados são apresentados ao utilizador por meio de uma tabela. Contudo, estes ficam logo disponíveis na consola de simulação, permitindo ao utilizador executar uma simulação e visualizar a aproximação dos dados simulados aos dados experimentais como referido no ponto 3.3. Pode ser gerado um relatório em formato PDF ou ASCII onde consta toda a informação referente às condições utilizadas e aos resultados obtidos.

Capítulo 4

Implementação da ferramenta OptFerm

No presente capítulo, faz-se uma descrição pormenorizada de como foi implementada a ferramenta *OptFerm*, e está seccionado da seguinte forma:

4.1. A plataforma *AIBench*

A ferramenta *OptFerm* foi desenvolvida sobre uma aplicação de nome *AIBench Framework*. Esta ferramenta foi desenhada seguindo um conceito de "**Model–View–Controller"** (MVC), no qual o Modelo (*Model*) representa a informação e dados da aplicação, a visualização (*View*) corresponde à interface com o utilizador e o controlador (*controller*) faz toda a gestão da comunicação dos dados entre a interface e o modelo. Este paradigma permite ao *AIBench* tornar-se uma aplicação leve, não intrusiva e permite a conexão e execução de operações com entradas/saídas bem definidas, facultando uma rápida visualização do resultado final dessas mesmas operações.

O conceito associado a esta aplicação (*AIBench)* torna-a uma ferramenta de programação poderosa, pois faculta um rápido desenvolvimento de novas ferramentas informáticas, permitindo aplicar as seguintes premissas:

- Dissociar a lógica das operações da interface com o utilizador.
- A conexão entre operações pode ser dissociada rapidamente e seguir um conceito de ensaio.
- O programador é forçado a pensar primeiro antes de programar e realizar depois, facilitando a reutilização de código.
- Novas operações podem ser facilmente adicionadas, bastando adicionar os ficheiros *.jar* num directório específico.

Para realizar todas as premissas supracitadas, o *AIBench* segue os seguintes princípios:

- O programador deve ter a possibilidade de executar as suas operações com o menor número de linhas de código possível. A plataforma deve fornecer de forma inteligente padrões para cada operação, podendo ser estes aperfeiçoados no futuro.
- Independente do problema: o *AIBench* é agnóstico ao tipo de dados, sendo o programador que designa o tipo de dados e a transição entre as suas classes

4.1.1. Arquitectura do AIBench

O *AIBench* utiliza um sistema baseado em *plugins*, provendo este com capacidades avançadas, tais como a descoberta dinâmica e o arranque de novas operações com um simples restaurar da aplicação. A Figura 7 apresenta um esquema geral do funcionamento do *AIBench,* demonstrando como se inter-relacionam os seus diferentes módulos. O motor de *plugins* (*Platonos*)[56] do *AIBench* permite criar novas aplicações com uma filosofia modular, adoptando os seguintes paradigmas:

- Um *plugin* é um conjunto de classes isoladas das restantes classes presentes na aplicação.
- Um *plugin* pode utilizar classes de outro *plugin* apenas no caso de estas dependências terem sido definidas.
- Um *plugin* pode definir pontos de extensão, isto é, um local onde outros plugins poderão ser ligados e utilizados imediatamente.

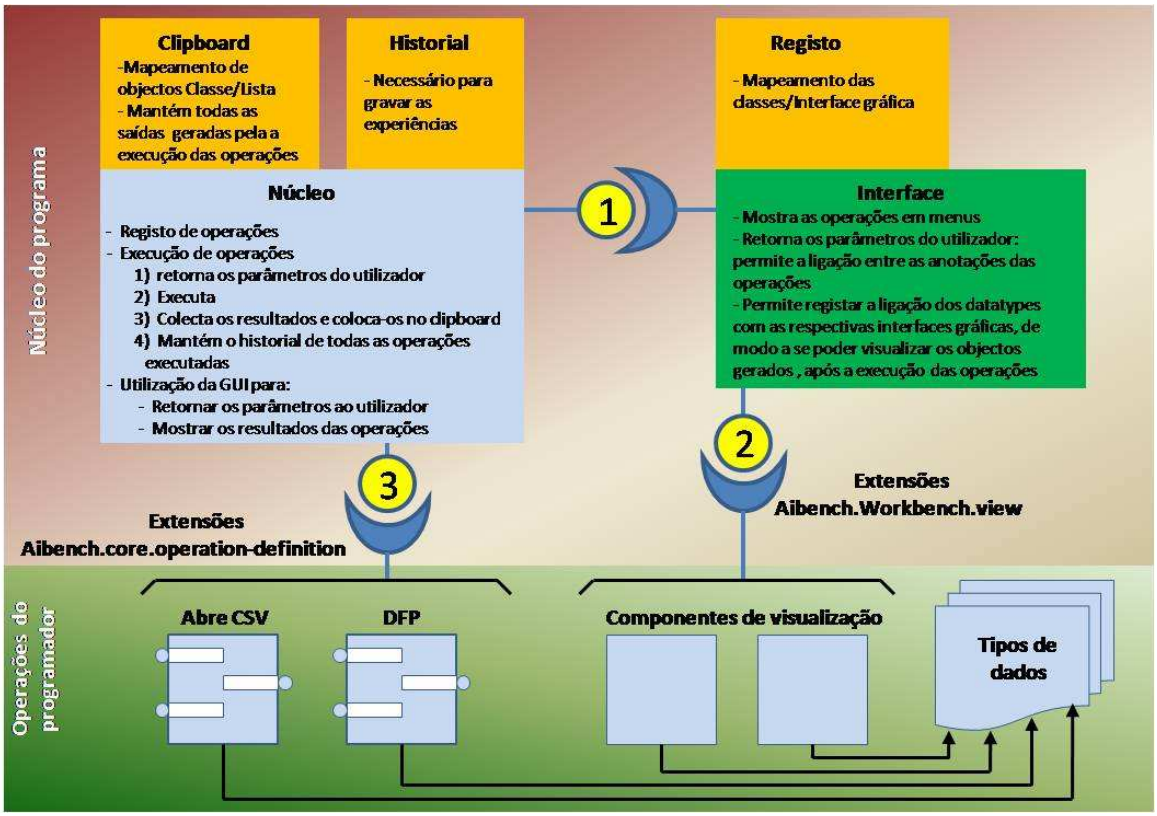


Figura 7: Arquitectura do *AIBench* (fonte http://www.aibench.org/)

Existem duas secções distintas no *AIBench*, como é demonstrado na Figura 7: a zona verde corresponde às operações e tipos de dados definidos pelo programador; a zona vermelha correspondente ao funcionamento interno do *AIBench*, incluindo o seu núcleo (*Core*) e a interface (*WorkBench*). Estes dois *Plugins* comunicam internamente por um ponto de ligação (Figura 7, ponto 1). De modo semelhante, o *Core* implementa um ponto de extensão "AIBench.core.operation-definition" (Figura 7, ponto 3) no qual as operações do programador podem ser conectadas à aplicação. Podem ser adicionadas novas operações, bastando referir no *plugin* este ponto de ligação, não sendo necessário qualquer tipo de interface. O *WorkBench* tem um ponto de ligação associado ao tipo de dados através de uma extensão designada "AIBench.core.WorKbench.view" (Figura 7, ponto 2). Esta extensão permite a visualização dos dados bastando definir no *plugin* a conexão de determinado objecto a este ponto. Todas as operações, tipos de dados e visualizações podem estar

conectadas a uma ou mais extensões, desde que se definam, se necessário, as suas dependências.

4.1.2. Modelo de Operação do AIBench

As aplicações desenvolvidas com o *AIBench* seguem a sua estruturação através de operações (*Operations*), tipos de dados (*Datatypes*) e interfaces gráficas de visualização (*Views*).

Os *Datatypes* permitem a estruturação dos dados na aplicação e a sua apresentação ao utilizador final. O conteúdo dos *Datatypes* pode ser visualizado pelo utilizador final através de *Views*, de acordo com a permissão e definição do programador. A estruturação dos dados é visualizada num painel lateral designado de *Clipboard*, no qual os objectos são dispostos de forma hierárquica. Apesar de o *AIBench* ser agnóstico em relação ao tipo de dados, têm que ser definidas anotações para que estes possam ser manipulados, sendo estas:

- **SIMPLE** – assente já por pré-definição, compreende apenas uma instância do objecto.
- **LIST** – anota *Datatypes* com uma lista dinâmica de sub-elementos, composta por outros *Datatypes*, ficando estes automaticamente dispostos no *Clipboard*.
- **COMPLEX** – Um *Datatype* constituído por outros *Datatypes* do tipo *SIMPLE*, *COMPLEX* e *LIST*.

Uma operação descreve uma função que recebe dados ou objectos como entrada e retorna objectos como saída. Porém, também podem ser definidas operações de modificação de dados de determinado objecto. É considerada uma operação uma classe Java que contenha métodos com portas associadas a estes, e essas portas podem ser definidas da seguinte forma:

- **Input** – recebe dados de entrada através de um parâmetro.
- **Output** – retorna os dados produzidos.
- **Input/Output** – produz ambos os resultados descritos em cima.

4.2. Módulos de operações (BioFerm)

Na biblioteca principal denominada de *BioFerm* estão encapsulados todos os módulos que permitem executar todas as operações incluídas nesta aplicação (simulação, optimização e estimação de parâmetros). Os 5 módulos principais são:

- *bio* – biblioteca onde estão contidos os processos e as funções referentes a cada um dos modelos; contém também funções para criar novos conjuntos de valores iniciais e de limites (inferior e superior) das variáveis de estado e parâmetros cinéticos e de rendimento, e ainda funções para criar perfis de alimentação definidos pelo utilizador.

- *gaferm* – contém os algoritmos para executar os três tipos de optimização supracitados, utilizando funções da biblioteca *Jecoli*.
- *Jecoli* (Java Evolutionary Computation Library) – biblioteca genérica que incorpora algoritmos meta-heuristicos, nomeadamente: algoritmos genéticos, de evolução diferencial, *simulated annealing* e algoritmos evolucionários para optimizações multi-objectivo [57]. Toda a informação relacionada com esta biblioteca pode ser encontrada em http://darwin.di.uminho.pt/jecoli.
- *parameterEstima* – agrupa os algoritmos para realizar as operações de estimação de parâmetros, recorrendo às bibliotecas *Jecoli*, *bio* e *odetojava* no decorrer das operações.
- *Odetojava* – biblioteca com todos os algoritmos para realizar as simulações numéricas. Os algoritmos existentes foram desenvolvidos por Ascher et al. [55] tendo por base rotinas de Runge-Kutta.

Os restantes módulos contêm funcionalidades de suporte às operações dos módulos principais.

4.2.1. Modelos

Definiram-se duas classes, *FermProcess* e *FermFunction,* que servem como classes de nível superior (super) a todas as outras classes representantes de modelos biológicos. Ao criar um novo modelo de um sistema biológico, é obrigatório escrever uma subclasse de *FermFunction* com os seguintes parâmetros:

- Método para iniciar um objecto do tipo *FermParameterValues* que contém os nomes e os valores dos parâmetros (cinéticos, de rendimento);
- Método para iniciar um objecto do tipo *FermParameterLimits* que contém os limites inferiores e superiores de todos os parâmetros.
- Equações diferenciais;
- Equações cinéticas;

Numa outra subclasse associada ao *FermProcess* têm que estar definidos:

- A inicialização de um objecto do tipo *FermStateValues* contendo os nomes e os valores iniciais das variáveis de estado;
- A inicialização de um objecto do tipo *FermVarLimits* contendo os limites inferiores e superiores das variáveis de estado;
- Função objectivo (produtividade ou rendimento).

A classe *FermProcess* é a classe principal ao criar e usar um modelo, ou seja, quando se cria uma instância de *FermProcess*, esta tem que ter associados a si os objectos do tipo *FermVarLimits*, *FermStateValues* e *FermFunction*. Por sua vez, o objecto *FermFunction* tem que conter os objectos *FermParameterValues* e *FermParameterLimits*. A Figura 8 apresenta um esquema simplificado de como se relacionam os diferentes objectos incluídos num modelo, quando instanciados.

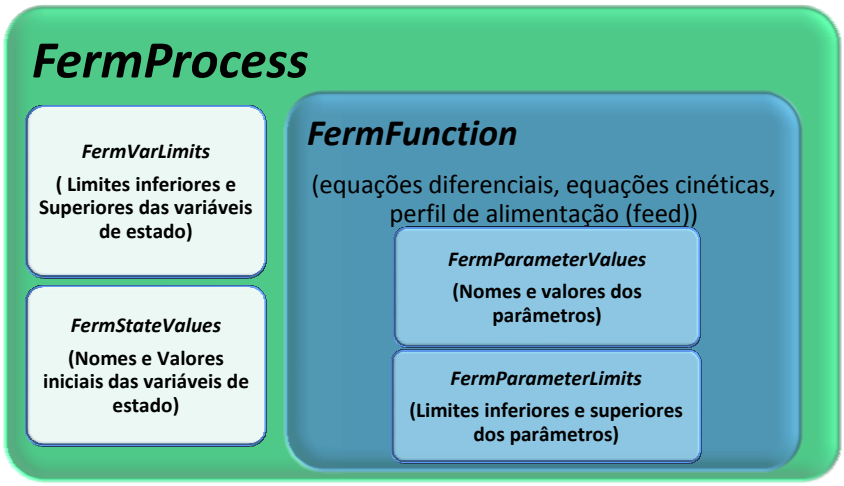


Figura 8: Esquema conceptual de como estão relacionados os diferentes objectos respeitantes a um modelo

As classes apresentadas na Figura 8 têm que estar obrigatoriamente definidas de forma a instanciar qualquer objecto do tipo *FermProcess*. Porém, a classe *FermFunction* contém um atributo respeitante à alimentação (*feed*), o qual não é obrigatório estar definido inicialmente. Este atributo pode ser definido numa fase posterior.

4.2.2. Simulação numérica

Para realizar a simulação numérica foram escritas duas classes *ErkFerm* e *ImexFerm* que se encontram no pacote *bio*, sendo invocadas pelo *FermProcess* sempre que necessário. Estas foram desenvolvidas de forma a apropriar a simulação numérica aos requisitos da ferramenta *OptFerm*. Estas são subclasses das classes *Imex* e *Erk* presentes na biblioteca *odetojava* [55]. Na Figura 9, é mostrado um esquema geral exemplificando como se inter-relacionam as duas bibliotecas *bio* e *odetojava*. Implementaram-se duas rotinas de Runge-kutta, uma que utiliza um método implícito-explícito de passo linear múltiplo (*IMEX*), adequada à resolução de problemas não lineares complexos (*stiff*) e outra que utiliza um método explícito de passo constante (*ERK*), adequado para a resolução de problemas menos complexos (*non-stiff*).

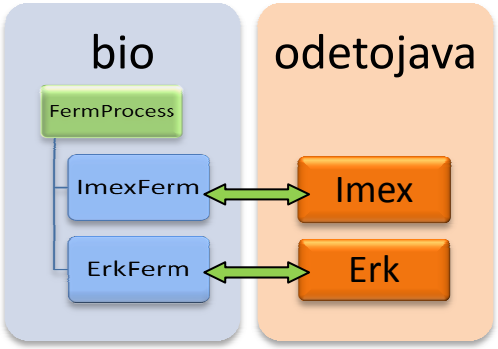


Figura 9: Esquema geral, representando a inter-ligação das classes mais importantes que permitem realizar a simulação numérica.

4.2.3. Optimização

Todas as operações específicas para execução de tarefas de optimização na aplicação *OptFerm* estão encapsuladas no pacote *gaferm*. As classes presentes em *gaferm* foram escritas de modo a permitirem realizar os seguintes tipos de optimização:

- Perfil de alimentação de determinado substrato ao longo do tempo.
- Determinação dos valores iniciais das variáveis de estado (mais adequados) para iniciar uma fermentação, e o respectivo perfil de alimentação de substrato.
- Prever qual o melhor tempo para finalizar a alimentação de substrato, dentro de determinado intervalo definido pelo utilizador, e optimizar o respectivo perfil de alimentação.

A biblioteca *gaferm* utiliza as funções presentes numa biblioteca mais geral *Jecoli*, que contém rotinas de optimização baseadas em Algoritmos Evolucionários, Evolução Diferencial e *Simulated Annealing.* Na Figura 10, é descrito um esquema geral de como interagem as diferentes operações durante as tarefas de optimização.

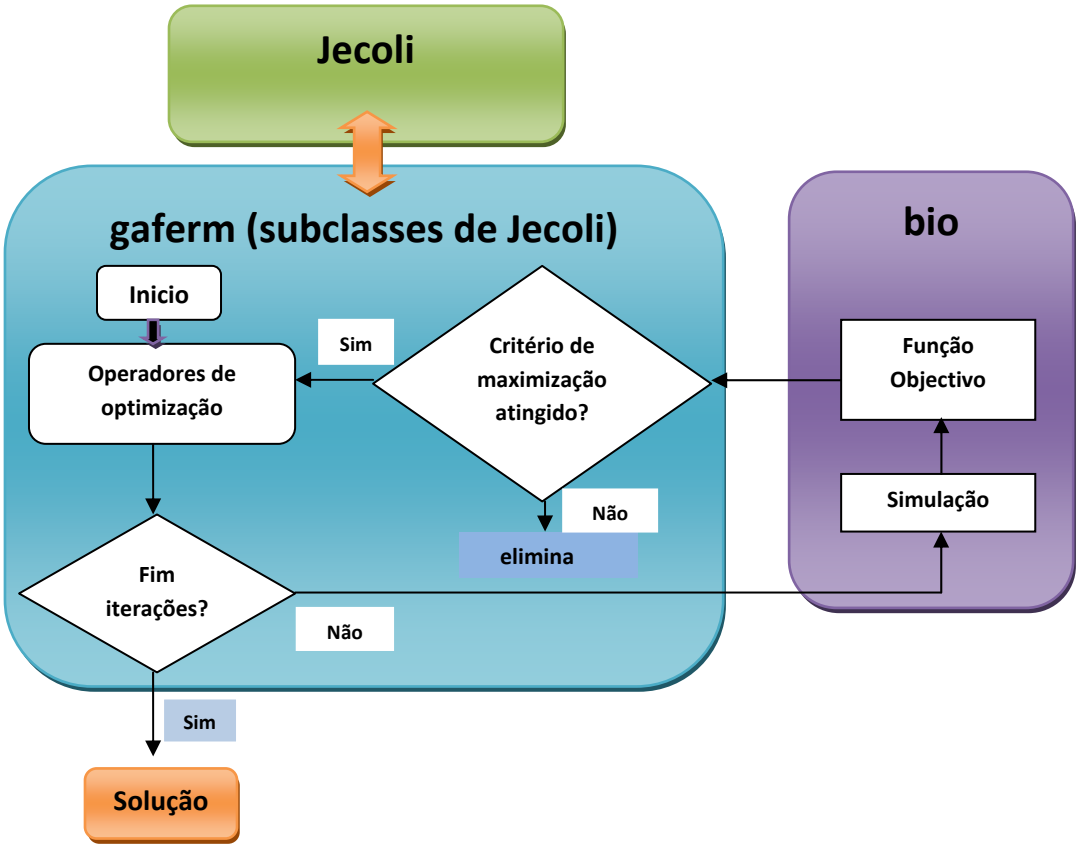


Figura 10: Esquema ilustrativo de como são realizadas as tarefas de optimização e quais as bibliotecas envolvidas no processo.

A *gaferm* contém as principais funções para realizar as tarefas de optimização. A maioria das classes contidas nesta biblioteca são subclasses de classes presentes na biblioteca *Jecoli.* Estas subclasses tiveram de ser criadas de modo a dotar a aplicação *OptFerm* com a capacidade de realizar as optimizações supracitadas. Como se pode observar na Figura 10 as funções relacionadas com os operadores de optimização (selecção, cruzamento, mutação, etc.), avaliação e decisão dos resultados obtidos estão incluídas em *gaferm*, mas utilizam funções incorporadas em *Jecoli*. As funções relacionadas com os modelos, simulação numérica e a função objectivo estão incluídas na biblioteca *bio*, sendo estas convocadas sempre que necessário no decorrer das optimizações.

4.2.4. Estimação de parâmetros

Para executar as operações de estimação de parâmetros foram criadas duas classes específicas estando estas encapsuladas no pacote *parameterEstima*. Em conjunto, estas classes realizam as tarefas de estimação dos parâmetros. Uma das classes (*EstimAlgorithms*) contém as rotinas responsáveis pela determinação de novos valores dos parâmetros, e no decorrer do processo são convocadas funções presentes na biblioteca *Jecoli*. Esta classe está ainda responsável por configurar inicialmente o algoritmo de optimização com as devidas condições iniciais e restrições, tais como:

- Escolha do algoritmo de optimização para realizar a estimação.
- Configuração do algoritmo de optimização (número de iterações, população, etc.).
- Definição dos valores iniciais dos parâmetros cinéticos e de rendimento.
- Definição dos limites mínimos e máximos que os parâmetros poderão atingir.

Após a definição dessas condições e restrições, estes argumentos são passados para o algoritmo de optimização escolhido (presente na biblioteca *jecoli*) e procede-se à determinação e avaliação dos novos parâmetros obtidos no decorrer da estimação.

As rotinas de avaliação estão descritas noutra classe, *EvalEstimParam*, que é uma subclasse de uma classe *Eval* presente na *Jecoli*. Esta classe está responsável por avaliar o valor da função objectivo com base na aproximação dos dados obtidos nas simulações aos dados experimentais, ou seja, está responsável por avaliar a convergência da função de custo total (equação 12) para um mínimo global. Na Figura 11 é apresentado um esquema simplificado de como se processam as tarefas de estimação de parâmetros.

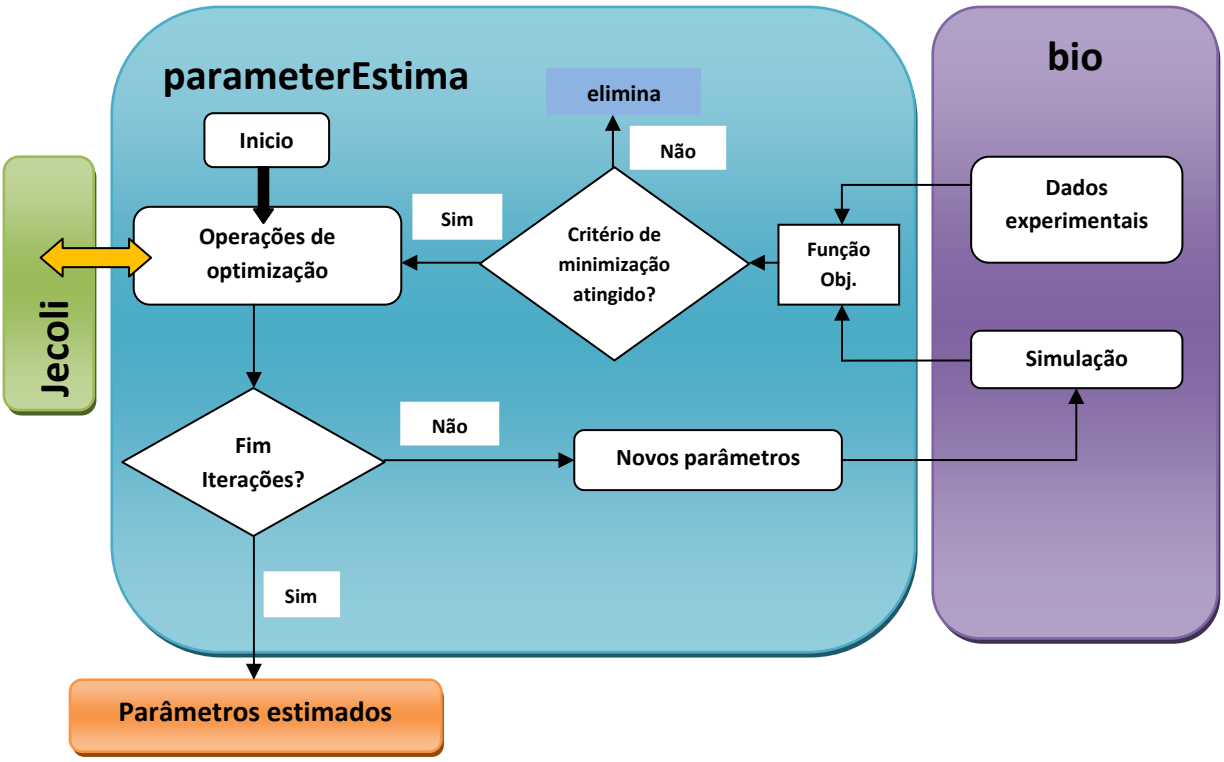


Figura 11: Esquema ilustrativo de como são realizadas as tarefas de estimação de parâmetros e quais as bibliotecas envolvidas no processo.

4.3. Integração no *AIBench*

Ao desenvolver a aplicação *OptFerm* definiram-se os tipos de dados (*Datatypes*), operações (*controller*) e as interfaces gráficas (*Views*). De igual modo, desenvolveram-se também todas as bibliotecas externas, contendo as funções para a realização das tarefas de optimização, simulação e estimação de parâmetros.

Na Figura 12 está descrito um esquema geral de como foi implementada a aplicação *OptFerm*, no qual se pode observar o tipo de objectos que tiveram de ser definidos. Todos os pontos de ligação das operações à ferramenta *AIBench* foram escritas num ficheiro designado *plugin.xml*, podendo-se considerar este ficheiro equivalente às setas que unem o *AIBench* e *OptFerm* na Figura 12. É neste ficheiro, escrito em *XML* que se define como ficarão dispostas as várias operações no menu principal da aplicação, como são ligadas as operações e se associa uma determinada interface gráfica a uma determinada operação ou *datatype*, seja ela de entrada ou saída de dados. Este conceito permite uma grande flexibilidade no futuro, caso seja necessário alterar alguma interface gráfica ou acrescentar novas operações.

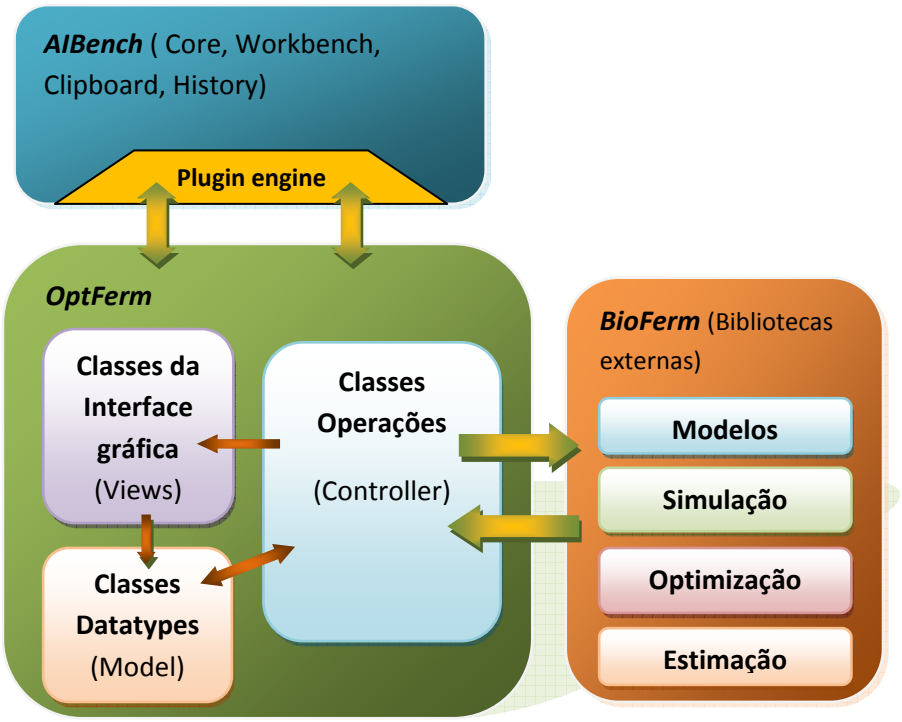


Figura 12: Esquema conceptual de como foi implementada a aplicação OptFerm sob a aplicação AIBench.

4.3.1. Tipos de Dados (Datatypes)

Ao desenvolver a aplicação foi necessário definir todos os tipos de dados (*Datatypes*) que iriam estar presentes na aplicação e o modo como estes deveriam ser apresentados no *ClipBoard* (como seriam apresentados ao utilizador). Na Figura 13 é apresentado um esquema em árvore de todos os tipos de dados utilizados na aplicação e as respectivas classes que instanciam os devidos objectos. O esquema está estruturado da mesma forma como são apresentados os diferentes objectos ao utilizador, no *Clipboard* da aplicação *OptFerm*.

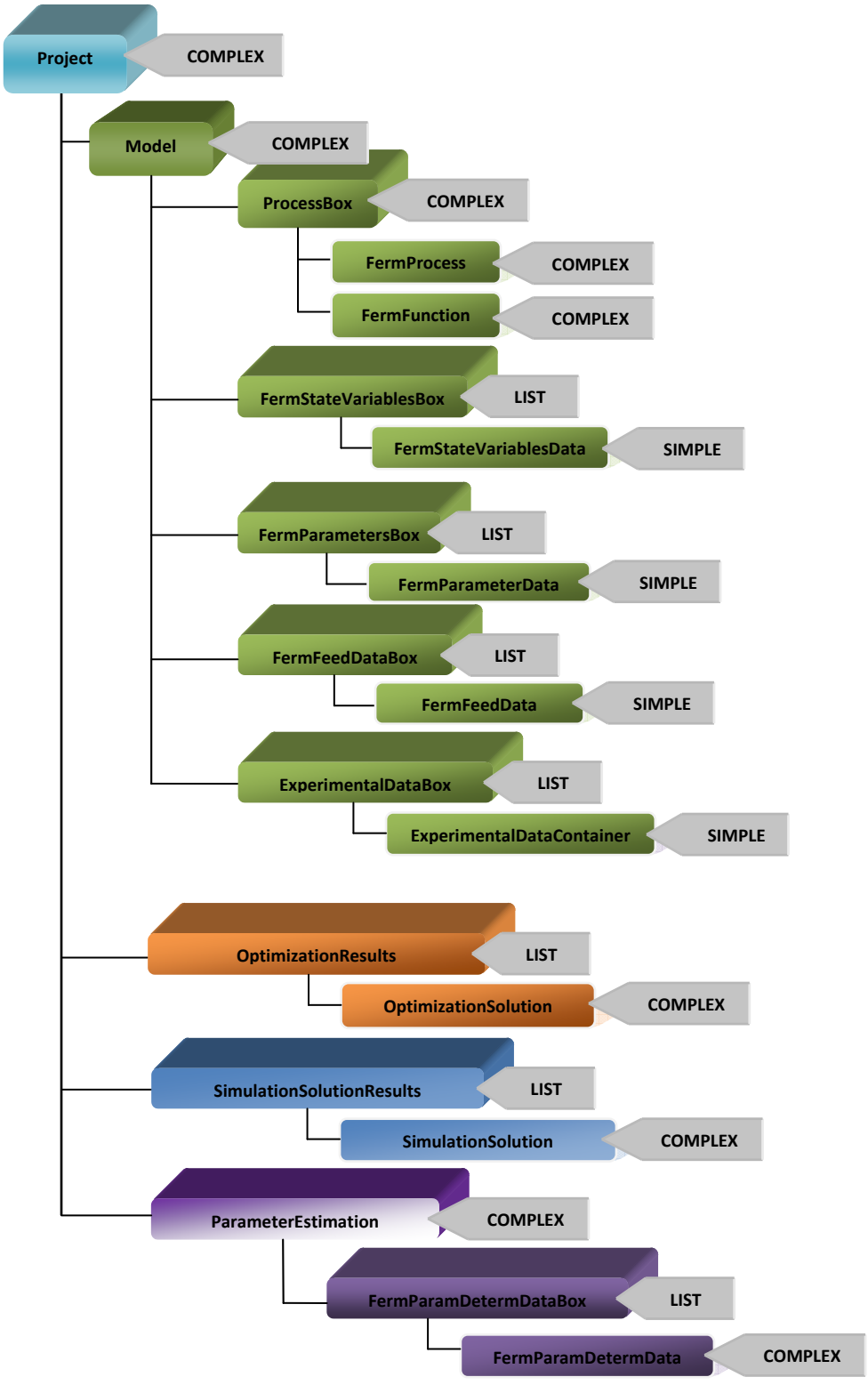


Figura 13: Esquema ilustrativo de como foram estruturados os *Datatypes* e as respectivas anotações associadas a cada um dos *Datatypes*.

A estrutura em árvore apresentada na Figura 13 também tem como objectivo explicar como os diferentes objectos estão incorporados uns nos outros. Por exemplo, um objecto *Project* (do tipo complexo) engloba todos os outros objectos descritos na Figura 13.

No início, ao criar um novo projecto, são imediatamente instanciados os vários objectos descritos em cima, com excepção do *FermFeedData*, *FermParameterData*, *SimulationSolution*, *ExperimentalDataContainer* e *FermParamDetermData*. Isto deve-se ao facto de serem tipos de dados contidos numa lista, sendo estes objectos apenas instanciados após a execução de operações específicas, tais como a realização de uma optimização, simulação ou estimação de parâmetros. No caso do *FermFeedData* o utilizador poderá não querer realizar nenhuma operação com perfil de alimentação, logo só é instanciado um objecto deste tipo quando o utilizador final definir uma alimentação. Estes objectos guardam cópias dos objectos utilizados nas operações que originaram a sua instanciação, de modo a que o utilizador possa visualizar em qualquer etapa os dados utilizados nessas operações (optimização, simulação, estimação de parâmetros).

4.3.2. Operações

Foram desenvolvidas várias classes com as devidas funções de forma a prover a aplicação *OptFerm* com capacidades de criar, guardar ou remover instâncias de objectos ou então reabrir projectos anteriormente guardados. Similarmente, foram desenvolvidas as várias classes que fazem a ligação às bibliotecas externas de modo a realizar as diversas operações de optimização, simulação e estimação de parâmetros. As classes desenvolvidas e incorporadas no *OptFerm* para realizar as diversas operações foram as seguintes:

Fermentationplugin.operations.Creators

- ***NewProject*** - Criar um novo projecto;
- ***NewStateValuesSet*** - Criar uma nova instância do objecto *FermStateVariablesData;*
- ***NewParametersValuesSet*** - Criar uma nova instância do objecto *FermParameterData*;
- ***NewFeed*** – Criar uma nova instância do objecto *FermFeedData*;
  NewExperimentalDataSet- Criar nova instância do objecto *ExperimentalDataContainer*

Fermentationplugin.operations.Loaders

- ***LoadProject***- Reabrir um projecto guardado anteriormente

Fermentationplugin.operations.Savers

- ***SaveProject***- Guardar um projecto presente no *Clipboard.*

Fermentationplugin.operations.simulation

- ***Simulation*** – Estabelecer os argumentos a passar ao pacote BioFerm para efectivar uma simulação.

Fermentationplugin.operations.Removers

- ***RemoveStateVar*** - Remover um objecto do tipo FermStateVariablesData presente em FermStateVariablesBox.
- ***RemoveParamSet*** - Remover um objecto do tipo FermParameterData presente em FermParametersBox.
- ***RemoveFeed*** - Remover um objecto do tipo FermFeedData presente em FermFeedDataBox.

Fermentationplugin.operations.optimization

- ***GA_Optimization*** – Configura e passa os argumentos necessários para o algoritmo genético presente na biblioteca Jecoli de modo a executar a optimização.
- ***DE_Optimization*** – Configura e passa os argumentos necessários para o algoritmo de evolução diferencial presente na biblioteca Jecoli de modo a executar a optimização.
- ***SA_Optimization***– Configura e passa os argumentos necessários para o algoritmo *simulated annealing* presente na biblioteca Jecoli de modo a executar a optimização.

Fermentationplugin.operations.ParameterEstimation

- ***ParameterEstimation*** – Configura o algoritmo de optimização e as condições que vão ser utilizadas nas tarefas de estimação de parâmetros, e passa esses argumentos para as classes presentes na biblioteca externa parameterEstima de modo a executar as operações.

4.3.3. Interface Gráfica

Para todas as operações apresentadas em cima foram realizadas interfaces gráficas, de modo a garantir ao utilizador facilidade de utilização e compreensão. Todas as interfaces gráficas foram desenvolvidas com o *Jigloo GUI Builder* (toda a informação sobre esta ferramenta pode ser encontrada em http://www.cloudgarden.com/jigloo/). Tal como já foi referido na secção 4.3, foram desenvolvidos dois tipos de interfaces gráficas. Um que permite visualizar e aceder aos dados contidos em cada um dos *Datatypes,* ao qual se denominou de *views*, e um

outro tipo denominado de *GUI* (*Graphical User Interface*) que está directamente associado à execução das diversas tarefas na aplicação OptFerm. As GUI foram desenhadas com o único propósito de proporcionar ao utilizador um instrumento simples para fornecer os vários argumentos (dados) de modo a executar as diferentes tarefas, podendo-se considerar como interfaces de entrada de dados. A associação de cada uma das interfaces gráficas às devidas operações ou *Datatypes* foi definida no ficheiro plugin.xml. No caso de estudo demonstrado no próximo capítulo podem-se visualizar a maioria das interfaces desenvolvidas para a aplicação *OptFerm*. As classes desenvolvidas para a visualização dos *Datatypes* foram as seguintes:

Fermentationplugin.views

- **ViewStateValues** - Visualização das variáveis de estado e dos limites definidos no objecto *FermStateVariablesData*.
- **ViewParameters** - Visualização dos parâmetros e dos limites inferiores e superiores definidos no objecto *FermParameterData.*
- **ViewFeed** – Visualização da alimentação ao longo do tempo definida pelo utilizador no objecto *FermFeedData.*
- **ViewOptimizationSolution** – Visualização dos resultados obtidos na optimização; sãoapresentados uma tabela e um gráfico com o perfil de alimentação ao longo do tempo.
- **SimulationViewGraph** - Visualização dos resultados obtidos nas simulações. Tem associadas a si as condições utilizadas em cada uma das diferentes simulações.
- **ExperimentalDataView** – Visualização dos dados experimentais inseridos pelo utilizador para a estimação de parâmetros.
- **EstimatedParametersView**- Visualização dos parâmetros obtidos na estimação.
- **ProjectInformationView –** Permite ver a informação referente a um projecto, as variáveis de estado, o numero de parâmetros presentes no modelo, a data de criação do modelo, o modelo utilizado, etc.
- **View_experimental_simulation_data -** Permite comparar os dados simulados com os dados experimentais.
- **ViewUsedEstimSettings** – Para a visualização das condições utilizadas no algoritmo de optimização após a execução de uma determinada optimização.

As classes desenvolvidas para criação de novos conjuntos de dados (variáveis de estado, parâmetros, dados experimentais e perfis de alimentação) e para a execução das tarefas de simulação, optimização e estimação de parâmetros foram as seguintes:

Fermentationplugin.GUI

- **CreateStateVariablesSet_GUI** – Permite criar um novo conjunto com novos valores iniciais das variáveis de estado.
- **CreateParametersSet_GUI** - Permite criar um novo conjunto com novos valores para os parâmetros presentes no modelo (cinéticos e de rendimento).
- **ExperimentalDataSetGUI** – Permite definir um novo conjunto de dados experimentais.
- **MakeFeedSet_GUI_table** – Permite ao utilizador definir um novo perfil de alimentação ao longo do tempo, onde o utilizador pode definir os intervalos de tempo e os respectivos caudais nesses intervalos.

- **OptimizationDE_GUI** – Interface para realizar as tarefas de optimização, as quais são executadas utilizando o algoritmo de evolução diferencial.
- **OptimizationGA_GUI** - Interface para realizar as tarefas de optimização, as quais são executadas utilizando o algoritmo genético.
- **OptimizationSA_GUI** - Interface para realizar as tarefas de optimização, as quais são executadas utilizando o algoritmo *simulated annealing*.
- **Simulation_GUI** - Interface para realizar as tarefas de simulação.
- **ParameterEstimation_GUI** - Interface para realizar as tarefas de estimação de parâmetros.

Capítulo 5

Caso de Estudo

No presente capítulo é apresentado um caso de estudo onde são exemplificadas as várias funcionalidades da aplicação *OptFerm.* O objectivo deste estudo foi demonstrar as capacidades da aplicação e não fazer um estudo detalhado do modelo utilizado. Porém, foi realizado um pequeno estudo com o modelo apresentado, com o propósito de analisar a robustez da aplicação nas tarefas de optimização e estimação de parâmetros, sendo este efectuado com base no método de Monte Carlo. O capítulo está dividido do seguinte modo:

5.1. Modelos

Como referido anteriormente, o elemento base de todas as operações na aplicação *OptFerm* é o respectivo modelo que caracteriza os processos fermentativos em modo semi-continuo. Todos os modelos utilizados nesta aplicação devem ser representados por equações diferenciais.

No caso de estudo apresentado foi utilizado um modelo que caracteriza uma fermentação em semi-continuo para produção de etanol pela levedura *Saccharomyces cerevisiae,* descrito pelos autores Chen and Huang [58]. O sistema é representado pelas seguintes equações diferenciais:

$$\frac{dx_1}{dt} = g_1 x_1 - u\frac{x_1}{x_4} \tag{13}$$

$$\frac{dx_2}{dt} = -10 g_1 x_1 + u\frac{150 - x_2}{x_4} \tag{14}$$

$$\frac{dx_3}{dt} = g_2 x_1 - u\frac{x_3}{x_4} \tag{15}$$

$$\frac{dx_4}{dt} = u \tag{16}$$

onde x_1, x_2 e x_3 representam as concentrações de biomassa, substrato e etanol (g/L), x_4 representa o volume do reactor (L) e u o caudal de alimentação de substrato ao reactor (L/h). As variáveis cinéticas g_1 e g_2 são definidas pelas seguintes equações:

$$g_1 = \frac{0.408}{1 + \frac{x_3}{16}} \frac{x_2}{0.22 + x_2} \tag{17}$$

$$g_2 = \frac{1}{1 + \frac{x_3}{71.5}} \frac{x_2}{0.44 + x_2} \tag{18}$$

Foi definida uma função objectivo com o intuito de obter a produtividade máxima de etanol quando o volume útil do reactor x_4 fosse atingido, da seguinte forma:

$$prod = x_3(T_f)x_4(T_f) \tag{19}$$

em que T_f representa o tempo final de fermentação.

Primeiramente, foi necessário escrever duas classes Java, a classe *EthSCProcess* que iria ser uma subclasse da classe principal *FermProcess* e a *EthSCFunction* que iria ser subclasse da classe *FermFunction* como explicado em 4.2.1.

Na classe *EthSCProcess* definiram-se os nomes num vector *VarNames* e os valores iniciais das variáveis de estado num vector *VarValues* como apresentado na Figura 14. Como se pode verificar, os nomes das variáveis de estado do modelo apresentado em cima foram alterados para *X*, *S*, *P* e *V* correspondentes a Biomassa, substrato, etanol, e volume do reactor para melhor identificação. Foram definidos os limites inferiores e superiores das variáveis de estado, e no final foi descrita a função objectivo como apresentado na Figura 15, que iria ser utilizada posteriormente nas operações de optimização do perfil de alimentação. Esta função deve ser definida com imenso cuidado visto que as operações matemáticas realizadas durante o processo de optimização são baseadas na maximização ou minimização do valor desta função.

```
public FermStateValues defaultStateValues()
 {
     double[] VarValues = new double[4];
     VarValues[0] = 1.0; // X1
     VarValues[1] = 150.0; // X2
     VarValues[2] = 0.0; // X3
     VarValues[3] = 10.0; // X4

      String [] varNames = new String[4];
        varNames[0] = "X";
        varNames[1] = "S";
        varNames[2] = "P";
        varNames[3] = "V";

      FermStateValues StateValues = new FermStateValues(VarValues,varNames);
      return StateValues;
 }
```

Figura 14: Método *defaultStateValues* onde se definiram os nomes e valores iniciais das variáveis de estado.

```
// process productivity: (X3final*X4final) per time unit
public double productivity (double tf)
{

   double prod = u[2][endPoint] * u[3][endPoint];
   return prod;
}
```

Figura 15: Método *productivity* onde se definiu a função objectivo para ser utilizada nas tarefas de optimização.

Na classe *EthSCFunction* descreveram-se os nomes dos parâmetros num vector *parNames* e os respectivos valores iniciais num vector designado *modelPars* de forma semelhante ao apresentado na Figura 14. Foram definidos também os limites inferiores e superiores desses parâmetros, as equações diferenciais ordinárias (ODEs)

e as equações cinéticas. As equações diferenciais foram transcritas para linguagem Java, sendo estas incorporadas num método (Figura 16) no qual em cada instante de tempo *t* se recebe um vector com os valores das variáveis de estado calculadas no tempo *t-1*. No decorrer da simulação, os valores das cinéticas eram actualizados através da convocação da função *updateKineticCoef(t)*, e após o cálculo dos valores das variáveis de estado correspondentes ao instante de tempo *t,* estas eram guardadas num vector *xp* de modo a serem utilizados na iteração seguinte.

```
// model diff. equations
public double[] f(double t, double[] x)
{
    double[] xp = new double[x.length];

    updateKineticCoefs(t);

    kinetics(x[1], x[2]); // X2, X3

    double u;
        if (super.feedon==false){
            u = 0;}
        else
        {
           u = feed(t);//feed(t);}
        } // F

    //dX1/dt = g1*X1 - u*(X1/X4);
    //dx/dt = mu x - F/V x
    xp[0]= kCoefs[0]*x[0] - u*(x[0]/ x[3]);  // X1 - X

    //dX2/dt = -10*g1*X1 + u*( (150-X2)/X4 )
    //dS/dt = - mu /Yxs x + F/V (Sf-S)
    xp[1]= -kCoefs[0]/ modelPars[0] * x[0] + u /x[3] * (modelPars[1]-x[1]); // X2 - S

    //dX3/dt = g2*X1 -u*(X3/X4)
    // dP/dt = pi x - F/V P
    xp[2]= kCoefs[1]*x[0] - u * ( x[2]/ x[3] ); // X3 - P

    //dX4/dt = u
    // dV/dt = F
    xp[3]= u; // X4 - V

    return(xp);
}
```

Figura 16: Equações diferenciais representando a fermentação em semi-descontinuo.

As equações cinéticas foram transcritas para linguagem Java conforme apresentado na Figura 17, e incluídas num método *kinetics* que era invocado pela função *updateKineticCoef(t)* sempre que necessário. No decorrer da simulação numérica, os valores das variáveis cinéticas eram calculados com base na variação das variáveis de estado *S* e *P* (presentes nas equações cinéticas). Esses valores eram então guardados num vector *kCoefs*, o qual era posteriormente utilizado nas equações diferenciais, como demonstrado na Figura 16.

```
public void kinetics (double S, double P)
{
    //kCoefs[0] = (0.408/(1.0 + (X3/16.0))) * (X2/(0.22+X2));

    kCoefs[0] = (modelPars[2]/(1.0 + (P/modelPars[3]))) * (S/(modelPars[4]+S));

    //kCoefs[1] = (1.0 / (1.0 + (X3/71.5))) * (X2/(0.44+X2));

    kCoefs[1] = (modelPars[5] / (1.0 + (P/modelPars[6]))) * (S/(modelPars[7]+S));
}
```

Figura 17: equações cinéticas referentes às variáveis g_1 e g_2

Após a definição das classes que caracterizavam o modelo em semi-descontinuo, estas foram compiladas e colocadas no pacote *bio* presente na biblioteca principal *bioferm*. A partir deste momento foi possível trabalhar com o modelo em questão e realizar as diversas operações presentes na aplicação *OptFerm*.

5.2. Clipboard do *OptFerm*

Inicialmente, é necessário criar um novo projecto (Figura 18), surgindo uma janela (Figura 19) com uma caixa de selecção em que o utilizador pode escolher o modelo com que pretende trabalhar, caso já estejam definidos modelos na livraria *bio*. Nesta janela, o utilizador pode dar um nome ao seu projecto e escolher se pretende guardar automaticamente todas as operações que vão sendo realizadas no decorrer do trabalho, bastando seleccionar a caixa "*automatic backup*".

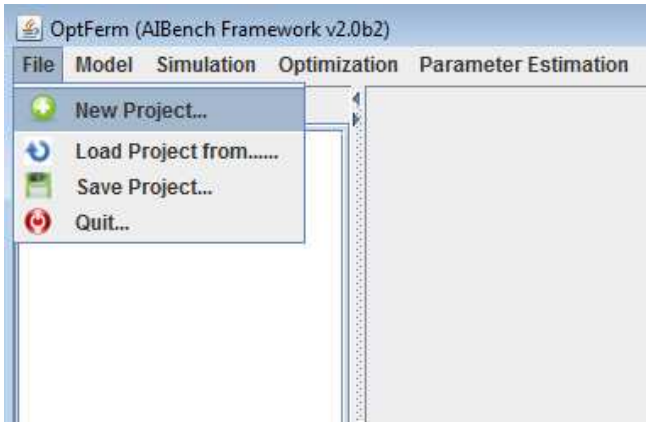


Figura 18: Menu onde se podem criar novos projectos, abrir projectos anteriormente gravados, gravar projectos ou sair da aplicação.

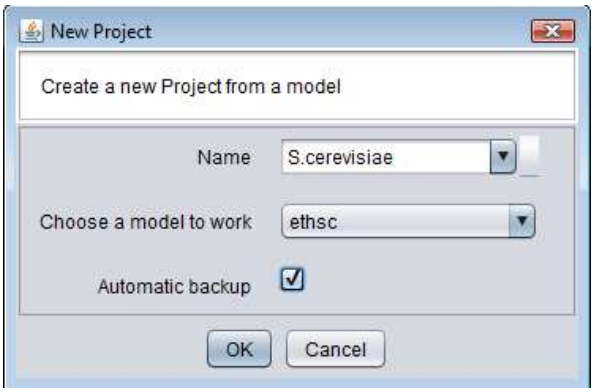


Figura 19: Janela que aparece ao utilizador quando é criado um novo projecto

Após criar um novo projecto são instanciados automaticamente os vários itens (*Model state variables*, *Model parameters*, *Simulation results*, *Optimization results* e *Parameter estimation Results*) ficando estes dispostos no *Clipboard* do *OptFerm* como apresentado na Figura 20. Podem ser criados vários projectos com o mesmo modelo mas não um projecto com vários modelos.

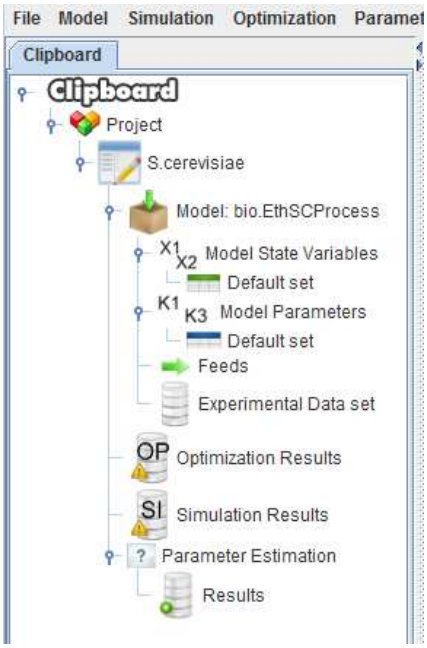


Figura 20: Disposição dos vários itens no Clipboard do OptFerm após criar ou abrir um projecto.

À medida que se vão realizando as diversas tarefas de optimização, simulação e estimação de parâmetros, os respectivos resultados serão adicionados ao *Clipboard*, ficando dispostos na devida categoria (simulação, optimização ou resultados de estimação) de acordo com a ordem de execução dessas mesmas tarefas. Podem ser criados novos conjuntos de valores iniciais das variáveis de estado, parâmetros, perfis de alimentação e dados experimentais sempre que necessário (Figura 21). Para aceder à informação contida em cada um dos objectos presentes no *clipboard*, basta

carregar sobre o próprio objecto, ficando a informação visível num painel ao lado (Figura 22). Esta informação pode ser posteriormente modificada ou guardada, dependendo do objecto em questão. No caso de serem conjuntos de dados das variáveis de estado, parâmetros, perfis de alimentação e dados experimentais, pode-se modificar, guardar e carregar dados anteriormente gravados sempre que necessário. Para objectos referentes a resultados (simulação, optimização e estimação), o utilizador apenas os pode visualizar ou então guardar num ficheiro em formato TXT, PDF ou JPEG.

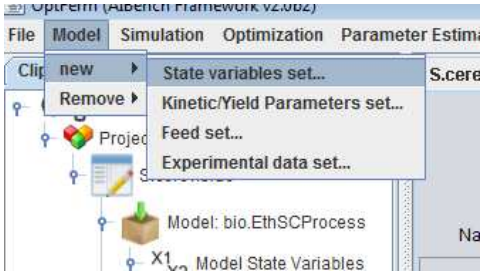


Figura 21: Menu para criar novos conjuntos de dados para os valores iniciais das variáveis de estado, parâmetros, perfis de alimentação e dados experimentais.

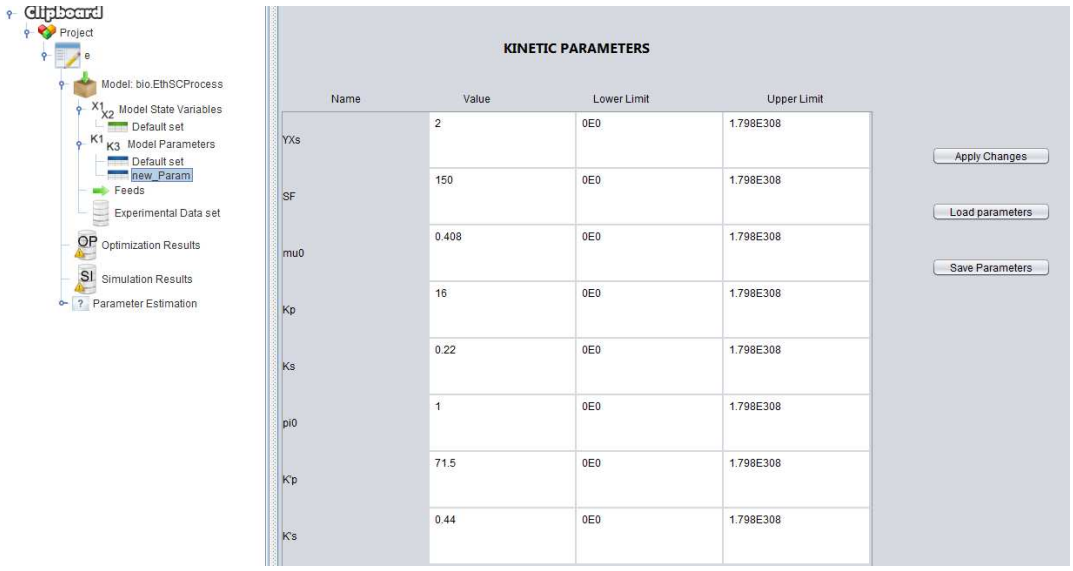


Figura 22: Exemplo de como são mostrados ao utilizador os dados referentes a um conjunto de parâmetros, após carregar sobre o respectivo objecto no clipboard.

5.3. Simulação

Para a realização das simulações, foi implementada uma interface gráfica igual à apresentada na Figura 23. Esta interface foi desenhada de modo a proporcionar ao

utilizador a capacidade de realizar facilmente várias simulações com diferentes combinações de valores iniciais das variáveis de estado, parâmetros e perfis de alimentação. À medida que as diversas tarefas de optimização do perfil de alimentação ou estimação de parâmetros vão sendo realizadas, os resultados ficam imediatamente disponíveis nas caixas de selecção (Figura 23). Deste modo, o utilizador pode utilizar esses mesmos resultados nas simulações e verificar de forma imediata se são os mais adequados. Os valores das variáveis de estado e dos parâmetros ficam dispostos no painel à medida que se seleccionem os diferentes conjuntos nas caixas de selecção. Os valores são mostrados ao utilizador de modo a ajudá-lo aquando da escolha das condições para realizar as simulações, podendo estes apenas ser visualizados mas não alterados. Se o utilizador pretender utilizar um perfil de alimentação previamente definido ou então resultante de uma optimização, basta seleccionar o campo "*with feed*" (Figura 23) e imediatamente ficam dispostos todos os conjuntos presentes no *Clipboard* nas caixas de selecção.

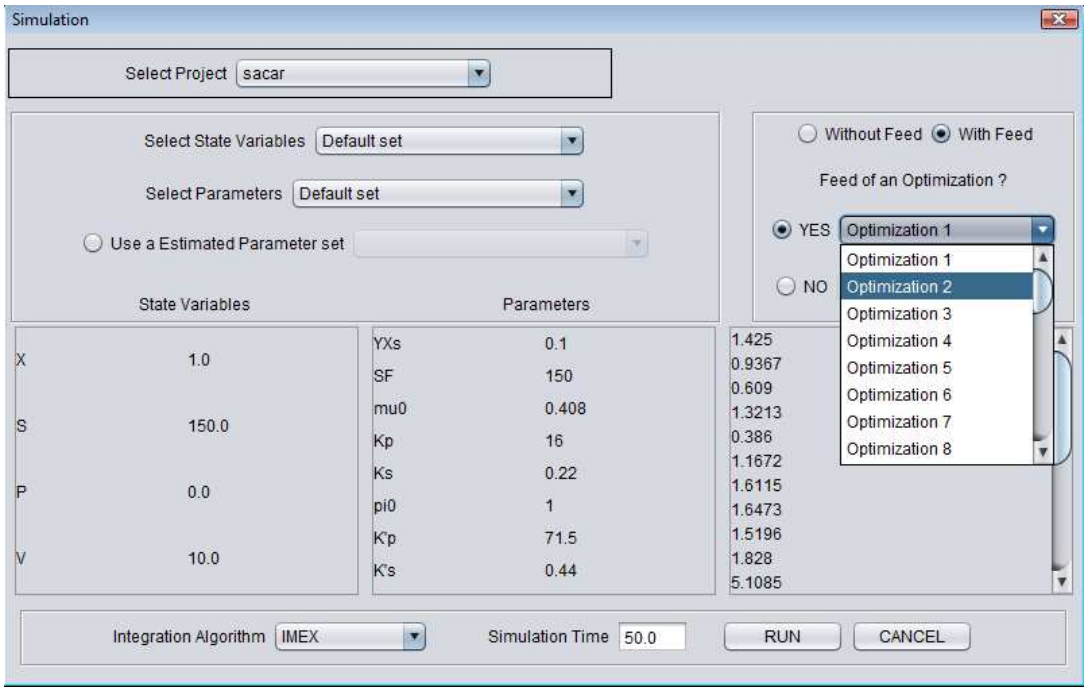


Figura 23: Interface Gráfica para executar as tarefas de simulação.

Após a selecção das condições a utilizar nas simulações pode-se escolher o algoritmo para efectuar a simulação numérica e o tempo de simulação pretendido. No caso de se utilizar um perfil de alimentação obtido numa optimização, o tempo de simulação será automaticamente preenchido, sendo este igual ao definido na tarefa de optimização (este valor pode ser alterado).

Assim que as simulações são executadas, os resultados podem ser visualizados através de um gráfico semelhante ao apresentado na Figura 24, no qual é facultada ao utilizador a visualização da variação das variáveis de estado ou a variação das cinéticas ao longo do tempo. Pode-se visualizar simplesmente uma

variável de estado ou cinética de cada vez ou todas em simultâneo (variáveis de estado ou cinéticas) conforme apresentado na Figura 24.

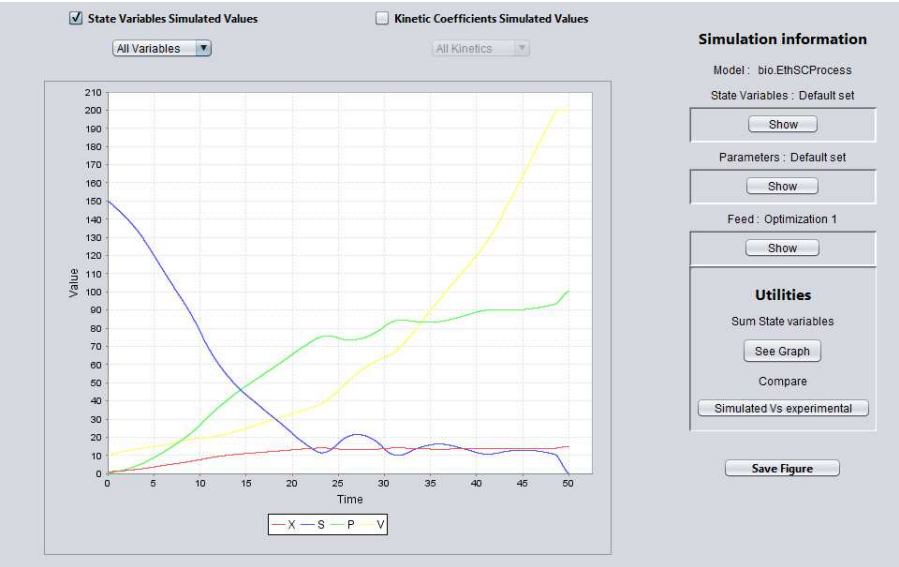


Figura 24: Painel onde são mostrados os resultados das simulações (através de um gráfico) e as condições utilizadas.

No lado direito do painel, o utilizador tem acesso às condições utilizadas em cada uma das simulações efectuadas, ou então pode aceder a duas funcionalidades, uma para comparar os resultados simulados com dados experimentais (Figura 25) ou a uma outra que serve para somar duas ou mais variáveis de estado (Figura 26). Pode-se comparar uma simulação com diferentes conjuntos de dados experimentais ou vice-versa. No entanto, só é possível comparar uma variável de estado de cada vez tal como apresentado na Figura 25. No final, o gráfico pode ser guardado num ficheiro em formato JPEG.

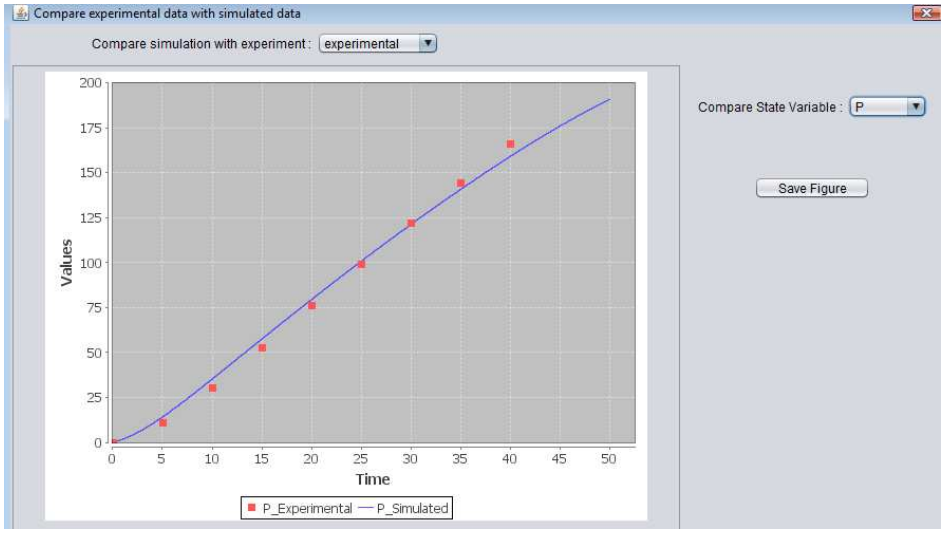


Figura 25: Interface gráfica mostrada ao utilizador quando é realizada a comparação dos dados simulados com os dados experimentais.

A funcionalidade que permite somar duas ou mais variáveis de estado torna-se útil para casos em que se realize a optimização de duas ou mais variáveis de estado, e no final se pretenda visualizar a soma das partes como o total de produto final.

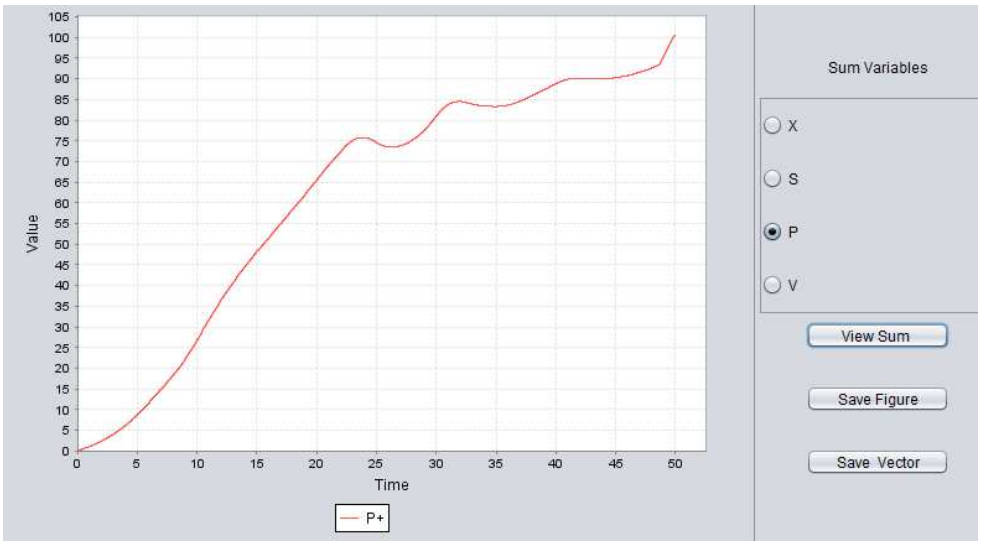


Figura 26: Interface gráfica mostrada ao utilizador quando é efectuada a soma de duas ou mais variáveis de estado.

5.4. Optimização

Para a realização das optimizações foi desenhada uma interface gráfica tal como apresentado na Figura 27. O utilizador pode aceder facilmente e de forma intuitiva a todas as opções. Tal como nas simulações, o utilizador pode escolher o projecto, o conjunto dos valores iniciais das variáveis de estado e os parâmetros com que pretende trabalhar, desde que estes tenham sido definidos previamente. Podem-se realizar três tipos diferentes de optimização como referido anteriormente em 3.4; porém, apenas se pode executar um tipo de cada vez.

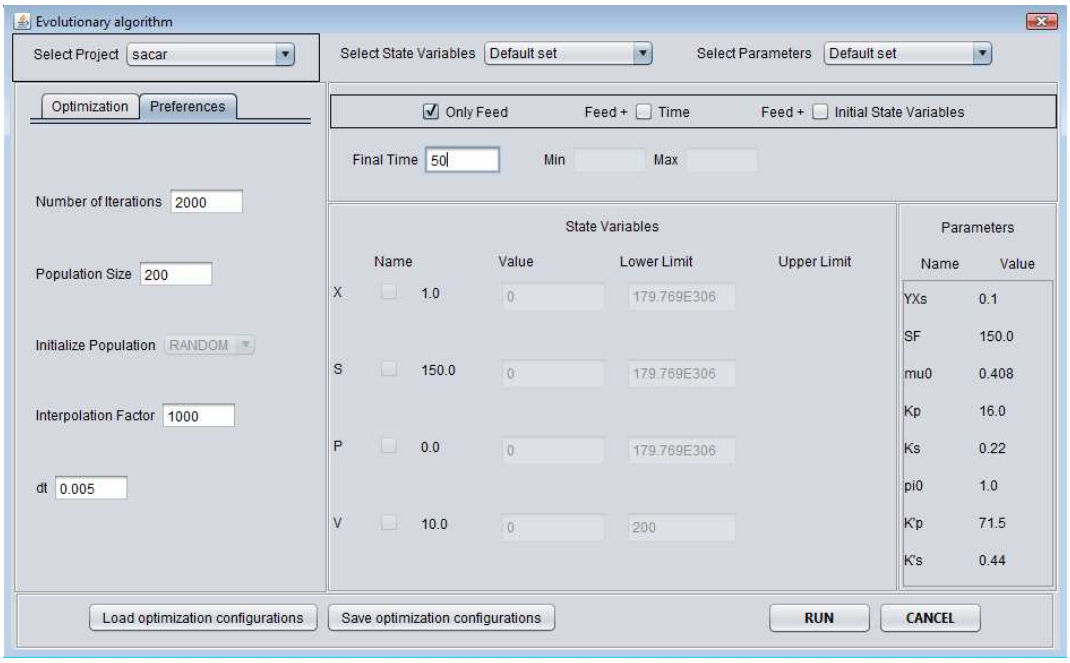


Figura 27: Interface gráfica apresentada ao utilizador quando são realizadas as tarefas de optimização.

Existe um painel "*preferences*" relacionado com as configurações do algoritmo de optimização: o número de iterações, o tamanho da população, o intervalo de discretização e um factor de interpolação utilizado para reduzir o tamanho da solução final, tal como explicado na secção 3.4. No painel "*Optimization*" o utilizador pode definir os limites mínimos e máximos da bomba de alimentação ao reactor, como explicado em 3.4. Após a execução da optimização, os resultados obtidos podem ser visualizados num painel semelhante ao apresentado na Figura 28, onde são apresentados simultaneamente um gráfico e uma tabela. Sempre que se realizem as optimizações para determinar os valores iniciais mais adequados para as variáveis de estado, estes serão também apresentadas no painel.

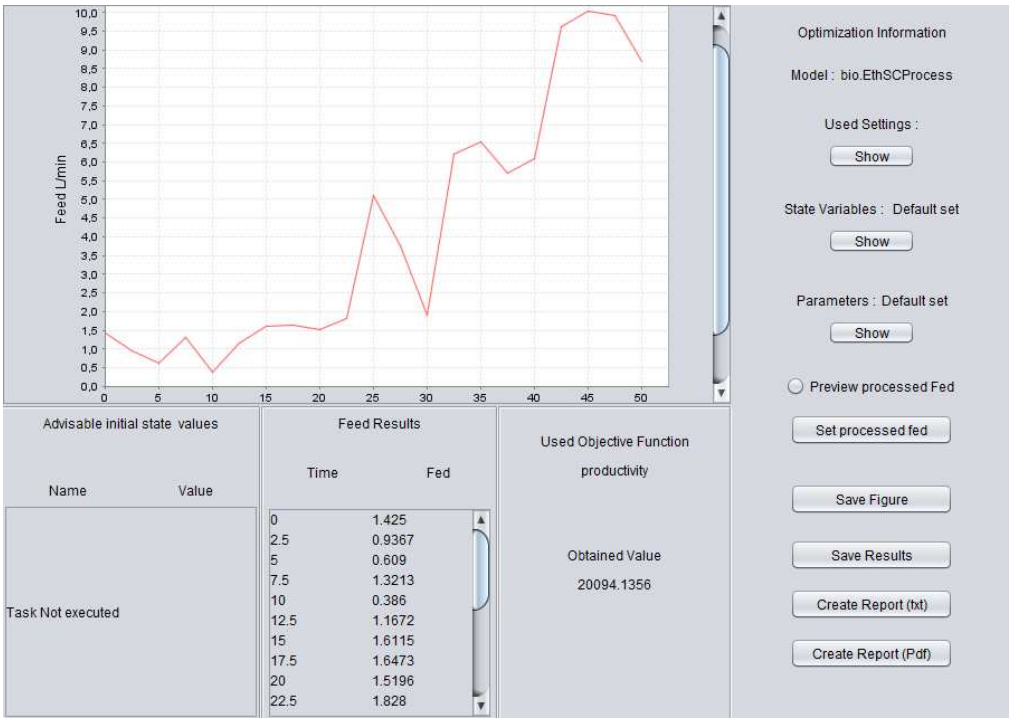


Figura 28: Painel apresentado ao utilizador após a execução das tarefas de optimização.

No lado direito do painel, são expostas as condições utilizadas na optimização, que podem ser consultadas sempre que necessário. Os resultados poderão ser guardados num ficheiro JPEG (gráfico) ou então num ficheiro TXT (tabelas). Existe ainda a possibilidade de gerar um relatório em formato PDF ou TXT com os resultados e as condições utilizadas nas optimizações.

5.5. Estimação de parâmetros

Para a realização das tarefas de estimação de parâmetros, foi desenhada a interface gráfica apresentada na Figura 29. As várias opções foram dispostas ordenadamente de modo a facilitar a vida do utilizador nestas tarefas. Tal como na simulação e na optimização, os diferentes conjuntos de dados correspondentes aos valores iniciais das variáveis de estado, parâmetros e dados experimentais são dispostos em caixas de selecção. Porém, todos os conjuntos de dados a utilizar deverão ser definidos previamente, pois de outra forma não serão apresentados nas caixas de selecção. O utilizador poderá estimar parâmetros recorrendo a dados experimentais provenientes de fermentações em semi-continuo. Contudo, tem que utilizar o mesmo perfil de alimentação usado nessas experiências reais de modo a haver a concordância de dados no processo de determinação de parâmetros. Por pré-definição, a utilização de um perfil de alimentação não está considerada, sendo necessário a sua activação por parte do utilizador (seleccionando "*yes"* em "*use feed*"); todavia, ter-se-á que definir anteriormente um perfil de alimentação.

Em cada estimação, o utilizador pode seleccionar os parâmetros que pretende fixar, ficando estes com o valor constante no decorrer das operações de estimação. Existe também a possibilidade de o utilizador seleccionar variáveis de estado de modo a que estas não sejam englobadas nos cálculos de estimação dos parâmetros. Sempre que uma variável de estado é seleccionada, esta é desprezada dos cálculos da função de custo. Esta funcionalidade torna-se importante por exemplo no caso de uma variável de estado apresentar um valor nulo ao longo do tempo, situação em que a função de custo tende sempre para infinito, impossibilitando a sua convergência para um valor mínimo. Consequentemente, os valores dos parâmetros não serão determinados correctamente.

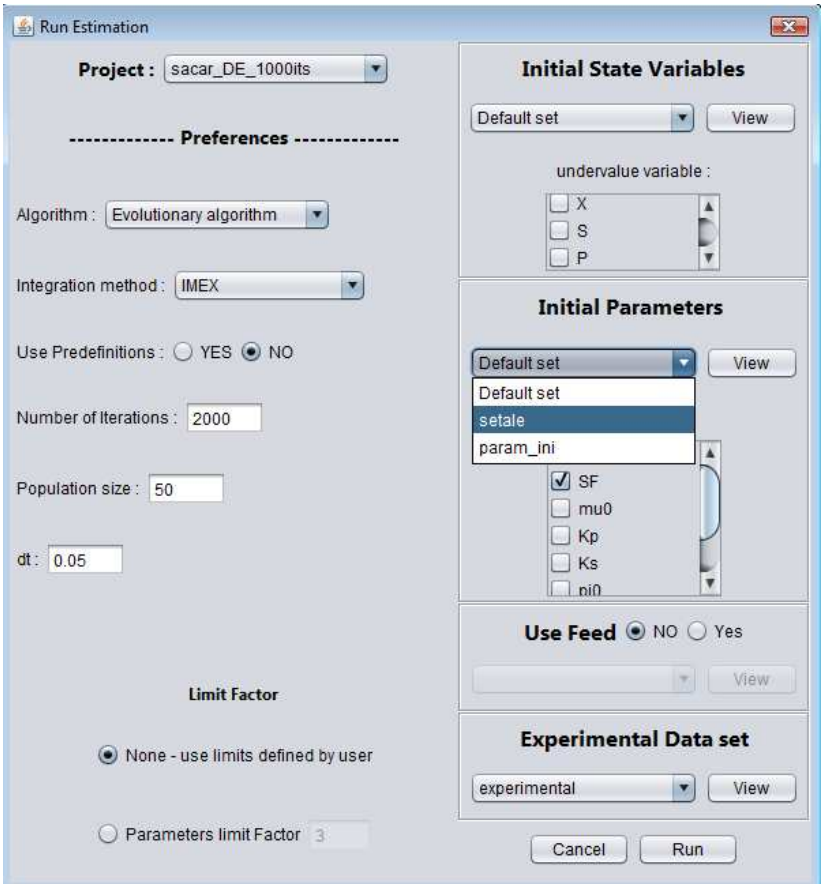


Figura 29: Interface Gráfica para a execução da estimação de parâmetros.

No menu "*preferences*" (Figura 29) o utilizador pode seleccionar o algoritmo de optimização, o método de integração e escolher as configurações a utilizar no algoritmo de optimização (número de iterações e o tamanho da população). Existe um outro parâmetro designado de "*Limit Factor*" que consiste num valor que é multiplicado e dividido aos valores iniciais dos parâmetros com a finalidade de criar automaticamente os limites superiores e inferiores dos parâmetros a utilizar nas operações de estimação.

No final, os parâmetros estimados e os limites utilizados são apresentados numa tabela conforme apresentado na Figura 30. O utilizador pode ainda criar um

relatório com toda a informação relacionada com a estimação de parâmetros, no qual são descritas as condições utilizadas e os resultados obtidos.

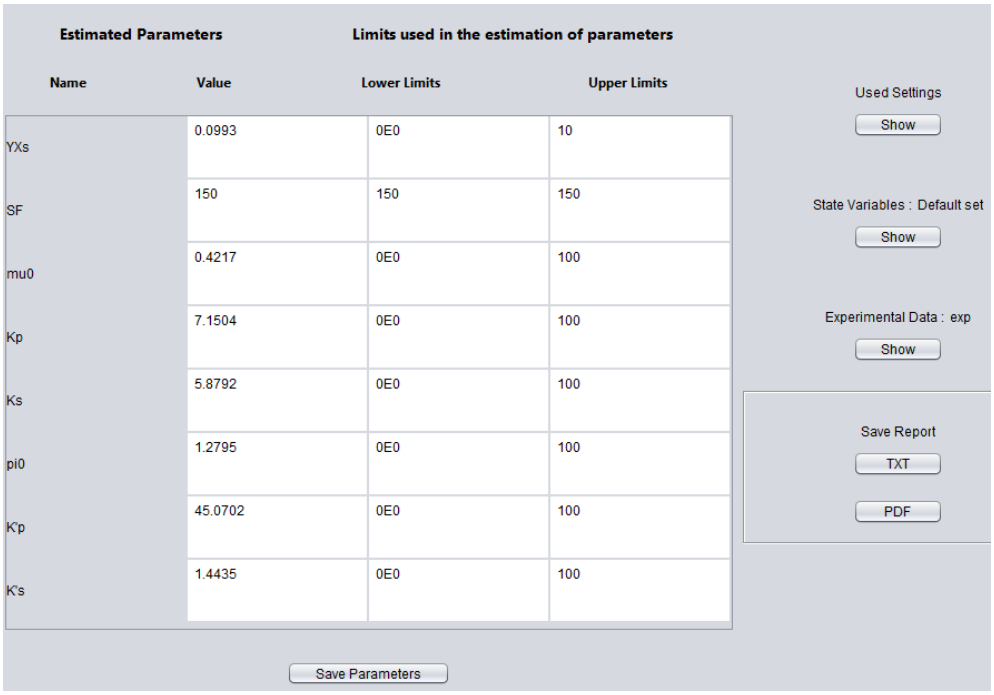


Figura 30: Painel apresentado ao utilizador após a realização das tarefas de estimação de parâmetros, onde são apresentados os novos parâmetros determinados e os limites utilizados nas operações de estimação.

5.6. Estudo da execução das operações

Para verificar a robustez da aplicação realizou-se um estudo baseado no método de Monte Carlo (repetição de experiências), no qual foram realizadas várias optimizações e estimações de parâmetros utilizando as mesmas condições. A seguir, calcularam-se os intervalos de confiança para os valores da função objectivo obtidos nas várias optimizações do perfil de alimentação e para os valores da função objectivo obtidos nas tarefas de estimação de parâmetros. Caso houvesse algum erro na concepção da aplicação *OptFerm*, os valores da função objectivo obtidos poderiam ser bastante diferentes em experiências onde se utilizaram as mesmas condições, causando grandes intervalos de confiança. Os intervalos de confiança foram calculados de acordo com a seguinte equação:

$$\left[\overline{X} - t.\frac{\sigma}{\sqrt{n}}; \overline{X} + t.\frac{\sigma}{\sqrt{n}}\right] \qquad (20)$$

onde $\bar{X}$ é a média dos valores da função objectivo obtidos nas diferentes optimizações ou estimações, *t* representa o valor de uma distribuição de t-student com uma confiança de 95% referente a (*n-1*) graus de liberdade, *n* é o número de experiências (optimizações ou estimações) e *σ* é o desvio padrão.

As tarefas de optimização e estimação de parâmetros foram repetidas 30 vezes, em que para cada conjunto se utilizou um algoritmo de optimização diferente, de entre os existentes na aplicação. Nas várias tarefas, utilizou-se o modelo supracitado, seguindo a metodologia exemplificada neste capítulo, em optimização, simulação e estimação de parâmetros.

Ao realizar as optimizações do perfil de alimentação foram definidas as mesmas condições iniciais para todas elas, que foram:

- Os valores iniciais das variáveis de estado e parâmetros cinéticos e de rendimento foram os mesmos definidos inicialmente no modelo (valores por defeito).
- O tempo total de optimização (duração da fermentação) foi de 50 horas.
- O factor de interpolação foi definido como 500.

No entanto, o número de iterações utilizado no algoritmo *Simulated Annealing* foi diferente dos restantes, pois em cada iteração do SA são testados menos soluções. Assim, foi tido em conta o equilíbrio do esforço computacional. Na Tabela 6 são apresentados os intervalos de confiança obtidos após a realização das várias optimizações com os diferentes algoritmos de optimização. Verificou-se que os intervalos de confiança são bastante pequenos, indicando que os valores da função objectivo tendem a convergir para um mesmo valor, quando são utilizadas as mesmas condições. Deste modo, pode-se concluir que a aplicação OptFerm é capaz de executar adequadamente as operações de optimização do perfil de alimentação.

Tabela 6: Intervalos de confiança para o valor da função objectivo (relativa à optimização do perfirl de alimentação) após a realização de 30 optimizações com cada um dos diferentes algoritmos.

	Simulated annealing	**Evolução diferencial**	**Algoritmo genético**
Nº iterações	5000	1000	1000
Função objectivo I.C	20087.031 ± 16.198	20114.832 ± 0.609	20103.202 ± 4.445

Como complemento ao estudo anterior foi realizado um outro estudo do qual se procedeu da seguinte forma:

1. Efectuaram-se as simulações, utilizando em cada uma delas um perfil de alimentação obtido nas optimizações. Procedeu-se de igual forma ao explicado no passo 5.3.
2. Calcularam-se os intervalos de confiança de acordo com a equação 20 para cada uma das variáveis de estado presentes no modelo, com uma ligeira alteração. O $\bar{X}$ representava a média do valor de cada uma das variáveis de estado no tempo final de simulação (50 horas).

Como se pode verificar na Tabela 7,os intervalos de confiança para cada uma das variáveis de estado no tempo final são muito pequenos, mostrando que os valores finais das variáveis de estado vão ser idênticos quando utilizado um perfil de alimentação obtido numa optimização onde se utilizaram as mesmas condições

iniciais, reforçando assim a conclusão de que as optimizações estão a ser adequadamente realizadas.

Tabela 7: Intervalos de confiança para cada uma das variáveis de estado, referentes ao valor obtido no tempo final da simulação (50 horas), após realizar a simulação com o perfil de alimentação obtido nas optimizações. As unidades das variáveis de estado são em (*g/L*) com a excepção do volume (*v*) que é em litros *L*.

Variáveis de estado	Simulated annealing	Evolução Diferencial	Algoritmo genético
X	15.040 ± 0.0	15.042 ± 0.0	15.041 ± 0.0
S	0.0958 ± 0.0001	0.0705 ± 0.0003	0.0807 ± 0.0001
P	100.396 ± 0.0004	100.550 ± 0.0	100.483 ± 0.0001
V	200.265 ± 0.002	200.227 ± 0.0012	200.253 ± 0.001

Ao realizar os estudos para a estimação de parâmetros, foi necessário criar um conjunto hipotético de dados experimentais, pois não foram encontrados dados experimentais na bibliografia. Assim, procedeu-se da seguinte forma:

a) Realizou-se uma simulação utilizando o conjunto de valores iniciais das variáveis de estado e parâmetros definidos inicialmente no modelo.
b) A cada 5 horas de simulação foi registado o valor para cada uma das variáveis de estado, como se tratasse de uma experiência real.

Foi então realizada a estimação dos parâmetros, procedendo-se da seguinte forma:

1. Criou-se um conjunto de parâmetros com valores iniciais aleatórios.
2. O valor inicial das variáveis de estado foi igual às definidas inicialmente no modelo.
3. Utilizou-se o conjunto de dados experimentais criado como explicado nos passos a) e b).
4. Não foi utilizado nenhum perfil de alimentação.
5. Realizaram-se 30 estimações com cada um dos algoritmos de optimização referidos na Tabela 8, utilizando o número de iterações aí referido.

Após executar as várias tarefas de estimação de parâmetros, foram calculados os intervalos de confiança para o valor da função objectivo obtido no final de cada estimação. Como se pode verificar na Tabela 8, os intervalos de confiança são muito pequenos o que mostra que para as diferentes estimações, utilizando as mesmas condições, o valor da função objectivo converge para um valor comum. Estes dados indiciam que as tarefas de estimação parecem também estar a ser adequadamente realizadas, tal como foi observado nas tarefas de optimização.

Tabela 8: Intervalos de confiança para o valor da função objectivo após repetir 30 vezes a estimação de parâmetros com cada um dos diferentes algoritmos de optimização

	Simulated annealing	Evolução diferencial	Algoritmo genético
Nº iterações	20000	1000	1000
I.C Função objectivo	0.0191 ± 1.5E-6	0.0013 ± 2.1E-7	0.0271 ± 2.6E-6

È importante salientar que as tarefas de optimização e estimação de parâmetros são fortemente dependentes dos algoritmos de optimização e do número de iterações utilizadas. Como se pode constatar tanto na Tabela 6 como na Tabela 8, o algoritmo de evolução diferencial apresentou melhores resultados em relação aos restantes. Porém, noutros casos em que se utilize um modelo diferente, isto poderá não acontecer. Outro aspecto importante é o numero de iterações utilizado em cada procedimento, pois quanto maior o número de iterações maior a probabilidade de encontrar o mínimo ou máximo da função objectivo, mas maior o tempo de computação.

Capítulo 6

Conclusões e Trabalho Futuro

Neste capítulo, faz-se a exposição das conclusões mais importantes retiradas durante a realização deste trabalho. São ainda descritas as alterações a serem realizadas à aplicação *OptFerm* no futuro. O capítulo está dividido do seguinte modo:

6.1. Conclusões

Os objectivos principais desta tese, o desenvolvimento de uma ferramenta direccionada para processos fermentativos que permitisse realizar tarefas de simulação, optimização do perfil de alimentação e estimação de parâmetros, foram atingidos. Esta aplicação foi desenvolvida com o propósito de estudar os processos e as condições externas aplicadas, de modo a reduzir a quantidade de experiências (por tentativa-erro) actualmente necessárias para a optimização de determinado processo fermentativo. A validade do trabalho foi também comprovada com a aceitação de um artigo relacionado com este trabalho numa conferência internacional de qualidade (a *European Simulation and Modelling* conference – ESM 2009, Leicester, Reino Unido, Outubro de 2009).

A aplicação está apta a que qualquer utilizador possa analisar a robustez de qualquer modelo representativo de uma fermentação num sistema em semi-continuo ou descontínuo, permitindo ainda a comparação de dados simulados com dados experimentais. O utilizador fica também habilitado a determinar parâmetros desconhecidos de qualquer modelo através de técnicas de aproximação dos dados simulados aos dados experimentais, com a vantagem de se poder utilizar dados experimentais provenientes de experiências realizadas em semi-contínuo. Podem ser comparadas e analisadas diferentes condições de fermentação, como perfis de alimentação, que podem ainda ser optimizados, constituindo assim uma mais-valia para qualquer engenheiro de processos fermentativos.

A vantagem de utilizar esta aplicação na análise de processos fermentativos, em relação às existentes é que esta foi especificamente desenvolvida para trabalhar com esses processos, contendo todas a funcionalidades necessárias para simular e optimizar fermentações. No caso das outras ferramentas informáticas supracitadas, ter-se-iam que efectuar modificações estruturais (adicionar rotinas, novos algoritmos) para prover essas aplicações com as mesmas funcionalidades específicas para estudar processos fermentativos.

Nos estudos realizados para analisar a robustez da aplicação, verificou-se que os intervalos de confiança eram muito pequenos. Este facto indica que as rotinas estavam a ser executadas de forma estável e aparentemente com a ausência de erros. Porém, é de sublinhar que os resultados finais no que se refere à obtenção do melhor perfil de alimentação ou a determinação correcta dos parâmetros na estimação depende de vários factores, incluindo o algoritmo de optimização utilizado, o factor de interpolação, o número de iterações e o próprio modelo biológico. Assim, fica à consideração do utilizador determinar as melhores condições a utilizar.

A versão actual da aplicação OptFerm não permite construir, editar ou visualizar os modelos directamente, com o auxílio de uma interface gráfica desenhada especificamente para esse efeito. Esta limitação torna-se inconveniente para o utilizador comum, pois é mais difícil criar qualquer modelo para utilizar na aplicação, sendo necessário ter alguns conhecimentos da linguagem Java. Em contrapartida, o modo como os modelos são definidos actualmente na aplicação, possibilita uma

grande flexibilidade na sua construção, pois podem ser facilmente impostas várias restrições nas equações cinéticas através de estruturas de controlo da linguagem Java.

6.2. Trabalho futuro

Em trabalhos futuros, a prioridade é criar uma interface gráfica e as respectivas funções que permitam construir, editar e visualizar qualquer tipo de modelo. Ao implementar esta nova funcionalidade, pretende-se dar ao utilizador a mesma flexibilidade que se verifica actualmente, em termos de impor restrições a determinado processo ou reacção. Vão ser ainda adicionadas todas as funções que permitam exportar e importar os modelos em formato SBML.

Devido a esta ferramenta ter sido concebida sobre a aplicação *AIBench* e esta ser baseada num conceito de *plugins*, podem ser facilmente e rapidamente integradas novas funcionalidades no futuro. Existe ainda a vantagem de não ser necessário realizar qualquer modificação estrutural (modificar código antigo para que possa ser compatível com novos módulos) da aplicação *OptFerm* para que novos módulos possam ser integrados na aplicação.

Bibliografia

[1] C. Juma e V. Konde, *The new bioeconomy*, United Nations Conference on Trade and Development, 2001.

[2] B.B.V. Beuzekom e A. Arundel, "OECD Biotechnology Statistics 2009," *Statistics*, 2009.

[3] "Global biotechnology market expected to post ten-fold growth," *Focus on Catalysts*, vol. 2003, Abr. 2003, p. 2.

[4] M. Kawohl, T. Heine, e R. King, "Model based estimation and optimal control of fed-batch fermentation processes for the production of antibiotics," *Chemical Engineering and Processing: Process Intensification*, vol. 46, 2007, pp. 1223–1241.

[5] A. Kremling e J. Saez-Rodriguez, "Systems biology—An engineering perspective," *Journal of Biotechnology*, vol. 129, Abr. 2007, pp. 329–351.

[6] S. Hoops, S. Sahle, R. Gauges, C. Lee, J. Pahle, N. Simus, M. Singhal, L. Xu, P. Mendes, e U. Kummer, "COPASI--a COmplex PAthway SImulator," *Bioinformatics*, vol. 22, Dez. 2006, pp. 3067-3074.

[7] A. Funahashi, M. Morohashi, H. Kitano, e N. Tanimura, "CellDesigner: a process diagram editor for gene-regulatory and biochemical networks," *BIOSILICO*, vol. 1, 2003, p. 3.

[8] F.T. Bergmann, R.R. Vallabhajosyula, e H.M. Sauro, "Computational Tools for Modeling Protein Networks," *Current Proteomics*, vol. 3, Out. 2006, pp. 181-197.

[9] H. Schmidt e M. Jirstrand, "Systems Biology Toolbox for MATLAB: a computational platform for research in Systems Biology," *Bioinformatics*, Nov. 2005, p. bti799.

[10] M. Rocha, J. Pinto, I. Rocha, e E. Ferreira, "Evaluating Evolutionary Algorithms and Differential Evolution for the Online Optimization of Fermentation Processes," *Evolutionary Computation,Machine Learning and Data Mining in Bioinformatics*, 2007, pp. 236-246.

[11] M. Rocha, C. Vilela, e J. Neves, "A Study of Order Based Genetic and Evolutionary Algorithms in Combinatorial Optimization Problems," *Intelligent Problem Solving. Methodologies and Approaches*, 2000, pp. 1-9.

[12] M. Rocha, J. Neves, I. Rocha, e E.C. Ferreira, "Evolutionary algorithms for optimal control in fed-batch fermentation processes."

[13] M. Rocha, R. Mendes, P. Maia, J. Pinto, I. Rocha, e E. Ferreira, "Evaluating Simulated Annealing Algorithms in the Optimization of Bacterial Strains," *Progress in Artificial Intelligence*, 2007, pp. 473-484.

[14] J. Nielsen, J. Villadsen, e G. Lidén, *Bioreaction Engineering Principles*, Springer, 2002.

[15] L.Z. Chen, S.K. Nguang, e X.D. Chen, *Modelling and Optimization of Biotechnological Processes: : Artificial Intelligence Approaches*, Springer Verlag, 2006.

[16] M. Doble, A. Kruthiventi, e V. Gaikar, *Biotransformations and Bioprocesses*, Marcel Dekker, 2004.

[17] E. Heinzle, A.P. Biwer, e C.L. Cooney, *Development of Sustainable Bioprocesses*, Chichester, UK: John Wiley & Sons, Ltd, 2006.

[18] P.F. Stanbury, S. Hall, e A. Whitaker, *Principles of Fermentation Technology, Second Edition*, Butterworth-Heinemann, 1999.

[19] F. Swetz e J.S. Hartzler, *Mathematical modeling in the secondary school curriculum*, (Reston, Va), 1991.

[20] O. Demin e I. Goryanin, *Kinetic Modelling in Systems Biology*, Chapman & Hall/CRC, 2008.
[21] M. Thilakavathi, T. Basak, e T. Panda, "Modeling of enzyme production kinetics," *Applied Microbiology and Biotechnology*, vol. 73, Jan. 2007, pp. 991-1007.
[22] K. Schügerl, *Biotechnology, 2E, Vol. 4, Measuring, Modelling, and Control*, Wiley-VCH, 1996.
[23] M. Hofman e P. Thonart, *Engineering and Manufacturing for Biotechnology (Focus on Biotechnology)*, Springer, 2007.
[24] S. Tang, J. Chen, e Z. Zhang, "Structured models for recombinant human interleukin-11 fermentation," *Biochemical Engineering Journal*, vol. 35, Jul. 2007, pp. 210-217.
[25] L. Cazzador, L. Mariani, E. Martegani, e L. Alberghina, "Structured segregated models and analysis of self-oscillating yeast continuous cultures," *Bioprocess and Biosystems Engineering*, vol. 5, Jul. 1990, pp. 175-180.
[26] A.K. Gombert e J. Nielsen, "Mathematical modelling of metabolism," *Current Opinion in Biotechnology*, vol. 11, Abr. 2000, pp. 180-186.
[27] J.E. Bailey, "Mathematical Modeling and Analysis in Biochemical Engineering: Past Accomplishments and Future Opportunities," *Biotechnology Progress*, vol. 14, 1998, pp. 8-20.
[28] Mitchell D.A., von Meien O.F., Krieger N., e Dalsenter F.D.H., "A review of recent developments in modeling of microbial growth kinetics and intraparticle phenomena in solid-state fermentation," *Biochemical Engineering Journal*, vol. 17, Jan. 2004, pp. 15-26.
[29] G. Bastin, *On-line estimation and adaptive control of bioreactors*, (Amsterdam, New York): 1990.
[30] A. Liese, K. Seelbach, e C. Wandrey, *Industrial Biotransformations*, Weinheim, FRG: Wiley-VCH Verlag GmbH & Co. KGaA, 2006.
[31] T. Zhang e M. Guay, "Adaptive nonlinear observers of microbial growth processes," *Journal of Process Control*, vol. 12, Ago. 2002, pp. 633-643.
[32] H. Beyenal, S.N. Chen, Z. Lew, e owski, "The double substrate growth kinetics of Pseudomonas aeruginosa," *Enzyme and Microbial Technology*, vol. 32, Jan. 2003, pp. 92-98.
[33] L.T. Biegler e I.E. Grossmann, "Retrospective on optimization," *Computers & Chemical Engineering*, vol. 28, Jul. 2004, pp. 1169-1192.
[34] E.K.P. Chong e S.H. Zak, *An Introduction to Optimization, 2nd Edition*, Wiley-Interscience, 2001.
[35] T.F. Edgar e D.M. Himmelblau, *Optimization of Chemical Processes*, McGraw-Hill Science/Engineering/Math, 2001.
[36] W. Sun e Y. Yuan, *Optimization Theory and Methods: Nonlinear Programming*, Springer, 2006.
[37] J.R. Banga, E. Balsa-Canto, C.G. Moles, e A.A. Alonso, "Dynamic optimization of bioprocesses: Efficient and robust numerical strategies," *Journal of Biotechnology*, vol. 117, Jun. 2005, pp. 407-419.
[38] R. Mendes, I. Rocha, E. Ferreira, e M. Rocha, "A Comparison of Algorithms for the Optimization of Fermentation Processes," *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, 2006, pp. 2018-2025.
[39] C.R. Reeves e J.E. Rowe, *Genetic Algorithms - Principles and Perspectives: A Guide to GA Theory*, Springer, 2002.
[40] M. Mitchell, *An Introduction to Genetic Algorithms*, The MIT Press, 1998.
[41] K.V. Price, R.M. Storn, e J.A. Lampinen, *Differential Evolution: A Practical Approach to Global Optimization*, Springer, 2005.
[42] R. Schwartz, *Biological Modeling and Simulation: A Survey of Practical Models, Algorithms, and Numerical Methods*, The MIT Press, 2008.

[43] J.H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*, The MIT Press, 1992.
[44] C.W. Ahn, *Advances in Evolutionary Algorithms: Theory, Design and Practice*, Springer, 2006.
[45] S. Sumathi, T. Hamsapriya, e P. Surekha, *Evolutionary Intelligence: An Introduction to Theory and Applications with Matlab*, Springer, 2008.
[46] P. Cortez, "Algoritmos genéticos e redes neuronais na previsão de séries temporais," masterThesis, Universidade do Minho, 1997.
[47] R. Mendes, I. Rocha, J. Pinto, E. Ferreira, e M. Rocha, "Differential Evolution for the Offline and Online Optimization of Fed-Batch Fermentation Processes," *Advances in Differential Evolution*, 2008, pp. 299-317.
[48] R. Storn, "On the usage of differential evolution for function optimization," *Proceedings of North American Fuzzy Information Processing*, Berkeley, CA, USA: 1996, pp. 519-523.
[49] E. Simoncelli, "Least Squares Optimization," Fev. 1999.
[50] A. Pettinen, T. Aho, O. Smolander, T. Manninen, A. Saarinen, K. Taattola, O. Yli-Harja, e M. Linne, "Simulation tools for biochemical networks: evaluation of performance and usability," *Bioinformatics*, vol. 21, Fev. 2005, pp. 357-363.
[51] F.T. Bergmann e H.M. Sauro, "SBW - a modular framework for systems biology," *Proceedings of the 38th conference on Winter simulation*, Monterey, California: Winter Simulation Conference, 2006, pp. 1637-1645.
[52] L.M. Loew e J.C. Schaff, "The Virtual Cell: a software environment for computational cell biology," *Trends in Biotechnology*, vol. 19, Out. 2001, pp. 401-406.
[53] "http://www.sbtoolbox2.org/main.php," *System Biology Toolbox 2*.
[54] R. Alves, F. Antunes, e A. Salvador, "Tools for kinetic modeling of biochemical networks," Jun. 2006.
[55] U.M. Ascher, S.J. Ruuth, e R.J. Spiteri, "Implicit-explicit Runge-Kutta methods for time-dependent partial differential equations," *Appl. Numer. Math.*, vol. 25, 1997, pp. 151–167.
[56] "http://platonos.sourceforge.net/index.php?p=PluginEngine," *Platonos*.
[57] M. Rocha, P. Maia, R. Mendes, J. Pinto, E. Ferreira, J. Nielsen, K. Patil, e I. Rocha, "Natural computation meta-heuristics for the in silico optimization of microbial strains," *BMC Bioinformatics*, vol. 9, 2008, p. 499.
[58] C. Chen e C. Hwang, "Optimal Control Computation for Differential-algebraic Process Systems with General Constraints," *Chemical Engineering Communications*, vol. 97, 1990, p. 9.