DAS Departamento de Automação e Sistemas
CTC Centro Tecnológico
UFSC Universidade Federal de Santa Catarina

Otimização Sem Derivadas para Sintonia Automática de um Simulador de Poços de Petróleo

Relatório submetido à Universidade Federal de Santa Catarina
como requisito para a aprovação da disciplina:
DAS 5501: Estágio em Controle e Automação

Willian de Medeiros Galvani

Florianópolis, Junho de 2017

Otimização Sem Derivadas para Sintonia Automática de um Simulador de Poços de Petróleo

Willian de Medeiros Galvani

Este relatório foi julgado no contexto da disciplina

DAS 5501: Estágio em Controle e Automação

e aprovada na sua forma final pelo

Curso de Engenharia de Controle e Automação

Prof. Eduardo Camponogara

Orientador

Fly, you fools!

J. R. R. Tolkien

Resumo

Este trabalho propõe o uso de métodos de otimização sem derivadas na indústria de petróleo e gás, para sintonia de simuladores de poços de petróleo a partir de dados de poços reais. São analisados dois algoritmos, Nelder-Mead (Simplex) e OrthoMADS. O simulador utilizado é o Pipesim, e para interface com ele são utilizados Python, Opal e pyWin32. Os métodos são testados de forma a minimizar a distância quadrática entre a curva de produção de um poço real e a do poço simulado. São testadas também variações dos algoritmos. O sistema proposto foi capaz de sintonizar os modelos desejados em tempo aceitável e com boa precisão.

Palavras-chave: otimização sem derivada, poços de petróleo, simulação, sintonia automática .

Abstract

This work proposes the use of derivative-free optimization methods in the oil and gas industry, with the purpose of tuning an oil well simulators to match real oil wells. Two algorithms are analized, Nelder-Mead (simplex) and OrthoMADS. The simulator used is Pipesim. Python, pyWin32 and Opal were used to interface with it. The methods are tested by attempting to reduce the quadratic distance between the real well's production curve and the simulator's. Some variations of the algorithms are also tested. The proposed system was able to tune the simulator to a desired accuracy within an acceptable computing time.

Keywords: derivative-free optimization, oil well, simulation, automatic tuning.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

1.1 Motivação

Na indústria de petróleo e gás, frequentemente duas ou mais plataformas de produção conectam-se à redes comuns para o escoamento da produção. Este escoamento precisa ser planejado para que as restrições físicas de fluxo e pressão sejam respeitadas. Para que estes valores estejam corretos, é necessário controlar os processos de modo a produzir exatamente o necessário em cada plataforma. Além disso, para que campos de petróleo gerem o maior retorno financeiro possível, os processos devem idealmente estar sempre em um ponto ótimo de performance. A produção em falta ou excesso de determinados componentes pode afetar os elementos do sistema, como bombas e separadores, ou as interfaces com sistemas auxiliares, como linhas de transporte. Outros problemas ocasionados pelo excesso de produção são o estresse dos sistemas da plataforma e perdas desnecessárias com desgaste ou subprodução.

Como estes são sistemas complexos e que variam com o tempo, simuladores são utilizados para estimar as respostas do sistema real à diversas condições. Para reproduzir as medidas observadas apropriadamente, estes simuladores devem ser rotineiramente resintonizados pelo engenheiro de processos. Esta sintonia pode ser uma tarefa cansativa e duradoura.

Um simulador bem sintonizado pode ser utilizado para modelagem matemática de novos modelos, desenvolvimento e testes de algoritmos de controle, planejamento de operações e análise de riscos, entre outros.

Como a escala de produção destes sistemas costuma ser grande, até pequenas variações dos modelos reais podem causar impactos consideráveis na produção.

1.2 Objetivos

O objetivo final deste trabalho é o desenvolvimento de uma ferramenta capaz de auxiliar a sintonia de simuladores de fluxo multifásico de petróleo e gás utilizando dados de plantas reais em plataformas em alto mar. Pretende-se identificar, implementar e testar métodos de otimização sem derivadas, os quais sejam compatíveis para sintonizar tais simuladores. É esperado que a ferramenta seja capaz de aproximar em pouco tempo as curvas de produção de um simulador aos dados coletados em campo. Inicialmente os métodos a serem estudados são o Simplex de Nelder-Mead [1], com uma implementação própria, o OrthoMADs [2], e OrthoMADS com SGTELIB (estes dois últimos com a implementação

NOMAD [3]). É esperado que o método desenvolvido ajude desenvolver ferramentas de automação para aliviar a carga sobre o engenheiro de processos, e melhorar a produção.

1.3 Estrutura

Este relatório está dividido em cinco capítulos. No capítulo 1, é dada uma introdução geral ao relatório. O capítulo 2 traz uma visão geral dos campos de petróleo e como eles funcionam, explicando a estrutura de uma FPSO[1], e seus componentes principais. A medida que os sistema são descritos, as variáveis usadas para os testes serão destacadas. O capítulo 3 oferece uma apresentação geral dos conceitos de otimização sem derivada, e são mostrados cada um dos métodos utilizados. No capítulo 4, são apresentadas as ferramentas envolvidas no trabalho. No capítulo 5 é apresentado o problema, a estrutura da solução proposta, como os componentes se conectam, as configurações e as condições dos testes, e variáveis envolvidas. Os capítulos 6 e 7 contêm os resultados dos experimentos realizados. Considerações finais e perspectivas para trabalhos futuros são apresentados no capítulo 8.

[1] Floating Production, Offloading and Storage, um tipo de estrutura utilizada para extração de petróleo e gás.

2 Modelagem e Simulação de Poços de Petróleo e Gás

2.1 Produção de Petróleo

Um poço de petróleo é uma perfuração na superfície terrestre conectando a um reservatório subterrâneo. Primeiramente, é iniciada uma perfuração na superfície utilizando-se brocas. Durante a perfuração, periodicamente o furo precisa ser reforçado, para evitar que todo o poço colapse. Este reforço é composto por cilindros estruturais que são instalados nas paredes externas da perfuração, que podem ser cimentados no exterior [4]. O poço então é finalizado, etapa em que são abertos buracos no envoltório estrutural para a passagem de óleo e gás. Os métodos para criar estes orifícios variam de buracos pré-existentes nos envoltórios a explosões controladas.

Após a conclusão dos orifícios, é bombeado líquido de fraturação para abrir novos canais no reservatório e facilitar o escoamento. O poço então recebe a árvore de natal (uma estrutura composta por um *choke*, indicador de pressão, e outras válvulas) que é conectada a um *manifold*, que concentra a produção para envio para a plataforma.

Em plataformas submarinas, podem existir diferentes configurações. Um poço principal pode conectar-se diretamente a um *manifold* submarinho, enquanto poços satélites podem conectar-se por linhas de fluxo multifásico ao *manifold* ou diretamente a um *manifold* na plataforma de produção. A esta última configuração, se dá o nome de poços satélites. Todos os equipamentos e válvulas nestes casos são atuados remotamente, através do umbilical que conecta à plataforma. A injeção de gás lift também pode ser feita pelo mesmo umbilical. O *manifold* então conecta-se ao *riser*, uma tubulação responsável por transportar os fluidos até a superfície.

O *riser*, dependendo do tipo de óleo e condições climáticas, pode necessitar de aquecimento ou diluentes para facilitar o transporte.

Ao chegar à superfície, a mistura vai para um separador, geralmente uma grande estrutura cilíndrica onde são separadas as diferentes fases do fluxo (gás, óleo e água), basicamente por ação da gravidade e diferença de densidade.

A separação pode ocorrer em diversos estágios, com diferentes pressões, para separar componentes distintos. A partir deste estágio, o óleo pode ser armazenado e enviado para refinarias por tubulações, ou armazenado no casco e descarregado por navios tanque semanalmente. Já o gás não costuma ser armazenado, e tende a ser re-utilizado ou enviado diretamente por tubulações, após um pré-processamento para se adequar às condições

exigidas pela tubulação.

2.2 Produção em Alto Mar

A produção de petróleo e gás pode ser estruturada em diversos layouts diferentes. Em águas rasas, até 100m, é comum a utilização de complexos de águas rasas, os quais são compostos de diversas plataformas, geralmente conectadas por pontes, cada uma com uma função principal, como acomodações, processamento, e geração de energia. Já entre 100 e 500m, é possível utilizar-se de bases de gravidade, grandes estruturas cônicas e ocas de concreto, que tem função tanto estrutural quanto de armazenamento.

Torres articuladas são uma estrutura semelhante às bases de gravidade, mas a base é conectada à plataforma por uma torre articulada, de forma a absorver os impactos de correntes marinhas e ventos.

FPSOs (*Floating, Production, Storage and Offloading*) são embarcações geralmente construídas a partir de navios-tanque, com uma torre giratória na proa ou central aonde são conectadas as tubulações, permitindo que o navio se alinhe livremente ao vento e correntes marítimas. As FPSOs dominam os novos campos de petróleo, já que os novos campos de exploração estão predominantemente em áreas de maior profundidade, e são o foco nesse trabalho.

2.3 Floating Production, Storage, Offloading

Uma FPSO (Floating Production, Storage and Offloading) é uma estrutura utilizada para produção de óleo e gás em grandes profundidades. Trata-se geralmente de um navio-tanque modificado[1], conectado a poços no fundo por *risers* e umbilicais, e ancorado no fundo do mar, como pode ser visto na Figura 1.

Estas estruturas são capazes de produzir, estocar e descarregar óleo de forma autônoma. O gás no entanto costuma ser re-utilizado para a injeção na forma de *gas-lift*, uma forma artificial de aumentar a produção de poços, transportado por tubulações, ou comprimido e liquefeito para o armazenamento e transporte.

No fundo do oceano podem existir um ou mais poços conectados a um *manifold*, centralizando a coleta, que por sua vez conecta-se à FPSO por meio de *flowlines* e *risers*. Outra configuração possível, chamada de poços satélites, é aquela em que todos os poços se ligam diretamente a um *manifold* a bordo da plataforma. Esta última tem a vantagem de ter menos componentes submersos, facilitando a manutenção.

[1] as FPSOs também podem ser construídas do zero, sem um navio tanque como base.

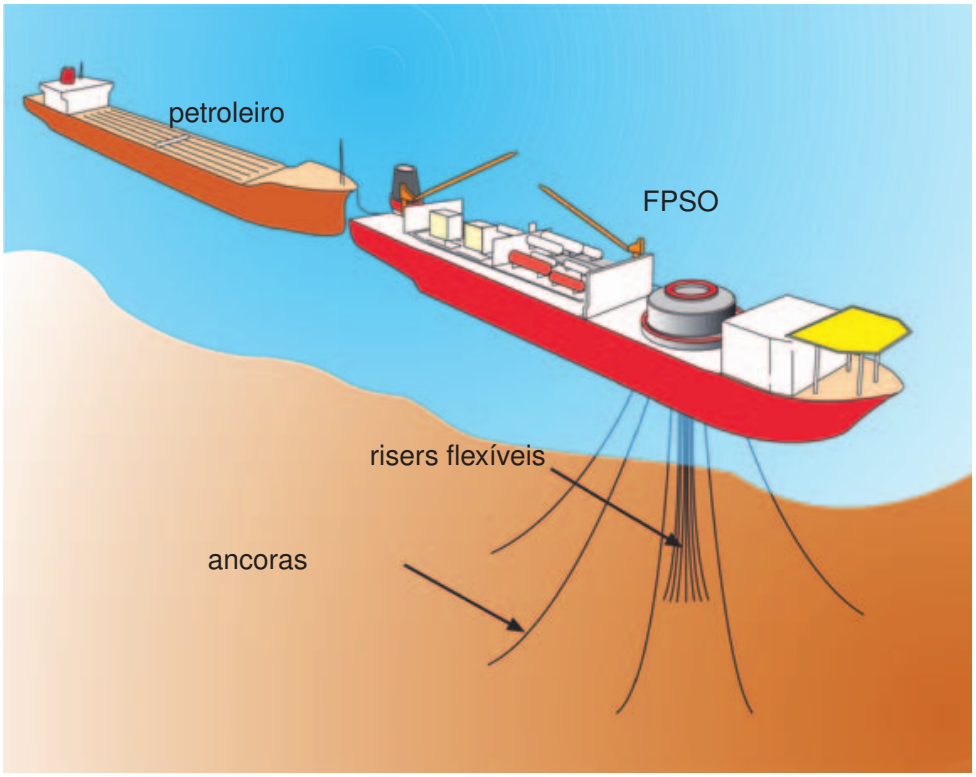


Figura 1 – Esquema de FPSO com poços satélites.

O gás, por ter sua armazenagem e transporte mais complicados, frequentemente é utilizado para *gas-lift*, ou escoado por tubulações submarinas, mas estas tubulações podem ter restrições de fluxo, requerendo certas especificações de composição e pressão, exigindo mais processamento.

Pequenas quantidades de hidrocarbonetos, em situações especiais, como desestabilização temporária de algum sistema, ou queda de um compressor, podem ser queimadas no *flare*. Embora fosse tradicional o uso contínuo do *flare* no passado, atualmente, por regulações ambientais, eles são utilizados apenas esporadicamente, quando necessário. No Brasil o seu uso é penalizado pela ANP (Agência Nacional do Petróleo).

2.4 Elevação Artificial por Gas Lift

À medida que um reservatório é escoado, a sua pressão interna diminui, o que em alguns casos faz com que não haja pressão o suficiente para vencer a pressão da coluna de produção e manter a vazão desejada apenas com o controle da abertura do *choke*.

Uma solução para este problema é o *gas-lift*, um método para a elevação artificial de fluidos, largamente empregado na indústria do petróleo. Este método consiste na injeção de gás pressurizado nos poços facilitando o deslocamento dos fluidos até a plataforma de produção. Ele funciona tanto pelo efeito da energia da expansão do gás injetado, quanto pela diminuição da densidade média do fluido no *riser*, causada pela mistura de mais gás

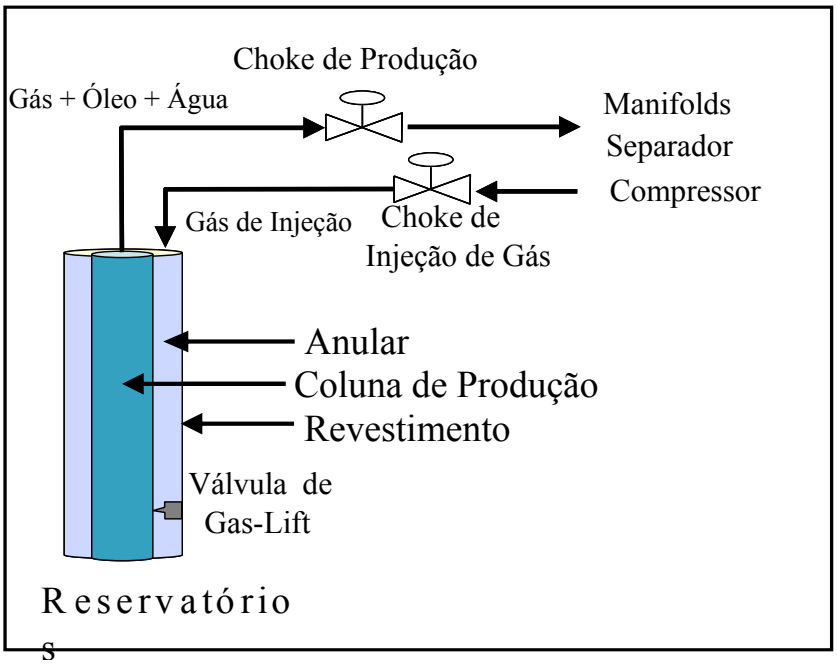


Figura 2 – Funcionamento do gas-lift.

à mistura que é produzida.

Uma característica do processo de *gas-lift* visível neste trabalho é que existe um ponto ótimo de injeção, pois após um certo ponto, adicionar mais gás apenas aumenta a produção do próprio gás, ao invés de produzir mais líquidos. Para a utilização do *gas-lift* são necessários compressores para pressurizar o gás novamente e reinjetá-lo no poço. Para decisão da quantidade de gás injetado, podem ser utilizados simuladores de fluxo multifásico, como o Pipesim, da Schumblerger, introduzido na seção 4. Estes software, se bem sintonizados, são capazes de fornecer curvas de produção por gás injetados, que auxiliam os engenheiros da operação.

3 Otimização Livre de Derivadas

3.1 Visão Geral

Frequentemente em problemas de engenharia os modelos utilizados são aproximações simplificadas da realidade, ou mesmo completamente desconhecidos.

Métodos de otimização sem derivadas são adequados em casos aonde o modelo matemático é não-explícito, sua avaliação é custosa, ou as derivadas não estão disponíveis devido à inexistência do modelo explícito ou presença de ruídos, impossibilitando a estimação das derivadas.

Os métodos de otimização sem derivadas procuram encontrar mínimos realizando o menor número possível de simulações, de modo a tentar minimizar também o tempo de execução da otimização.

3.2 Método do Simplex de Nelder-Mead

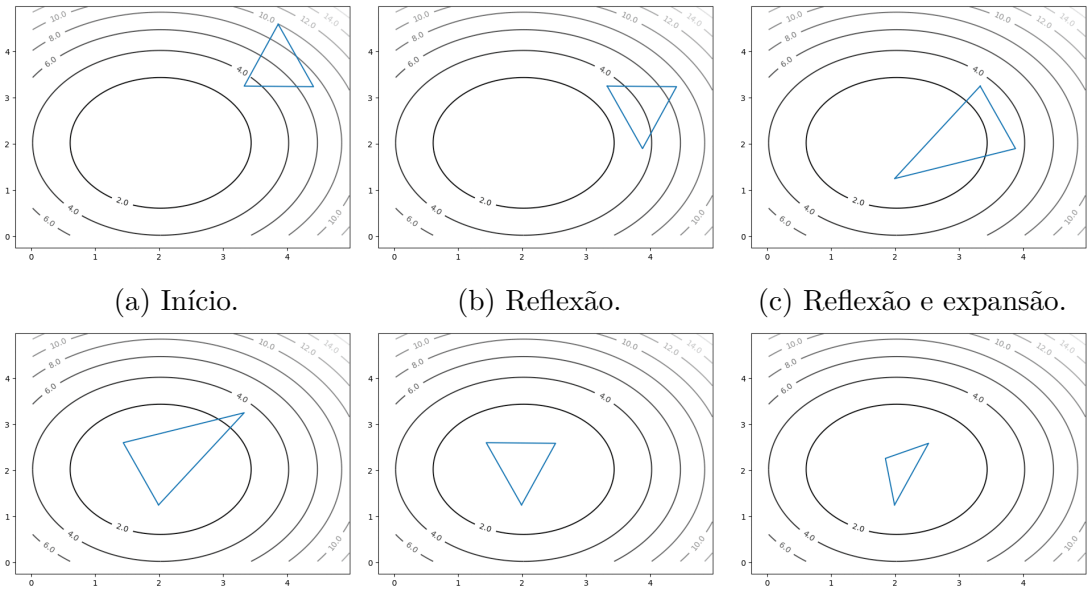
Também conhecido como Downhill Simplex Method, ou Amoeba Method, consiste em utilizar um polígono com $n + 1$ vértices em n dimensões que se expande, contrai, ou reflete de modo a se mover em direção ao gradiente da função objetivo.

A partir de um ponto inicial, o simplex se locomove como uma ameba até encontrar um ponto de mínimo.

Um exemplo passo-a-passo da execução do algoritmo de Nelder-Mead pode ser visto na figura 3, aonde se vê, nesta ordem: uma reflexão, reflexão com expansão, outra reflexão, e dois encolhimentos. A figura 4 ilustra os passos totais e o valor da função objetivo com as iterações. A execução detalhada pode ser analisada pelo algoritmo 1.

3.3 OrthoMADS

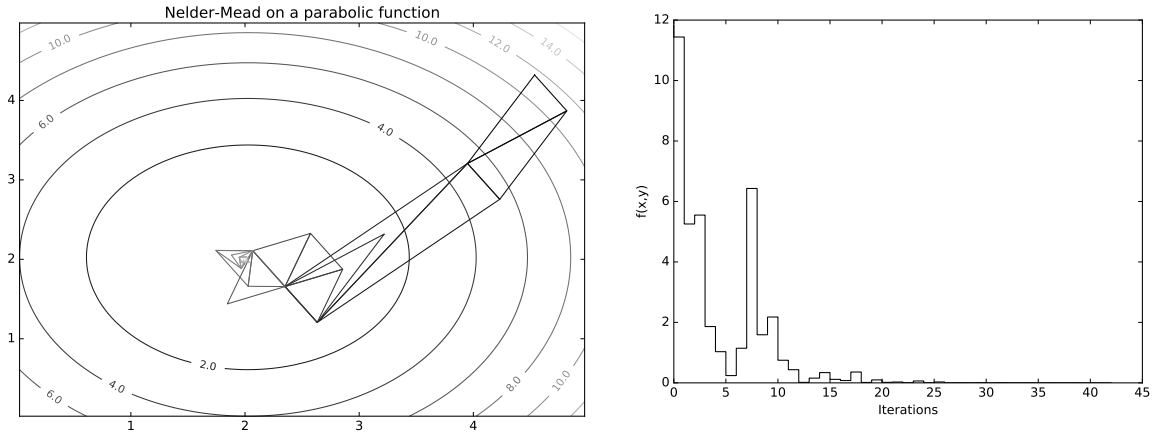
Os métodos MADS (Mesh Adaptative Direct Search) são uma classe de métodos de otimização Direct Search, cuja estratégia busca encontrar um ponto ótimo de uma função ao avaliar e comparar apenas o valor da função em diversos pontos [5]. Os métodos *Mesh Adaptative* se destacam pelo fato de que todos os pontos avaliados estão posicionados em uma malha, que pode ser refinada. A cada iteração k são executados dois passos de busca, *Search* e *Poll*, analisando a viabilidade e valor da função. O objetivo de cada nova iteração é encontrar um ponto em que $f(x) < f(x_k)$, aonde x_k é o melhor ponto encontrado até a iteração atual. As buscas são feitas sempre em uma grade, definida por:



(a) Início. (b) Reflexão. (c) Reflexão e expansão.

(d) Reflexão. (e) Contração. (f) Contração.

Figura 3 – Ilustração do Nelder-Mead passo-a-passo.



(a) Todos os passos do Nelder-Mead

(b) Pontos avaliados pelo Nelder-Mead aplicado em um parabolóide.

Figura 4 – Demonstração gráfica do Nelder-Mead

Algorithm 1 Nelder-Mead's Downhill Simplex

Require:

$X = (x_1, ..., x_{n+1})$ pontos de teste:
$\alpha = 1$ (coeficiente de reflexão)
$\gamma = 2$ (coeficiente de expansão)
$\rho = 0.5$ (coeficiente de contração)
$\sigma = 0.5$ (coeficiente de encolhimento)

while Não Convergiu **do**
    ordenar X; $(f(x_1) \leq f(x_2) \leq \cdots \leq f(x_{n+1}))$
    Calcular o centroide x_0 de $(x_1, ..., x_{n+1})$
    Calcular x_r refletido: $x_r = x_0 + \alpha(x_0 - x_{n+1})$
(Reflecção)
    if $f(x_1) < f(x_r) < f(x_n)$ **then**
        $x_{n+1} \leftarrow x_r$
        Continue
    end if
(Expansão)
    if $f(x_r) < f(x_1)$ **then**
        Calcular ponto expandido $x_e = x_0 + \gamma(x_r - x_0)$
        if $f(x_e) < f(x_r)$ **then**
            $x_n \leftarrow x_e$
        else
            $x_n \leftarrow x_r$
        end if
        Continue
    end if
(Contração)
    Computar o ponto contraído $x_c = x_0 + \rho(x_{n+1} - x_0)$
    if $f(x_c) < f(x_{n+1})$ **then**
        $x_{n+1} \leftarrow x_c$
        Continue
    end if
(Encolhimento)
    $x_i \leftarrow x_1 + \sigma(x_i - x_1), i = 2 \ldots n + 1$
end while

$M_k = \{x + \Delta_k^m Dz : x \in V_k, z \in \mathbb{N}^{n_D}\} \subset \mathbb{R}^n$ Onde M_k é o conjunto de pontos da grade, x é o ponto mínimo atual, Δ_k^m é o parâmetro de tamanho da malha, D é uma matriz $\mathbb{R}^{n \times n_D}$ composta por n_D direções que definem um conjunto gerador no $\mathbb{R}^n$. Para o OrthoMads e LtMads, D é simplesmente definida como $[I_n \quad -I_n]$ aonde I_n é a matriz identidade de dimensões n. O passo *Search* pode ser qualquer tipo de heurística que escolha um ponto mais adequado da malha para tentar acelerar a convergência. Neste trabalho são utilizado um polinômio quadrático e a SGTELIB(*Surrogate lib*). O passo *Poll* é a parte mais importante do método, que garante sua convergência. A cada iteração k os pontos a serem utilizados são definidos por:

$$P_k = \{x_k + \Delta_k^p d : d \in D_k\} \subset M_k$$

Aonde x_k é o ponto atual e cada coluna de D_k é formada por combinações inteiras das colunas de D de forma a criar um conjunto gerador positivo. Δ_k^p é o *parâmetro de tamanho de poll*. Ambos LtMads e OrthoMads utilizam um parâmetro ℓ_k chamado de *índice de malha* para atualizar os parâmetros de tamanho de *poll* e *search* de acordo com esta lógica:

$$\Delta_k^p = 2^{-\ell k} \text{ e } \Delta_k^m = \min\{1, 4^{-\ell_k}\} \tag{3.1}$$

A cada nova iteração, se em uma iteração um novo *incumbente*[1] não é encontrado, ela é dita mal sucedida, e $\ell_{k+1} \leftarrow \ell_k + 1$ (reduzindo Δ_k^m e Δ_k^p), por outro lado, se for encontrado um novo *incumbente*, a iteração é dita bem sucedida, e $\ell_{k+1} \leftarrow \ell_k - 1$ (aumentando Δ_k^m e Δ_k^p). Devido à definição (3.1), no caso de uma iteração mal-sucedida o parâmetro de *Poll* diminui mais rápido que o de malha, de modo a permitir o uso de mais pontos, refinando a malha.

A diferença entre OrthoMads e LtMads se dá na geração da base D_k. LtMads utiliza uma matriz triangular inferior para a geração da base, fazendo permutações entre os elementos e completando ela em uma base maximal ou minimal, sem garantir ortogonalidade entre as direções, de modo que os ângulos entre as direções podem ser grandes, causando grandes cones de espaço não explorado. Já OrthoMads utiliza uma base maximal definida por $[H_k \quad -H_k]$, aonde as colunas de H_k formam um base ortogonal de $\mathbb{R}^n$. Além disso, as direções de D_k são inteiras, de modo que os pontos gerados estão automaticamente contidos na malha definida por $D = [I_n - I_n]$.

Para a geração de D_k, OrthoMads utiliza a sequência pseudo-aleatória de Halton, que cobre mais uniformemente o espaço que uma sequência aleatória real, para gerar vetores u_t.

[1] Ponto com o melhor valor da função custo até então.

A saída da sequência de Halton, no entanto, não respeita as restrições impostas pela malha. É necessário arredondar, escalar e rotacionar o vetor u_t. O índice ℓ é utilizado para transformar a direção u_t na *direção ajustada de Halton* $q_{t,\ell} \in \mathbb{Z}^n$, uma direção cuja norma é próxima a $2^{\frac{|\ell|}{2}}$

Para definir $q_{t,\ell}$, primeiramente são definidas duas funções baseadas na t-ésima direção de Halton u_t:

$$q_t(\alpha) = \text{round}\left(\alpha \frac{2u_t - e}{||2u_t - e||}\right) \in \mathbb{Z}^n \cap \left[-\alpha - \frac{1}{2}, \alpha + \frac{1}{2}\right]^n$$

Onde *round* é a operação arredondar para cima ($round(0,5) = 1$, $round(-0,5) = -1$) e $\alpha \in \mathbb{R}_+$ é um fator de escala.

Desta forma, tem-se um problema de otimização, que consiste em encontrar um $\alpha_{t,\ell}$ tal que $|q_t(\alpha_{t,\ell})|$ seja o mais próximo possível de $2^{\frac{|\ell|}{2}}$ sem ultrapassá-lo.

$$\begin{aligned} \alpha_{t,\ell} \in \arg\max_{\alpha \in \mathbb{R}_+} & ||q_t(\alpha)|| \\ \text{s.t.} & ||q_t(\alpha)|| \leq 2^{\frac{|\ell|}{2}} \end{aligned}$$

O problema pode ser resolvido facilmente, já que que os degraus da função $||q_t(\alpha)||$ acontecem em todos os α no conjunto

$$\left\{ \frac{(2j+1)||2u_t - e||}{2|u_t^i - e|} : i = 1, 2, \ldots, n, j \in \mathbb{N} \right\}$$

De forma que o problema pode ser solucionado varrendo os pontos do conjunto.

Com um vetor normalizado e na malha, $q \in \mathbb{Z}^n$, é necessário transformá-lo em uma base ortonormal de $\mathbb{R}^n$. Para isto é utilizada a transformação de Householder:

$$H = ||q^2||(I_n - 2vv^T), \ \ onde\ v = \frac{q}{||q||} \tag{3.2}$$

Onde H é uma base ortonormal gerada a partir de q. Com a base ortonormal criada, é possível utilizar o algoritmo 2, comum ao LtMads e OrthoMads.

Algorithm 2 OrthoMads

[0] Inicialização
$x_0 \in \Omega, \ell_0 \leftarrow 0, k \leftarrow 0, t_0 \leftarrow p_n$
while Não Convergiu **do**
(ITERAÇÃO k)
    Search (opcional)
    Avalia f em um conjunto finito $S_k \subset M_k$
    POLL
    if menor tamanho de POLL até então ($\Delta_k^p = \min\{\Delta_j^p : j = 0, 1, \ldots, k\}$) **then**
        $t_k \leftarrow \ell_k + t_0$
    else (Já foram considerados tamanhos menores)
        $t_k \leftarrow 1 + \max\{t_j : j = 0, 1, \ldots, k-1\}$
    end if
    Computa $u_{tk}, q_{tk}\ell_k$ and $D_k = \begin{bmatrix} H_{tk} & -H_{tk} \end{bmatrix}$
UPDATES
    if A iteração foi bem sucedida
(se existe um $x_s \in S_k$ ou $x_p \in P_k$ tal que $f(x_s) < f(x_k)$ ou $f(x_p) < f(x_k)$) **then**
        $x_{k+1} \leftarrow x_s$ or x_p
        $\ell_{k+1} \leftarrow \ell_k - 1$
    else(iteração falhou)
        $x_{k+1} \leftarrow x_k$
        $\ell_{k+1} \leftarrow \ell_k + 1$
    end if
    $k \leftarrow k + 1$
end while

4 Ferramentas Computacionais

4.1 O Simulador Pipesim

Sistemas de produção de petróleo são sistemas nos quais se deseja sempre produzir o máximo possível e de forma segura. O PIPESIM, da Schlumberger, é um simulador de fluxo multifásico em regime permanente que pode ser utilizado tanto para o projeto como para planejamento de operações em campos de petróleo. Ele permite que sejam simuladas situações alternativas de forma mais rápida e segura que testes reais. É possível simular desde um único poço, como no projeto atual, até uma complexa rede de produção como as em uso pela Petrobras.

4.2 Python

O Python é uma linguagem de alto nível, interpretada, de desenvolvimento rápido muito utilizada para prototipação e na academia. Suas faciliades, como Jupyter Notebook [6], um ambiente que facilita a criação de documentos com códigos e análise de dados, e o Matplotlib [7], uma biblioteca que a torna quase tão poderosa quando o MatLab para visualização de dados, a tornaram muito utilizada também na academia.

Python tem como filosofia de design a legibilidade do código (foi escolhido o uso de espaços no lugar de chaves pra delimitação de blocos de código) e uma linguagem que facilita a expressão de conceitos complexos em poucas linhas de código se comparada, por exemplo, com Java ou C++. A linguagem expõe estruturas e conceitos complexos em outras linguagens de forma mais fácil e intuitiva de utilizar.

A Linguagem surgiu em 1991 [8], e segundo Tiobe [9] é a quarta linguagem de programação mais popular atualmente.

Sua facilidade de uso, expêriencia prévia não necessária, disponibilidade de ferramentas, e facilidades de visualização de dados foram decisivos para a sua escolha para este trabalho.

4.3 Interface Python e Open Link

Para interfaceamento do Pipesim com outros software, a Schlumberger disponibiliza uma API (Application Programming Interface) chamada Open Link, idealizada para programação em C++, VBA, ou Visual Basic. Esta pode ser utilizada para interação programática com o Pipesim, habilitando a configuração de novos poços, alteração de po-

ços existentes, análises, simulações e automação de simulações. Com essa API, é possível variar os parâmetros do poço e avaliar as curvas características.

Utilizando-se a biblioteca de Python pyWin32, é possível comunicar-se com a API, mas apenas em versões do Windows de 32 bits. Apesar de algumas peculiaridades no tratamento de arrays e outros tipos de dados, esta biblioteca permite o uso da API em uma linguagem de desenvolvimento mais rápido [10] e com grandes facilidades de análise de dados.

4.4 NOMAD

Para utilização do OrthoMADS, foi escolhida a ferramenta NOMAD [3], uma implementação em C++ do OrthoMADS desenvolvida pelo GERAD [11], um centro de pesquisa multi-universidades canadenses. Seu objetivo é a solução de problemas de otimização sem-derivadas com problemas caixa-preta, aonde não se conhece o modelo explícito do problema. Bastando fornecer uma função objetivo e restrições, a ferramenta é capaz de encontrar um ponto ótimo para o problema. Ela também disponibiliza variações do algoritmo (como $2n$ ou $n+1$ bases) e a possibilidade de paralelismo (consultar mais que um ponto de forma concorrente).

4.5 Opal

Para o uso do NOMAD com o Python, era sugerido, até o começo destes trabalhos, o uso da ferramenta Opal [12] (A Framework for Optimization of Algorithms). Uma interface open-source de alto nível para a interface de Python com o NOMAD. Este framework dá a liberdade para configurar todos os parâmetros de otimização do NOMAD e também é capaz de paralelismo.

No entanto ele suporta apenas Python 2.7 que está para ser aposentado em 2020 [13], de modo que foi necessário portá-lo para Python 3 [14], contribuição que deve ser enviada para os repositórios originais assim que finalizada.

É interessante ressaltar também que a partir da versão 3.8.0 do NOMAD, foi implementada um interface própria em Python (novamente Python2.7) que deve ser analisada para uso em trabalhos futuros.

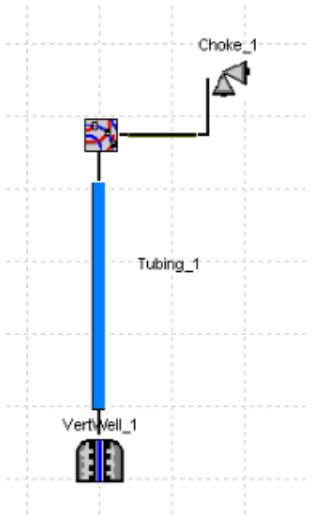
O Opal é um framework complexo, que embora seja utilizado atualmente apenas para o uso do NOMAD, contém a base para implementação de outros solvers genéricos, suporte a paralelismo, e a outras plataformas, como Sun Grid Engine e Symmetric Multiproccessing (SMP).

5 O Problema de Sintonia

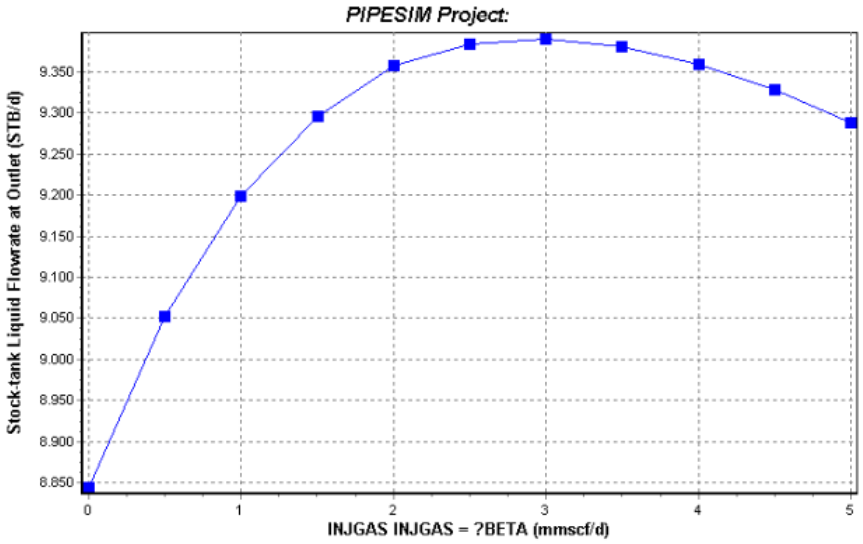
A sintonia de simuladores de poços de petróleo costuma ser feita a partir de dados experimentais. A partir de testes de campo, adquirem-se pontos de testes do sistema real. Os pontos reais são utilizados como referência para que o operador varie os parâmetros de forma a encontrar o modelo que melhor represente os dados coletados.

Para os testes neste trabalho, primeiramente foi configurado um único poço de petróleo, com uma estrutura simples e que pode ser visto na figura 5a. A seguir foram escolhidos dados arbitrariamente para compor a curva "real" de produção (figura 5b). Neste Experimento a curva sintonizada foi a de fluxo de líquido (Barris padrões por dia) por gás injetado (milhões de pés cúbicos padrões por dia), no entanto é possível utilizar outras curvas, ou ainda mais que uma, para a sintonia.

Escolheu-se a pressão estática do reservatório como sendo 4000 psi absoluto, e um índice de produtividade de líquido de 25 STDB/d/psi (barris padrões por dia por pressão estática). Desta forma a figura 5b demonstra o poço "real" a ser sintonizado.



(a) Setup do poço de petróleo.



(b) Curva "real" do poço de petróleo.

Figura 5 – Configuração e curva do poço do experimento 1.

Em todos os testes, os parâmetros s_p (pressão estática) e l_{pi} (Índice de produção de líquido) foram iniciados respectivamente em 3000 (psi) e 15 (STDB/d). A interface OpenLink foi utilizada para modificar os parâmetros e ler uma nova curva de fluxo de líquido por injeção de gás, a distância quadrática entre as duas curvas foi utilizada como o erro na otimização.

O problema de otimização, então, é dado por

$$\min_{x \in \Omega} f(x). \tag{5.1}$$

Onde:

$$x = (s_p, l_{pi}) \tag{5.2}$$

$$\Omega = \{s_p \in \mathbb{R} \mid s_{p,min} < s_p < s_{p,max}\} \tag{5.3}$$

$$\{l_{pi} \in \mathbb{R} \mid l_{pi,min} < l_{pi} < l_{pi,max}\}. \tag{5.4}$$

Em que s_p é a pressão estática do reservatório e l_{pi} é o índice de produtividade de líquidos.

Seja $f(q_{inj}; x)$ a produção de líquido estimada pelo simulador quando sintonizado com parâmetros x ao se injetar q_{inj} gás no poço, e seja $\theta(q_{inj}; x)$ a produção "real" de líquido com parâmetros x. Então:

$$f(x) = f(s_p, l_{pi}) \tag{5.5}$$

$$= \sum_{i=1}^{n} \left[\theta(q_{inj}^n; x) - f(q_{inj}^n; x))\right]^2 \tag{5.6}$$

Em que $q_{inj}^1, q_{inj}^2, \ldots, q_{inj}^n$ são os pontos amostrados da curva de produção.

Neste problemas foram impostos:

$$l_{pi,min} = 15$$
$$l_{pi,max} = 35$$
$$s_{p,min} = 2000$$
$$s_{p,max} = 7000$$

6 O Experimento de Sintonia Sem Ruído

Foram realizados três experimentos de sintonia de curva, e utilizadas três abordagens diferentes para a sintonia. Primeiramente foi utilizado o OrthoMADS, com a implementação NOMAD, em seguida, foi utilizado novamente o NOMAD, mas com um surrogate model calculado pela SGTELIB, e finalmente, uma implementação do Nelder-Mead Simplex. Os resultados então são comparados quanto a número de avaliações e acurácia do resultado.

Os experimentos foram realizados em uma máquina virtual VirtualBox, rodando em um Xeon 2665, e utilizando dois núcleos.

6.1 Setup para sintonia de curva com o orthoMADS

Para a sintonia com o orthoMADS, foi utilizado o framework OPAL (A Framework for Optimization of Algorithms), interface Python para o solver NOMAD. A implementação com o OPAL utiliza dois arquivos, "well_declaration.py"e "well_optimize.py".

O arquivo "well_declaration.py"expõe os componentes do problema:

- Nome do algoritmo:

```
# Define Algorithm object.
tuning = Algorithm(name='TUNING', description='Well Tuning')
```

- Comando utilizado pelo solver para avaliar a função:

```
tuning.set_executable_command('python pipesim_run.py')
```

- As variáveis de decisão:

```
static_pressure = Parameter(kind='real',
                            default=sp,
                            bound=(2000, 7000),
                            name='sp',
                            description='Static Pressure')
liq_pi = Parameter(kind='real',
                   default=pi,
                   bound=(15, 35),
                   name='pi',
                   description='Liq PI')

FD.add_param(static_pressure)
FD.add_param(liq_pi)
```

- E o erro:

```
error = Measure(kind='real', name='ERROR', description='Curve
    quadratic error')
FD.add_measure(error)
```

Já no arquivo "well_optimize", são declaradas estruturas auxiliares:

- É instanciado o solver:

```
def get_error(parameters, measures):
    return sum(measures["ERROR"])

data = ModelData(FD)
struct = ModelStructure(objective=get_error)  # Unconstrained
model = Model(modelData=data, modelStructure=struct)

NOMAD = NOMADSolver()
```

- São impostas as seguintes restrições ao solver do NOMAD:

```
F_TARGET = 0.1
```

De modo a limitar o tamanho mínimo da malha, e terminar a simulação quando a função custo chegar a um valor abaixo de 0.1

- E é inicilizada a otimização:

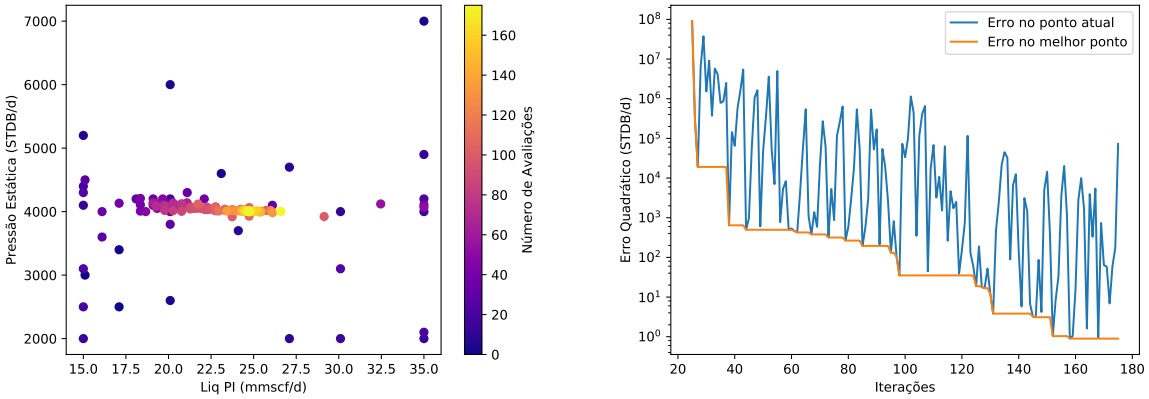
```
NOMAD.solve(blackbox=model)
```

6.2 Resultados da Sintonia de Curva com o orthoMADS

Em 525 segundos (8:45 minutos) e com de 206 avaliações de caixa preta. A performance do algoritmo pode ser vista na figura 6. A parada se deu pelo valor da função custo estar abaixo de 0,1. A convergência se deu para os seguintes valores:

$$s_p = 4000,691$$
$$l_{pi} = 24,954$$
$$Custo = 0,0757.$$

Que são muito próximos aos valores reais $s_p = 4000$ e $l_{pi} = 25$.



(a) Pontos escolhidos pelo NOMAD.

(b) Erro nos pontos avaliados.

Figura 6 – NOMAD aplicado ao experimento 1.

6.3 A Surrogate Lib

Durante a realização desses experimentos, o NOMAD foi atualizado e recebeu uma ferramenta auxiliar, uma biblioteca que tenta aproximar a função a partir dos pontos já utilizados, para utilização no passo "search"do algoritmo. Esta biblioteca faz com que o algoritmo, a cada iteração, tente adivinhar a melhor direção para avançar, ao invés de progredir aleatoriamente.

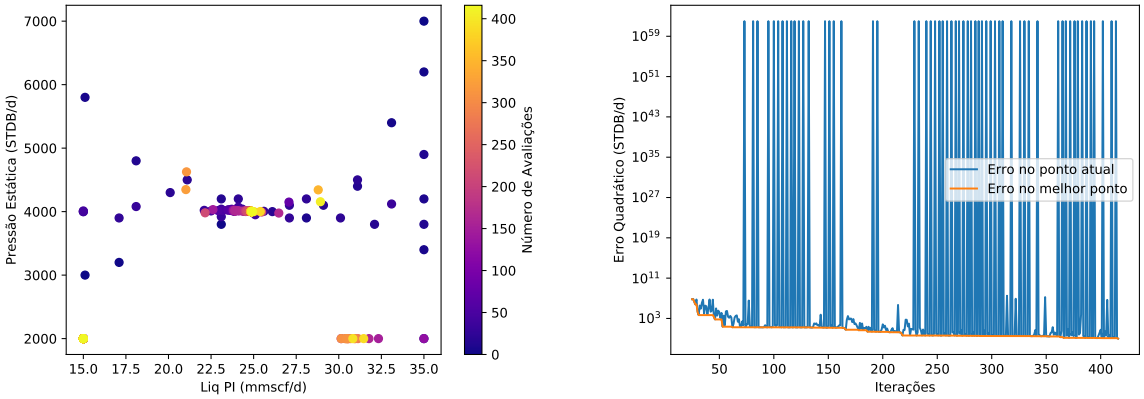
Para o cálculo do modelo substituto, são possíveis nove tipos de modelos, que podem ser vistos na documentação, além de onze tipos possíveis de núcleos. Neste trabalho foram utilizadas as opções padrão do NOMAD, um modelo PRS (Polynomial Response Surface) de ordem 2.

6.4 Setup Para Sintonia de Curva Com a Surrogate Lib

Um contra-tempo quanto a SGTELIB, como é chamada a biblioteca, é que ela está embutida nos binários do NOMAD, e, por algum erro, os binários para windows foram compilados com uma versão antiga do Microsoft Visual Studio, de forma que é necessário recompilar não apenas o SGTELIB (o que pode ser feito com o mingW sem grandes mudanças), mas todo o NOMAD, necessitando a instalação de aproximadamente 4Gb do Visual Studio.

Após sua recompilação, bastou substituir o binário antigo pelo novo, é inserir nos parâmetros do NOMAD para o uso da SGTELIB, com seus valores padrõess:

```
1 NOMAD.set_parameter(name="MODEL_SEARCH", value="SGTELIB")
```



(a) Pontos escolhidos pelo NOMAD.

(b) Erro nos pontos avaliados.

Figura 7 – NOMAD aplicado ao experimento 1 com SGTELIB.

Para que ele passasse a utilizar o SGTELIB no local do seu SEARCH baseado em modelo quadrático.

6.5 Resultados da Otimização Com a Surrogate Lib

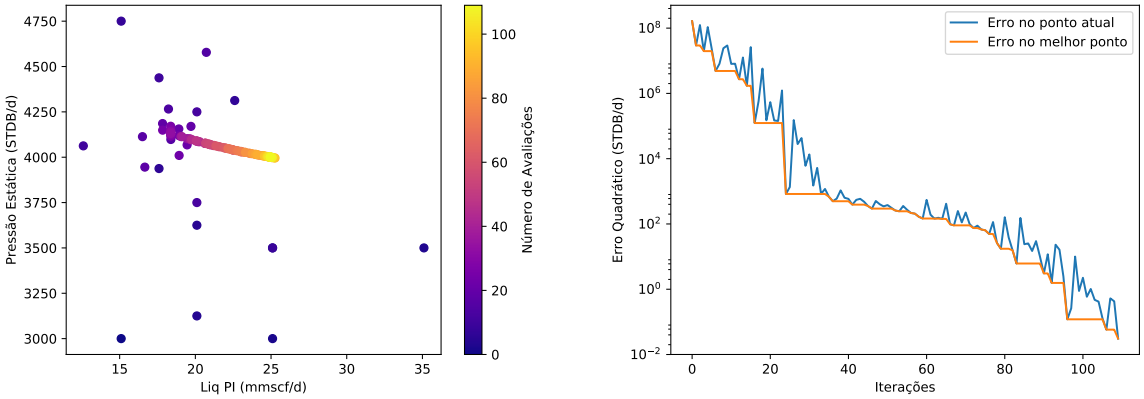
A execução demorou 1374 segundos (22:27 minutos) e necessitou de 417 avaliações de função de caixa preta. O Nomad novamente convergiu, desta vez para os valores:

$$s_p = 4001,389$$
$$L_{pi} = 24,905$$
$$Custo = 0,09986.$$

É possível notar, no entanto, que com o SGTELIB, em muitos pontos o erro quadrático saturou em 10^{64}. Nota-se também que pontos longe do ótimo foram avaliados perto da convergência (pontos claros no canto inferior esquerdo), o que pode significar que existe um problema com a SGTELIB, ou que o modelo utilizado não é o ideal, talvez pelo fato de que as superfícies polinomiais tendam a crescer muito para longe na origem, levando o algoritmo a testar pontos distantes.

6.6 Setup da Otimização com Nelder-Mead Simplex

Para o teste com o Simplex de Nelder-Mead, foi utilizada uma implementação própria, baseada no algoritmo 1, e como ele não é facilmente paralelizável, não houveram



(a) Pontos escolhidos pelo Nelder-Mead.

(b) Erro nos pontos avaliados.

Figura 8 – Nelder-Mead aplicado ao experimento 1.

preocupações com paralelismo. Os limites utilizados foram os mesmos dos casos anteriores:

$$15 < l_{pi} < 35 \tag{6.1}$$

$$2000 < s_p < 7000 \tag{6.2}$$

Como o simplex utiliza uma estrutura n-dimensional para a busca, são necessários $n+1$ pontos iniciais, por este motivo um triângulo foi expandido arbitrariamente a partir do ponto inicial, de modo que no começo do algoritmo, o simplex inicial seja grande. As restrições foram aplicadas somando-se um peso adicional a função custo.

Os resultados podem ser vistos nas figuras 8 e 9. Nota-se que a solução foi mais rápida, com 212 iterações em 133 segundos (2:13 minutos). No entanto, o método de Nelder e Mead não tem as mesmas características de convergência que os os métodos MADS. É interessante comentar o caso da figura 5, aonde é possível notar que o simplex primeiramente moveu-se para um ponto em torno de $l_{pi} = 18$, depois moveu-se lentamente para o ponto final:

$$s_p = 4001,389$$
$$l_{pi} = 24,905$$
$$Custo = 0,09986.$$

6.7 Discussão

É possível notar, pela figura 10 e pela 1 que o Nelder-Mead, embora tenha utilizado um número semelhante de iterações que o orthoMADS, foi aproximadamente quatro vezes

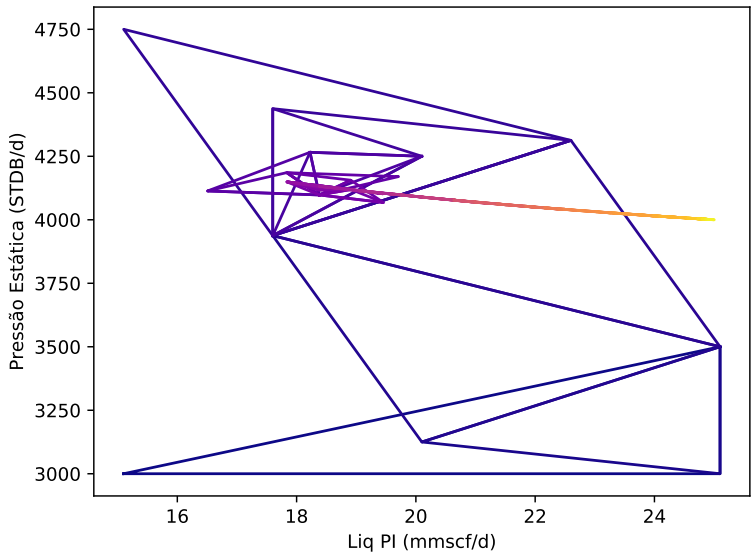


Figura 9 – Simplexes utilizados pelo Nelder-Mead.

Tabela 1 – Tempo de execução e numero de iterações por algoritmo

	Tempo	Iterações
OrthoMADS	525	206
OrthoMADS + SGTELIB	1374	417
Nelder-Mead	133	110

mais rápido. É possível que isso se dê pelo fato de que o OPAL, embora utilize várias *threads*, talvez não faça um gerenciamento ideal do processador, e sim faça um *busy wait*, que é quando um processo espera um sinal (no caso o fim de uma *thread*) sem liberar o processador para outras *threads*. Embora exista a suspeita do problema, não foi possível localizá-lo no código fonte.

Outra possível interpretação é que a diferença seja devido ao *overhead* no NOMAD. Todas suas iterações envolvem o disparo de novos processos do Python, que por sua vez disparam o processo do Pipesim, além da escrita e leitura de diferentes arquivos para a comunicação entre os processos. Como a implementação utilizada do Nelder-Mead foi feita diretamente em Python, o *overhead* existente é apenas a interface com Open Link.

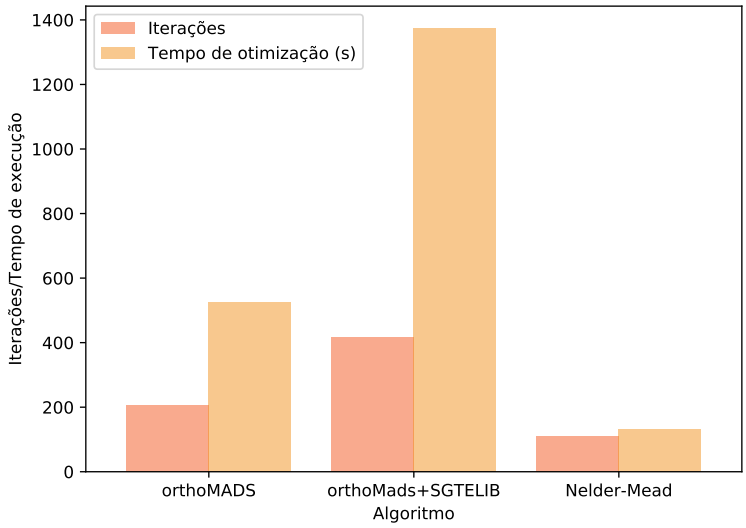


Figura 10 – Comparação entre tempos de execução e número de iterações entre os três experimentos.

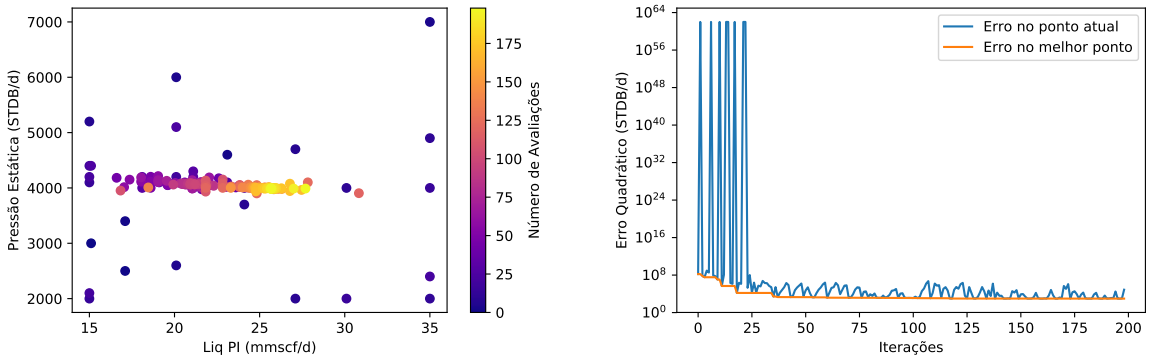
7 O Experimento de Sintonia Com Ruído

Para testar um caso um pouco mais realista, foram introduzidos ruídos ao sistema. Primeiramente foi introduzido um ruído de medição nos pontos "reais" com desvio padrão $\sigma = 10$ (STDB/d) e média nula, para uma distribuição gaussiana. Este ruído simula as condições adversas da medição em campo, perdas em sensores, sinais elétricos, imprecisões de montagem, e etc...

Adicionalmente, foi adicionado um pequeno ruído na simulação, para emular erros de precisão no caso de simuladores com soluções iterativas, que podem parar a simulação antes da precisão completa. A este ruído foi atribuído também uma distribuição normal com média zero, mas $\sigma = 10^{-5}$ para cada variável. Como não existe mais a possibilidade um *match* perfeito (zerar a função custo), é bom pensar no critério de parada. Como um balanço de precisão e execução, é escolhido o critério de que os pontos explorados estão a menos de 10^{-3} unidades de distância em cada variável. Para as novas configurações, foram repetidos os experimentos.

7.1 Sintonia de Curva com Ruído Com OrthoMADS

Com o OrthoMADS, a otimização levou 334 segundos (5:34 minutos) e precisou de 130 avaliações. OS resultados podem ser analisados na figura 11, que mostra os pontos avaliados e progressão do erro com as avaliações, e na figura 12, aonde é possível notar que a curva identificada está muito próxima a real, apesar do ruído na curva medida.



(a) Pontos escolhidos pelo NOMAD.

(b) Erro nos pontos avaliados.

Figura 11 – NOMAD aplicado ao experimento 2.

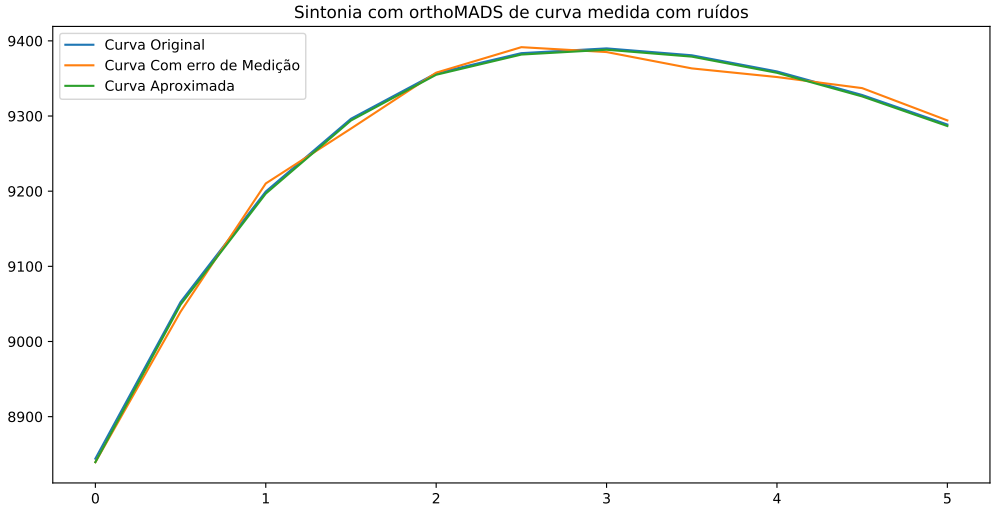
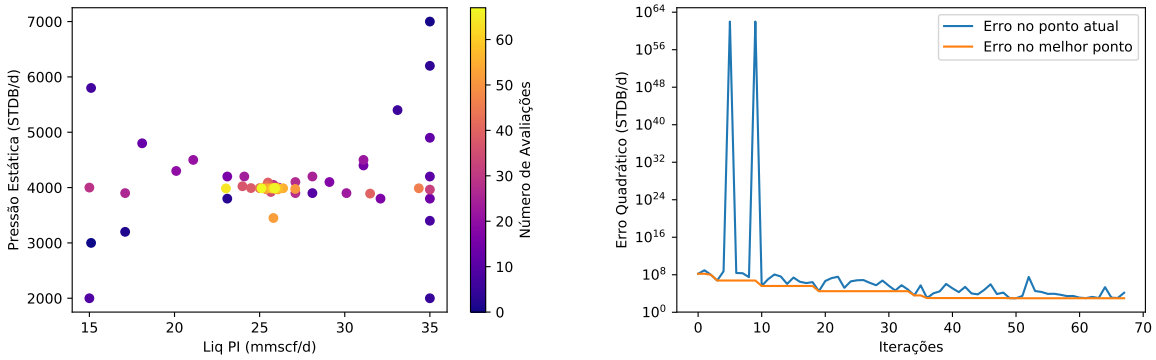


Figura 12 – Resultado da sintonia utilizando-se o OrthoMADS.

7.2 Sintonia de Curva com Ruído Com OrthoMADS + SGTELIB

Com o OrthoMADS e a SGTELIB, a otimização levou 187 segundos (3:07 minutos) e precisou de 68 avaliações de caixa preta. Os resultados podem ser avaliados na figura 13, que mostra os pontos avaliados e progressão do erro com as avaliações, e na figura 14, aonde é possível notar que a curva identificada também está muito próxima a real, apesar do ruído na curva medida.

Também é notável que o novo método de SEARCH foi capaz de diminuir o número de avaliações (52%). Ele converge para um ponto próximo ao do método anterior (respectivamente 982,524 e 956,827), consumindo menos tempo e avaliações.



(a) Pontos escolhidos pelo NOMAD.

(b) Erro nos pontos avaliados.

Figura 13 – NOMAD com SGTELIB aplicado ao experimento 2.

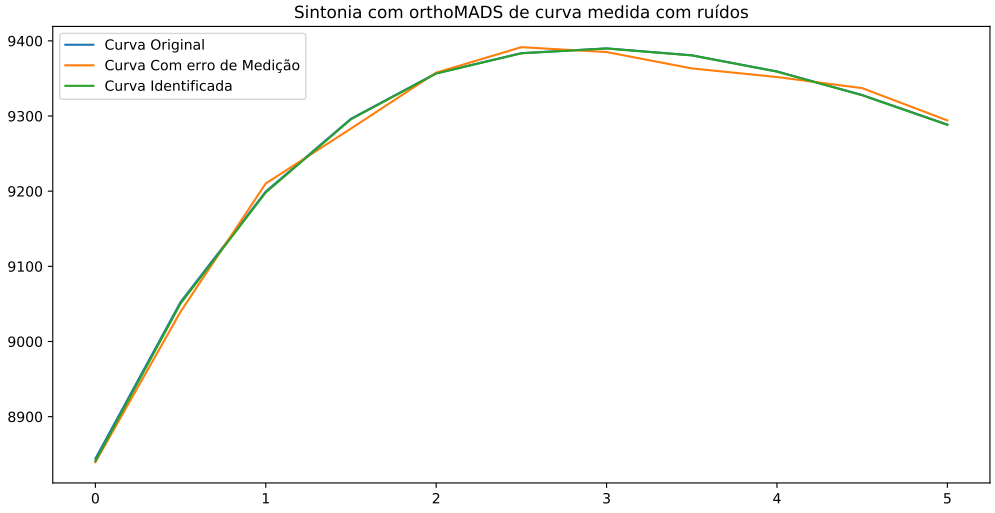
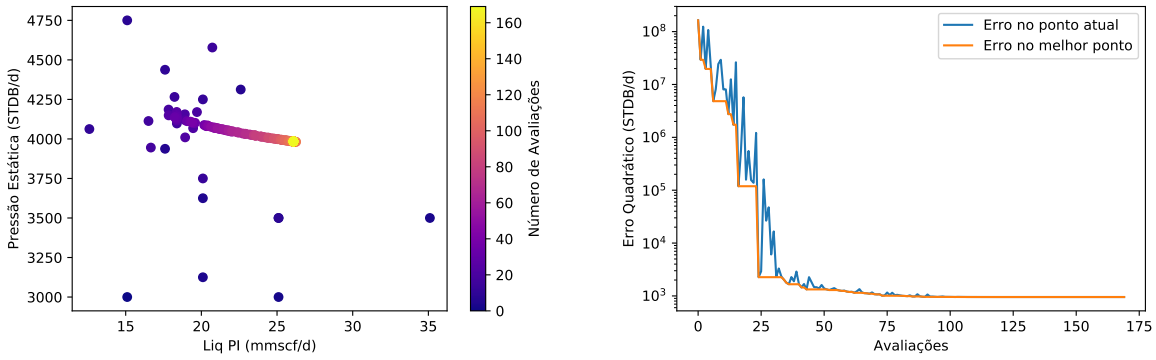


Figura 14 – Resultado da sintonia utilizando-se o OrthoMADS com a SGTELIB.

7.3 Sintonia de Curva com Ruído Com Nelder-Mead

Com a utilização do algoritmo de Nelder-Mead, em 217 segundos (3:27 minutos) e 92 avaliações, o algoritmo convergiu com uma uma função custo em 953.623. Como pode se ver na figura 15b, o algoritmo converge rapidamente no inicial, mas demora a adquirir a precisão requerida para a parada. Novamente a curva identificada está muito próxima da original sem ruídos.



(a) Pontos testados pelo Nelder-Mead.

(b) Erro nos pontos avaliados.

Figura 15 – Nelder-Mead aplicado ao experimento 2.

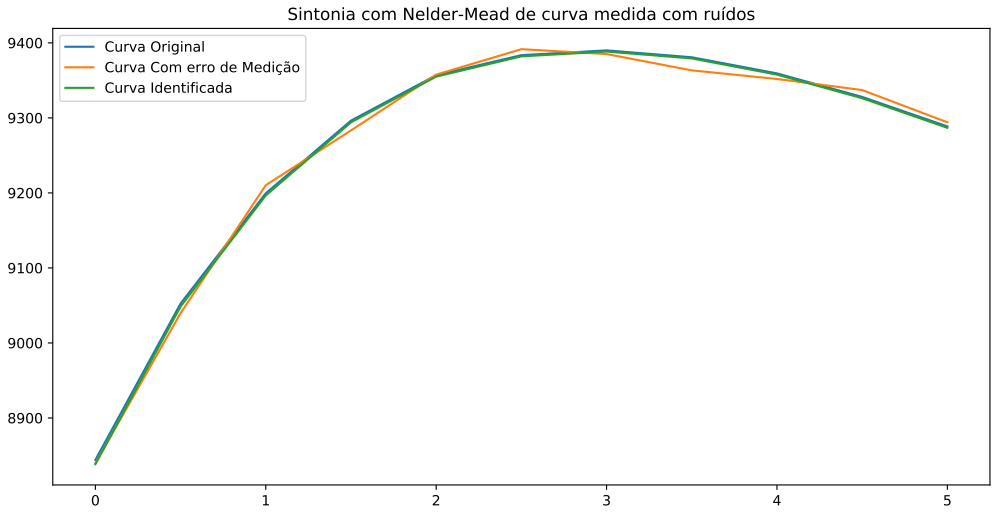


Figura 16 – Resultado da sintonia utilizando-se o Nelder-Mead.

7.4 Discussão

Nestes experimentos com ruído, não se repete o comportamento anômalo aonde o Nomad com SGTELIB avalia pontos em que aparentemente não há informações relevantes, aumentando sem necessidade o número de iterações e tempo computacional, como pode ser visto na Figura 17 e na Tabela 2. Pelo contrário, ele acaba sendo o mais eficiente dos três métodos. O Nelder-Mead, por sua vez, tornou-se relativamente mais lento.

Tabela 2 – Comparação dos métodos para sintonia de uma curva com ruído de medição

	Tempo	Iterações	Custo
OrthoMADS	334	130	956,827
OrthoMADS + SGTELIB	187	68	982,524
Nelder-Mead	217	92	953,522

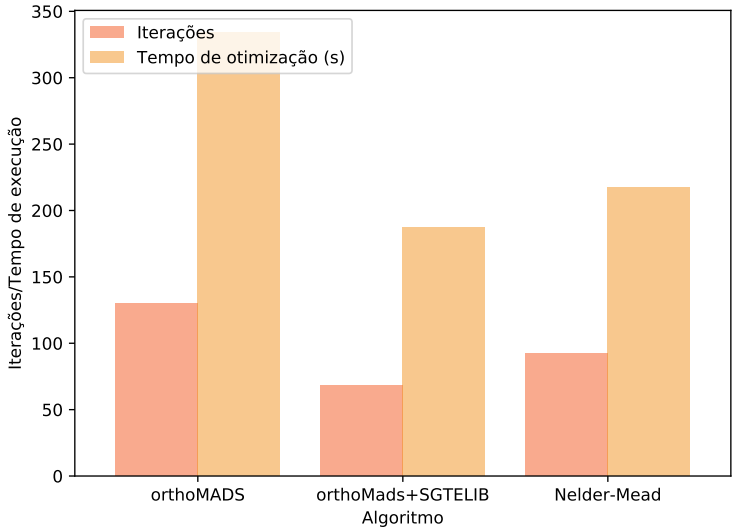


Figura 17 – Comparação entre tempos de execução e número de iterações entre os três experimentos com ruído.

8 Conclusões e Perspectivas

O principal objetivo deste trabalho, que foi o de utilizar métodos sem derivadas para sintonia de simuladores de escoamento multifásico de óleo e gás, foi alcançado. Foi possível desenvolver e utilizar software capazes de interagir com simuladores de fluxo multifásico para sintonizar modelos a dados experimentais por meio da manipulação dos parâmetros.

Desta forma foi mostrado que otimização sem derivadas é uma ferramenta que, dada uma função caixa-preta, é capaz de resolver problemas complexos, encontrando um ponto ótimo em tempo hábil sem necessidade da função explícita ou suas derivadas.

Para o futuro, é interessante explorar a opção do pyNOMAD, interface NOMAD para Python nativa, diminuindo a complexidade envolvida no Opal, e consequentemente simplificando os trabalhos.

Outro caminho não explorado é a imposição de restrições adicionais, dependentes do valor da saída do simulador, que podem ser adicionados ao NOMAD para sintonia de redes mais complexas.

Além disso, é interessante analisar o Opal, de forma a verificar o motivo do seu uso aparentemente excessivo de poder computacional, além de fazer experimentos com mais núcleos.

Referências

1 SINGER, A.; NELDER, J. Nelder-Mead algorithm. *Scholarpedia*, v. 4, n. 7, p. 2928, 2009. Citado na página 9.

2 ABRAMSON, M. A. et al. Orthomads: A deterministic MADS instance with orthogonal directions. *SIAM Journal on Optimization*, v. 20, n. 2, p. 948–966, 2009. Citado na página 9.

3 ABRAMSON, M. et al. *The NOMAD project*. Disponível em: <https://www.gerad.ca/nomad/>. Citado 2 vezes nas páginas 10 e 22.

4 DEVOLD, H. *Oil and gas production handbook*. [S.l.]: ABB Oil and Gas. Citado na página 11.

5 AUDET, C.; Dennis, Jr., J. Mesh adaptive direct search algorithms for constrained optimization. *SIAM Journal on Optimization*, v. 17, n. 1, p. 188–217, 2006. Citado na página 15.

6 JUPYTER. Disponível em: <http://jupyter.org/>. Citado na página 21.

7 MATPLOTLIB. Disponível em: <https://matplotlib.org/>. Citado na página 21.

8 THE History of Python. Disponível em: <http://python-history.blogspot.com.br/2009/01/brief-timeline-of-python.html>. Citado na página 21.

9 TIOBE. Disponível em: <http://www.tiobe.com/tiobe-index/>. Citado na página 21.

10 PRECHELT, L. An empirical comparison of seven programming languages. *IEEE Computer*, v. 33, n. 10, p. 23 – 29, 2000. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.113.1831&rep=rep1&type=pdf>. Citado na página 22.

11 GERAD. Disponível em: <https://www.gerad.ca/en>. Citado na página 22.

12 DANG, C. K. *Optimization of algorithms with the OPAL framework*. Tese (Doutorado) — ProQuest Dissertations Publishing, 2012. Citado na página 22.

13 PYTHON 2.7 Release Schedule. Disponível em: <http://legacy.python.org/dev/peps/pep-0373/>. Citado na página 22.

14 PYTHON3 Opal Port. Disponível em: <https://github.com/Williangalvani/opal/tree/python3>. Citado na página 22.