UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

ESCOLA DE ENGENHARIA

DEPARTAMENTO DE ENGENHARIA QUÍMICA

TRABALHO DE DIPLOMAÇÃO EM ENGENHARIA QUÍMICA

Reinforcement Learning aplicado para otimização da produção de poços de elevação de petróleo

Autor: Daniel Gabbardo Kucyk

Orientador: Marcelo Farenzena

Porto Alegre, maio de 2021

Autor: Daniel Gabbardo Kucyk

Reinforcement Learning aplicado para otimização da produção de poços de elevação de petróleo

Trabalho de Conclusão de Curso apresentado à COMGRAD/ENQ da Universidade Federal do Rio Grande do Sul como parte dos requisitos para a obtenção do título de Bacharel em Engenharia Química

Orientador: Marcelo Farenzena

Banca Examinadora:

Prof. Dr., Edson Cordeiro do Valle, DEQUI UFRGS

MSc., Fabio Cesar Diehl, Cenpes Petrobras

Porto Alegre

2021

AGRADECIMENTOS

Primeiramente gostaria de agradecer meu pai Paulo Kucyk e minha mãe Denise Gabbardo Kucyk, por incentivar meus estudos e apoiar minhas decisões, me dando confiança para encarar os desafios durante a realização desse trabalho.

Agradeço também meus familiares por participarem ativamente na minha formação pessoal e profissional.

Agradeço à UFRGS, essencial no meu processo de formação profissional e responsável por tudo que aprendi durante o curso. Agradeço também os professores, por compartilharem não só conhecimentos técnicos, mas também as experiências profissionais.

Em especial agradeço meu professor orientador, Marcelo Farenzena, por todo o apoio e dedicação durante a realização desse trabalho.

Agradeço os meus colegas de curso, por compartilharem comigo tantos momentos de descobertas e por todo o companheirismo ao longo desta trajetória.

Por fim agradeço de forma geral a todos aqueles que contribuíram, de alguma forma, para a realização deste trabalho.

RESUMO

A indústria petrolífera agrega grandes lucros a pequenos ganhos na produção, sendo de interesse otimizar o processo produtivo. Com o intuito de buscar novas formas para isso, o estudo explora a aplicabilidade do conceito de *reinforcement learning* (RL) em poços de elevação de petróleo. RL é um método de aprendizagem de máquina em crescente estudo e utilização na academia e indústria. RL possui um caráter exploratório, essa característica situacionalmente possibilita encontrar caminhos não ortodoxos que geram resultados excepcionais. Contudo, atualmente não há literatura envolvendo a utilização desse método para a otimização da produção de petróleo. Foi utilizado um método denominado *actor-critic* (AC) que, como todos os métodos de RL, envolve a interação de um agente no ambiente, com base em uma política, de modo a maximizar um sinal de recompensa. O método em questão atualiza continuamente a política com base nas recompensas obtidas, possibilitando explorar de maneira mais robusta a dinâmica do sistema de elevação de petróleo. Foi utilizado o modelo denominado *FOWM* para poços de petróleo como base para os testes de otimização. Nele atuou o método AC e, para comparação, dois métodos de otimização: um determinístico e local (*COBYLA*) e outro estocástico e global (*Differential Evolution*). O objetivo era o mesmo para todos: alcançar a maior produção, também entendido como a maior abertura da válvula *choke*, sem gerar golfadas. O método AC teve sucesso em gerar resultados viáveis, alcançando 72,8% de abertura de válvula, sem golfadas. Contudo, o tempo de execução é de aproximadamente 11,6 horas, demasiadamente elevado. *Differential Evolution* apresentou resultados inferiores para otimização do problema em questão, gerando resultados mais ágeis que os do método AC, porém menos eficazes. O método *COBYLA* apresentou o melhor resultado com o menor tempo de execução, alcançando 73,70% de abertura de válvula em aproximadamente 6 minutos. Entende-se que, de acordo com a configuração implementada neste estudo, não há motivos para escolher o método AC em detrimento ao método *COBYLA*. Os resultados também apontam uma não dependência do resultado do AC com o número de iterações, direcionando possíveis aprimoramentos para aumentar a efetividade desse algoritmo. Todavia, considera-se que o estudo obteve sucesso em implementar a metodologia de RL para poços de elevação de petróleo e que o tópico apresenta diversas oportunidades.

Palavras-chave: *reinforcement learning*; *actor-critic; machine learning; poços de elevação de petróleo; otimização de processo;*

ABSTRACT

The oil industry adds large profits to small gains in production, and it is of interest to optimize the production process. In order to seek new ways to do this, this study explores the applicability of the reinforcement learning (RL) concept in oil lift wells. RL is a learning method under increasing study and use in academia and industry. RL has an exploratory character, this characteristic situationally makes it possible to find unorthodox paths that yield exceptional results. However, there is currently no literature involving the use of this method for the optimization of oil production. A method called actor-critic (AC) was used which, like all RL methods, involves the interaction of an agent in the environment, based on a policy, in order to maximize a reward signal. The method in question continuously updates the policy based on the rewards obtained, making it possible to more robustly explore the dynamics of the oil lifting system. The so-called FOWM oil-well model was used as the basis for the optimization tests. The AC method was used and, for comparison, two optimization methods: one deterministic and local (COBYLA) and the other stochastic and global (Differential Evolution). The objective was the same for all: to reach the highest production, also understood as the largest choke opening, without generating slug flow. The AC method was successful in generating feasible results, achieving 72.8% valve opening, without slugging. However, the execution time is excessive, taking approximately 11.6 hours. Differential Evolution showed inferior results for optimization of the problem in question, generating more agile results than the AC method, but less effective. The COBYLA method presented the best result with the shortest execution time, reaching 73.70% of valve opening in approximately 6 minutes. It is understood that, according to the configuration implemented in this study, there is no reason to choose the AC method over the COBYLA method. The results also point to a non-dependence of the AC result with the number of iterations, guiding possible improvements to increase the effectiveness of this algorithm. Nevertheless, it is considered that the study was successful in implementing the RL methodology for oil lift wells and that the topic presents several opportunities.

Keywords: *reinforcement learning*; *actor-critic; machine learning; oil lift wells; process optimization;*

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Machine learning (ML) pode ser entendido como um sistema computacional que é capaz de aprender sem seguir instruções explícitas, utilizando algoritmos e modelos estatísticos para analisar e inferir padrões de uma base de dados. Com o avanço tecnológico e o crescente acesso à recursos computacionais avançados, ML foi evoluindo e dando origem a novos métodos que hoje compõem as diversas vertentes de aplicação. Trabalhos como o de Sun et al. (2020) para identificação de estados de conformação de polímeros, o de Barman et al. (2019) para detecção de genes hospedeiros de doenças infecciosas e o de J. Yu et al. (2014) para classificação de imagens, são exemplos dos múltiplos usos de ML. Esses usos podem ser enquadrados em três grupos de aprendizagem: supervisionada, não supervisionada e aprendizagem por reforço. *Reinforcement learning* (RL), ou aprendizado por reforço, similarmente ao método não supervisionado, é um método que não utiliza uma base de dados expressiva, porém, não é objetivo do RL classificar a informação, mas sim maximizar um sinal de recompensa recebido (Sutton & Barto, 2017).

RL é, como mencionado, um método de aprendizagem que busca receber a maior recompensa. Para isso um agente realiza ações em um ambiente, com base no estado atual, recebendo uma recompensa por essa ação, de acordo com o próximo estado alcançado. Na prática, RL explora as ações a serem tomadas em cada estado com o intuito de encontrar trajetórias ótimas, mantendo o objetivo de incrementar ao máximo as recompensas acumuladas. A peculiaridade desse método de aprendizagem é ser um método “esforçado”. Em outras palavras, RL vai, em vezes, tomar decisões não otimizadas com o intuito de explorar possíveis rotas que, a longo prazo, forneçam maiores recompensas (Sutton & Barto, 2017).

Em vista desse caráter exploratório, entende-se que esse método não será implementado diretamente em uma planta até, ou se achar um ponto ótimo ou, possivelmente, causar danos. Ao invés disso, RL é utilizado em conjunto com um modelo do sistema. Na engenharia química, esse método de aprendizado recentemente vem sendo aplicado em diversos setores, a exemplo de métodos de produção em batelada (Yoo et al., 2021), secagem de tabaco (Bi et al., 2020) e processo de hidrocraqueamento (Oh et al., 2021).

Um dos possíveis usos de RL é a área de exploração de petróleo. A indústria petroquímica como um todo move grandes valores de capital, o relatório publicado

pela agência nacional do petróleo, gás natural e combustíveis (ANP) em 2018, aponta que o setor de petróleo gás responde por 13% do PIB brasileiro (Décio et al., 2018). Logo, pequenos incrementos na produção podem acarretar grandes ganhos. O desafio está nos problemas operacionais, principalmente nas plataformas em alto mar. Por se tratar de um escoamento multifásico, quando o transporte dos fluidos apresenta um padrão de escoamento estratificado, esse cenário pode ocasionar golfadas severas, fenômeno prejudicial tanto à produção, quanto à segurança do processo. Otimizar a produção pode ser entendido como atuar nas variáveis do processo de elevação, sendo a principal delas a abertura da válvula *choke* no topo da plataforma, responsável por regular a vazão de óleo e também utilizada para controlar golfadas severas (Jahanshahi, 2013). Poucos trabalhos abordam diretamente o conceito de elevar a quantidade de óleo produzida e esses sempre possuem um caráter determinístico, a exemplo de Foss et al. (2018). Encontrar o ponto ótimo de operação da *choke* não é direto, visto que esse frequentemente se encontra acima do ponto de Hopf, ou seja, fora da zona de estabilidade do sistema. Caso fosse considerado somente uma instância, ou “tiro”, o otimizador finalizaria sua execução assim que encontrado Hopf. Contudo, é possível alcançar valores maiores, sem golfar, se considerada uma trajetória de ações, sem a presença de ciclo limite. Essa característica incentiva o uso de RL e corrobora à consideração desse sistema como um problema dinâmico.

Na literatura não há estudos referentes à otimização do sistema de elevação utilizando RL. Possivelmente, à primeira vista, essa metodologia não aparenta atrativa para esses sistemas, visto que não será utilizada diretamente na linha de produção. Todavia, RL pode guiar as tomadas de decisões reais com base nas trajetórias obtidas, servindo como um direcionador para otimizar o processo. Foi implementado no *software Python* um método de RL denominado *actor-critic* (AC) e se testou diferentes configurações de parâmetros para avaliar as melhores respostas. Tal metodologia foi comparada com a otimização dinâmica do sistema, utilizando otimizadores disponíveis atualmente no pacote *scipy*: *COBYLA* e *differential evolution*, sendo o primeiro determinístico e local e o segundo estocástico e global. Objetiva-se, principalmente, gerar resultados otimizados utilizando o método AC. Para isso, o primeiro passo será garantir que os resultados não apresentem golfadas severas. Além disso, é desejado que a trajetória apresentada seja plausível, ou seja, evite

movimentos súbitos na válvula. Procura-se também explorar possíveis aprimoramentos a serem feitos para aumentar sua efetividade.

O presente trabalho é dividido como segue. No capítulo 2 é feito uma revisão dos trabalhos já publicados referentes aos conceitos aplicados nesse estudo, dentre eles, métodos de aprendizagem, RL e poços de elevação de petróleo. O capítulo 3 detalha a metodologia aplicada, desde a formulação do modelo até as linhas de código do RL e dos outros otimizadores. Os resultados e comparações são explicitados no capítulo 4. Por fim, o capítulo 5 expõem a conclusão do estudo realizado, bem como direciona possíveis trabalhos futuros sobre o tema.

2 REVISÃO BIBLIOGRÁFICA

2.1 Métodos de Aprendizado

Estudos envolvendo inteligência artificial vem ganhando maior atenção com o passar dos anos. No site Scopus (https://www.scopus.com/, s.d.), filtradas pelo termo *machine learning* na subárea de engenharia química, há um incremento de 94 publicações em 2015 para 1158 em 2020. As aplicações são diversas, de produção de comida fermentada (Galimberti et al., 2021) à análise de tensão superficial de gotas a partir de imagens (Soori et al., 2021). Os métodos de aprendizado são classificados em três grupos, supervisionados, não supervisionados e por esforço (Ayodele, 2010).

Os métodos supervisionados abrangem diversas técnicas de aprendizado, a exemplo de algoritmos lógicos, redes neurais, algoritmos de aprendizado estatístico e máquinas de suporte vetorial. Todo cenário em qualquer base de dados usada em algoritmos de *machine learning* é representado por um mesmo grupo de recursos. Se esses cenários são fornecidos com parâmetros conhecidos, então o método é chamado de supervisionado (Kotsiantis, 2007). Em outras palavras, os dados utilizados para treinar esses modelos contém a resposta desejada, ou seja, contém a variável dependente resultante das variáveis independentes observadas. Bagherzadeh (2021) faz uso de três diferentes técnicas no seu estudo comparativo de predição de nitrogênio total em plantas de tratamento de esgoto, dois algoritmos lógicos (árvore de decisões) e uma rede neural. Redes neurais também foram aplicadas para determinar o teor de adulteração na gasolina com base em curvas de destilação (Foroughi et al., 2021).

Apesar de atualmente bem estabelecido, a taxa de sucesso nas aplicações do método supervisionado está atrelada à existência de uma base de dados informativa. Quando não é possível gerar um *dataset* refinado, normalmente se utiliza um levantamento geral e a partir desse, espera-se ser possível isolar as informações importantes. Há uma hierarquia de problemas que são frequentemente encontrados no preparo dos dados: Valores impossíveis ou improváveis foram computados, há valores faltando, atributos irrelevantes estão computados. (Kotsiantis, 2007).

Quando obter uma base de dados é custoso ou impossível, explora-se então os métodos não supervisionados. Uma das maiores diferenças dos métodos não supervisionados é que não há um conjunto para treino, logo não é um método direto para uma validação cruzada (Hahne et al., 2008). Outra peculiaridade desse método é que, apesar dos algoritmos buscarem critérios otimizados, não há garantia que uma solução global foi encontrada. Deve se considerar todas as combinações dos dados, logo, mesmo para uma amostra moderada, alguma abordagem heurística é utilizada. (Hahne et al., 2008)

As técnicas de aprendizado não supervisionado envolvem deixar o modelo em questão detectar padrões e informações relevantes. São elas: agrupamento hierárquico, particionamento e métodos híbridos. Agrupamentos hierárquicos podem ainda ser expandidos em aglomerantes ou divisíveis. Trabalhos envolvendo esse tipo de aprendizado incluem a identificação de estados de conformação de polímeros (Sun et al., 2020), design de catalisadores ativos para acoplamento oxidativo em baixas temperaturas (Ohyama et al., 2021), desenvolvimento de novo índice de risco de combustão para líquidos inflamáveis com base em algoritmos de agrupamento (Ji et al., 2021), dentre outros.

A terceira abordagem é a aprendizagem por reforço. O conceito derivou da psicologia onde o método, com o mesmo nome, impõem a um agente uma recompensa ou punição em resposta a uma ação tomada. Traduzindo a lógica para máquina, tem-se um estado atual e possíveis ações que levarão à estados futuros. Escolhe-se uma dessas ações e se obtém a recompensa atrelada a essa ação com base no novo estado alcançado. Com repetições, espera-se que o agente aprenda as ações que induzem ao maior número de recompensas positivas e passe a evitar ações punitivas. Aprendizado por reforço é diferente de aprendizagem não supervisionada. Aprendizado por reforço tenta maximizar um sinal de recompensa ao invés de tentar encontrar padrões escondidos (Sutton & Barto, 2017).

Os métodos de aplicação da metodologia do reforço são diversos, porém todos seguem o mesmo objetivo exposto anteriormente. As diferentes técnicas se fazem presentes para adequar o tipo de recompensa ao agente e como esse deve interagir com o ambiente. A vasta gama de estudos inclui, por exemplo, controle autônomo de pH para esgoto industrial de galvanoplastia (Alves Goulart & Dutra Pereira, 2020), controle ativo de vazão em condições turbulentas (Ren et al., 2020), otimização de conforto interno, qualidade do ambiente e gasto de energia em salas de aula (K. H. Yu et al., 2021), além de outros títulos em uma amplitude aplicações.

2.2 Reinforcement Learning

Reinforcement learning (RL) ou, como citado anteriormente, aprendizado por reforço, é um método de *machine learning* (ML) que atualmente está recebendo uma nova atenção. Sutton & Barto (2017) entendem que só uma pequena parcela do método de aprendizado por reforço foi estudada nos primórdios da inteligência artificial e há ainda muito a ser revelado nesse ramo de ML. Problemas de RL envolvem aprender a ação a se tomar, com o objetivo de maximizar um sinal denominado de recompensa. Pode-se entender o processo como um sistema fechado, onde o agente realiza uma ação que leva do estado original para um estado futuro, devolvendo ao agente a recompensa dessa escolha e a partir do novo estado, o agente avalia uma nova decisão. Esse processo se repete até o agente aprender as melhores decisões a serem tomadas.

A aprendizagem pode ser realizada pelos denominados métodos sem modelo, caracterizados como métodos de tentativa e erro, assim como podem empregar um modelo do ambiente para descrever o estado atual e possíveis estados futuros. O modelo necessita atender as limitações do ambiente e fornecer informações para serem interpretadas pelo agente. Independentemente do método, o ambiente deve ser passivo de alteração pelo agente e responder de acordo. Um agente não é necessariamente todo um robô ou organismo e o seu ambiente não é necessariamente somente o que está fora de tal robô ou organismo (Sutton & Barto, 2017).

Outros conceitos importantes quando se trata de RL são políticas e funções valor. Pode-se entender uma política como o método do agente agir dado certo tempo, em outras palavras, é um levantamento dos estados e das ações a serem tomadas quando em tais estados. O agente implementa um mapeamento dos estados com

base na probabilidade de selecionar cada ação. Esse mapeamento é denominado de política do agente (Sutton & Barto, 2017). A política é frequentemente vista como o coração do agente, visto que, ela sozinha é suficiente para determinar comportamento.

Utilizando *blackjack* ou “21” como exemplo, pode-se entender como estado a pontuação atual das cartas do jogador. As ações são somente duas: comprar uma carta ou manter as atuais. O objetivo é alcançar uma pontuação maior que a da banca, porém sem ultrapassar vinte e um. Se em um momento a pontuação do jogador é dezoito, uma política pode levar o jogador a comprar mais uma carta para tentar chegar mais próximo de vinte e um, assim como outra pode optar por não arriscar estourar a pontuação. Os métodos de *reinforcement learning* especificam como um agente muda sua política com base na experiência (Sutton & Barto, 2017).

Funções valor estão presente em quase todos os métodos de RL e são funções de estado para estimar o quão benéfico é para o agente estar naquele estado. A noção de valor de um estado é definida com base em recompensas futuras esperadas, ou, mais precisamente, ganhos esperados (Sutton & Barto, 2017). Como mencionado, as recompensas dependem das ações tomadas que estão sujeitas às políticas aplicadas. Logo, as funções valor são definidas respeitando as políticas particulares de cada caso.

Quando se trata da ação a ser tomada, com base na política, o agente pode encarar a situação de duas formas, uma delas gananciosa e a outra exploratória. Um agente ganancioso é definido por aquele que sempre busca a ação que oferece a maior recompensa instantânea, sem considerar possíveis ganhos a longo prazo. Por outro lado, um caráter exploratório direciona a decisão para ações que, apesar de no momento não serem ótimas, podem elevar a recompensa total a longo prazo, porém não há garantia disso. Sutton (2017) entende que *reinforcement learning* necessita de um balanço entre esses dois comportamentos.

A primeira metodologia analisada são os métodos de soluções tabulares. Esses aplicam os algoritmos de aprendizado por reforço de maneira simplista, de modo que o estado e espaço de ações podem ser definidos por um vetor ou uma tabela (Sutton & Barto, 2017). Na maioria das vezes, esses métodos encontram soluções, funções valor e políticas otimizadas. Um exemplo de método de solução tabular é o de ação-valor, que consiste em avaliar o valor de cada ação e usar essa estimativa para escolher a ação a ser tomada. A avaliação é feita utilizando a média aritmética das

recompensar obtidas até o momento. Se até o momento *t*, a ação *a* foi escolhida $N_t(a)$ vezes antes de *t*, com respectivas recompensas R_1, R_2, R_3, ..., $R_{Nt(a)}$ então seu valor é estimado em

$$Q_t(a) = \frac{R_1 + R_2 + R_3 + \cdots + R_{N_t(a)}}{N_t(a)}. \quad (2.1)$$

Uma política totalmente gananciosa seleciona a ação com base no maior valor estimado, ou seja, não usa tempo experimentando ações com valores menores para avaliar se elas possivelmente podem ser melhores. O termo ganancioso é usado em ciência computacional para descrever qualquer tomada de decisão que seleciona alternativas baseado somente em considerações locais ou imediatas (Sutton & Barto, 2017). Na maioria das vezes se opta por utilizar uma política denominada ***ε-greedy*** onde, com uma probabilidade ***ε,*** a tomada de decisão é totalmente randômica, ou seja, desconsidera o valor de cada ação. Os métodos não gananciosos apresentaram maior recompensa média ao final, visto que, a longo prazo, o método ganancioso se prende a decisões não otimizadas.

Outro método tabular é o processo de decisão de *Markov*, entendido como a definição do campo de RL. Qualquer método que é apropriado para resolver esse problema, é considerado um método de *reinforcement learning* (Sutton & Barto, 2017). O método faz uso da interface agente-ambiente, conforme ilustrado na Figura 2.1, para estabelecer probabilidades de se tomar uma ação dada resposta gerada pelo ambiente. A cada etapa *t,* o agente recebe uma informação do estado S_t e com base nisso seleciona a ação A_t disponível no conjunto de ações do estado atual. Na etapa seguinte, em consequência da etapa anterior, o agente recebe uma resposta no formato de uma recompensa numérica $R_{t+1.}$ O agente então mapeia as possíveis ações com base na política.

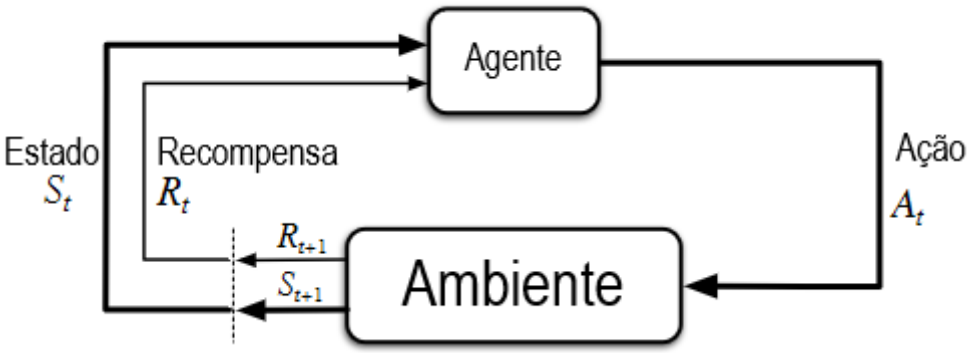

Figura 2.1: Interação agente-ambiente em aprendizado por esforço.
Fonte: Sutton & Barto, 2017

Se um sinal do estado conseguir reter toda a informação relevante, é dito que o problema possui a propriedade de *Markov.* Essa propriedade simplifica o cálculo da probabilidade de se tomar uma ação, visto que só o estado mais recente precisa ser analisado, já que este vai conter todos os pontos importantes do problema. Apesar do processo de decisão de *Markov* ser uma base importante para a compreensão dos modelos modernos de RL, nem sempre a propriedade de *Markov* se faz presente. Outros métodos englobados nas soluções tabulares são: programação dinâmica, método de Monte Carlo, *temporal-difference learning* e traços de elegibilidade. O estudo realizado por Li & Misra (2021) utiliza um método tabular para aprimoramento da predição do comportamento de reservatórios de petróleo. Yoo et al. (2021) também faz uso de uma solução tabular no seu estudo para aprimorar o controle de processos em batelada, no caso foi utilizado um processo de polimerização produzindo polióis para demonstração.

Para aplicação de métodos tabulares, as funções devem ser expressas com uma entrada para cada estado ou para cada par estado-ação. Isso limita a aplicação somente a problemas com um pequeno número de estados e ações. Em diversos problemas onde seria desejável aplicar RL, a maioria dos estados encontrados nunca foram experienciados antes. O único jeito de se aprender algo nesses problemas é fazer uma generalização dos estados experienciados anteriormente para aplicar nos que nunca foram vistos (Sutton & Barto, 2017). A generalização mais comumente utilizada é a chamada *function approximation*, que está no escopo de aprendizado supervisionado. A fusão desse princípio de aprendizagem supervisionada com os conceitos de aprendizagem por reforço dá origem ao grupo denominado métodos de solução aproximada. Os métodos de solução aproximada envolvem: aproximação de

valores de ação conforme política, aproximação de valores de ação sem política e aproximação de política, além de seus diversos subgrupos. Optou-se por dar preferência ao método de aproximação de política, em especial o método *actor-critic*, utilizado neste estudo.

2.2.1 Método *Actor-Critic*

O método *actor-critic* (AC) vem da combinação dos princípios dos métodos *actor-only* (AO) e *critic-only* (CO), objetivando conquistar as vantagens de ambos. Métodos *critic-only* usam uma função valor de estado-ação e nenhuma função explicita para a política (Grondman et al., 2012). Quando CO é aplicado para espaços e ações contínuas, essa função valor vai ser somente uma aproximação. O método calcula uma política determinística com base em um procedimento de otimização sobre a função valor, contudo, mesmo com otimizadores de função específicos, esse método não apresentara convergência para problemas simples de decisão de *Markov* (Grondman et al., 2012). A convergência é garantida para o caso de otimizadores de função com parâmetros lineares, se as trajetórias das ações forem amostradas de acordo com sua distribuição seguindo a política.

Actor-only por outro lado não utilizam nenhuma forma de armazenamento de função valor. A grande maioria dos algoritmos *actor-only* trabalham com uma família de políticas parametrizadas e otimizam a recompensa [...] diretamente sobre o espaço-parâmetro da política (Grondman et al., 2012). Métodos AO utilizam uma parametrização da política, diferenciando a mesma com base no vetor de parâmetros. A partir dessa diferenciação, aplicam-se técnicas de otimização para encontrar uma solução ótima local para a recompensa. A maior vantagem de métodos AO em relação a métodos CO é que eles permitem a política gerar ações em um espaço ação contínuo. Outra vantagem é a forte tendência a conversão apresentada por esse método. Conversão é alcançada quando os gradientes estimados são imparciais e a taxa de aprendizagem se adequa a equação da recompensa. Uma desvantagem do método *actor-only* é que o gradiente estimado pode apresentar grande variância. Além disso, todo gradiente é calculado sem usar informações de estimativas passadas (Grondman et al., 2012).

O método *actor-critic* é capaz de produzir ações contínuas como o método AO, mas limita a variância das políticas adicionando um crítico. O papel desse crítico é avaliar a atual política escrita por esse ator. Essa avaliação pode ser feita utilizando

qualquer tipo de método comum de avaliação de política, a exemplo de *temporal-difference with decay* (TD(λ)), *least square temporal-difference* (LSTD) ou gradientes residuais. O crítico aproxima e atualiza a função valor utilizando amostras e essa é usada para atualizar os parâmetros de política do ator para melhorar a performance, vide Figura 2.2. *TD error* é a função utilizada para atualizar o crítico.

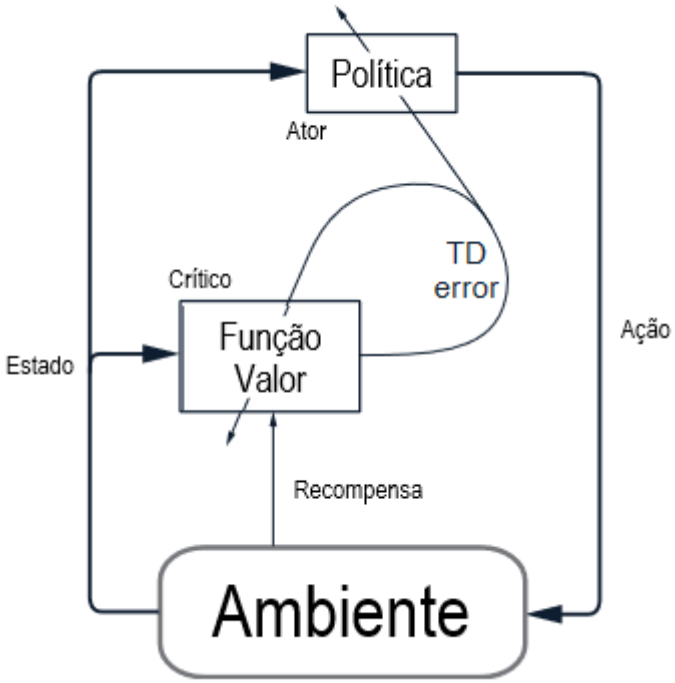

Figura 2.2: Arquitetura do modelo *actor-critic*.
Fonte: Sutton & Barto, 2017

O objetivo em métodos AC é achar a melhor política, dado um processo de decisão de *Markov* estacionário, ou seja, o espaço de estados e ações, a densidade dos estados de transição e a função recompensa, são constantes. Um pré-requisito é que o crítico consiga avaliar com acurácia dada política, ou seja, o objetivo do crítico é achar uma solução aproximada para a equação de *Bellman* para dada política. A equação de *Bellman* é genericamente representada por

$$\upsilon_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma \upsilon_\pi(s')]. \qquad (2.2)$$

onde $v_\pi(s)$ é o valor do estado dada política π, $\pi(a|s)$ é a probabilidade de se tomar a ação *a* quando no estado *s*, $p(s',r|s,a)$ é a probabilidade de alcançar o estado *s'* com recompensa *r*, partindo do estado *s* com ação *a* e γ é o fator de desconto. O termo *temporal-difference* (TD) *error* deriva dessa equação e é utilizado para atualizar o

crítico. A partir de *Bellman* e utilizando uma amostra de transição, a função erro TD é estimada como:

$$\delta_t = R_{t+1} + \gamma V_t(S_{t+1}) - V(S_t). \quad (2.3)$$

onde δ_t é termo do erro TD no momento *t,* V_t é a função valor implementada pelo crítico no momento *t* e *S* define um estado. De modo geral a equação, caso positiva, implica que a tendência de escolher a ação a_t deve ser incrementada, quando no caso negativo essa tendência deve ser reduzida (Grondman et al., 2012). Bi et al. (2020) utilizam *actor-critic* na otimização do processo de secagem do tabaco, similarmente, Oh et al. (2021) aplica o mesmo para estimar as condições de operação otimizadas no processo de hidrocraqueamento.

2.3 Sistema de Elevação de Petróleo

Parte da produção de petróleo atualmente acontece em plataformas de extração em alto mar. Um sistema de elevação de petróleo pode ser subdividido em reservatório, poço, linha de transporte, linha de elevação (*riser*) e plataforma, vide Figura 2.3. Para controle do processo, no topo da plataforma, fazendo a ligação com a linha de elevação há uma válvula *choke* (Di Meglio et al., 2012).

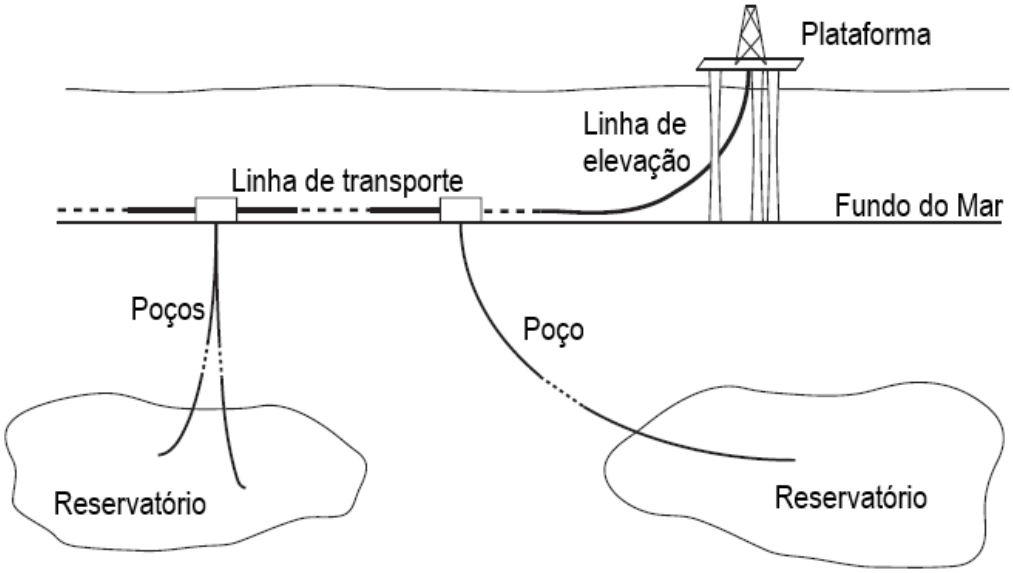

Figura 2.3: Esquema de um poço de extração de petróleo em alto mar
Fonte: Di Meglio et al., 2012

Durante o processo de produção, gás é injetado no poço para facilitar o transporte do petróleo, frente à grande energia requerida para elevação, no caso de reservatórios mais profundos. Petróleo, gás e água são transportados

simultaneamente do reservatório até a superfície por tubulações que interligam a plataforma até o fundo do mar, variando de 100 m a mais de 1 km de comprimento. O escoamento multifásico pode apresentar um perfil denominado *slugging,* que é um regime intermitente bifásico (ou trifásico) caracterizado por uma distribuição não homogênea entre a fase líquida e gasosa (Di Meglio et al., 2012). Esse regime possibilita um padrão de escoamento estratificado, podendo gerar o fenômeno denominado golfada severa.

A golfada severa acontece quando longas bolhas de gás fluem pela tubulação, separadas por acúmulos de líquido. Isso resulta em longos períodos de baixa produção líquida e, periodicamente, picos de produção. Um ângulo de inclinação acentuado entre o final da linha de transporte e a linha de elevação aumentam a probabilidade de *slugging*, visto que possibilita o acúmulo de líquido na entrada do *riser,* bloqueando a passagem de gás. Isso acarreta uma compressão do gás na linha de transporte, até a pressão da bolha ultrapasse a pressão hidrostática na linha de elevação. A Figura 2.4 representa um ciclo de golfada severa. O momento (1) demonstra a formação da bolha de ar, devido a uma pressão menor quando comparada à coluna de líquido. Durante (2) a bolha continua acumulando gás, aumentando a pressão dela enquanto somente líquido é elevado. Essa produção durante *slugging* é menor quando comparada a operação normal, pois essa elevação é ocasionada somente pela pressão estática do líquido. Em (3), a pressão da bolha é suficiente para ejetar a coluna de líquido, gerando a golfada severa, ou seja, um súbito pico na produção. Após, há uma queda na produção (4), líquido começa a se acumular novamente e outra bolha é formada, fechando o ciclo.

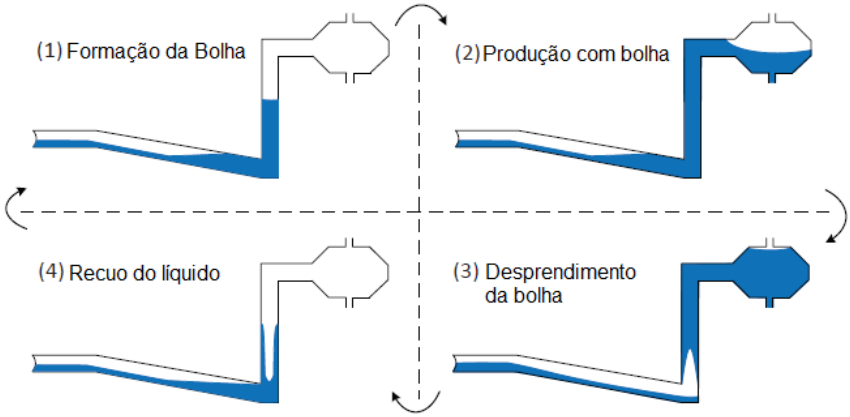

Figura 2.4: Esquema de golfada severa em sistemas de transporte e elevação.
Fonte: Jahanshahi, 2013

Uma solução estudada para evitar esse fenômeno foi um sistema de controle automático para atuar nas válvulas *choke* de modo a levar o regime para um estado sem golfar. Jahanshahi (2013) propôs um método mais robusto para realização do controle anti-*slug*, avaliando a eficácia do controle em pontos diferentes e explicitando quais são suscetíveis para o controle de golfadas.

No caso do sistema de *gas-lift,* o melhor método de controle depende da escala das variáveis controladas. Considerando a abertura da válvula *choke* como variável de controle, a vazão mássica como variável monitorada apresenta maiores ganhos do sistema e as estruturas de controle apresentaram maior performance quando comparadas a pressão como variável monitorada (Jahanshahi, 2013). Contudo, para controlar a vazão com restrições rígidas é necessário um sistema de medição preciso, o que é pouco viável para um sistema multifásico. Pressão, por outro lado, pode ser medida mais facilmente, podendo ser implementado com um fator de escala menor. Logo, as estruturas de controle usando pressão demonstram ser superiores no estudo de caso de *gas-lift* (Jahanshahi, 2013).

A otimização de sistemas *off-shore* se dá pela atuação na válvula *choke*, nos parâmetros de elevação artificial e nas rotas de corrente de poços. O estudo realizado por Foss et al. (2018) defende que uma otimização estática é suficiente para a grande maioria de problemas de otimização de produção diária. Opta-se por uma otimização estática, quando possível, devido à acurácia dos modelos ser um desafio, dada necessidade de medição de vazão para escoamento multifásico e esses medidores serem poucos. Por outro lado, uma abordagem dinâmica também pode ser aplicada quando o modelo é suficientemente adequado e autores como Di Meglio et al. (2012) e Jahanshahi (2013) utilizaram essa consideração nos seus estudos. O objetivo é maximizar a corrente de óleo, minimizando o consumo de gás e evitando golfadas, visto que essas impactam negativamente na produção. O desafio é operar o poço em aberturas de válvula superiores ao ponto de Hopf, sem gerar *slugging*. O ponto de Hopf é a abertura da *choke* que define onde o sistema troca de estabilidade, variando conforme as condições do poço. Em outras palavras, a otimização procura políticas de operação na região instável de abertura de válvula, sem ocasionar golfadas.

3 MATERIAIS E MÉTODOS

Para elaboração do trabalho foi implementado no software *Python* um modelo de poço de elevação de petróleo, criado um algoritmo *actor-critic* de aprendizado por reforço para tal sistema e aplicado algoritmos de otimização já existentes para comparação dos resultados. A seguir será exposto cada etapa mencionada, bem como considerações realizadas.

3.1 Modelo Simplificado Poço de Petróleo

É inerente a necessidade de um modelo computacional adequado para avaliação dos métodos de otimização propostos. Para tal, foi utilizado como base o trabalho de Diehl et al. (2017) que propõem um modelo para poços de elevação de petróleo dinâmico e robusto denominado *Fast Offshore Wells Model* (FOWM) (cuja descrição pode ser encontrada no Apêndice A). Diehl et al. (2017) integrou modelos publicados e validados para cada etapa do sistema de extração de petróleo e obteve um que é não linear e engloba: poço, anular de *gas-lift*, linha de produção, *riser*, estimação da pressão no instrumento do fundo da coluna de produção (PDG) e no transdutor de temperatura e pressão (TPT). Tal modelo representa *casing heading, slugging* gerado por inclinação do terreno e da linha de elevação, além de ser validado com dados industriais.

Para aplicação da metodologia, foram utilizados os dados de poços reais fornecidos no trabalho de Diehl et al. (2017), vide Tabela 3.1, além dos parâmetros referentes ao mesmo poço, vide Tabela 3.2. Optou-se por utilizar os dados do poço real A, porém tal escolha foi realizada de maneira imparcial, dada indiferença da escala do poço para aplicação dos métodos de otimização.

Tabela 3.1: Dados poço real A.

Variável	Poço A
ρ_L (kg/m^3)	900
P_r (bar)	225
P_s (bar)	10
α_{gw}	0,0188
P_{mres} (kg/m^3)	892
M (kg/kmol)	18
T (K)	298
L_r (m)	1569

L_{fl} (m)	2928
L_t (m)	1639
L_a (m)	1118
H_t (m)	1279
H_{pdg} (m)	1117
H_{vgl} (m)	916
D_{ss} (m)	0,15
D_t (m)	0,15
D_a (m)	0,14

Fonte: Diehl et al. (2017)

Tabela 3.2: Parâmetros poço real A.

Parâmetro	Caso 3 - Planta Real - Poço A
$m_{L,still}$	$6{,}222e^{+1}$
C_g	$1{,}137e^{-3}$
C_{out}	$2{,}039e^{-3}$
V_{eb}	$6{,}098e^{+1}$
E	$1{,}545e^{-1}$
K_w	$6{,}876e^{-4}$
K_a	$2{,}293e^{-5}$
K_r	$1{,}269e^{+2}$
ω_u	$2{,}780e^{0}$

Fonte: Diehl et al. (2017)

A injeção de gás W_{gc} e a inclinação do *riser* Θ foram consideradas constantes em, respectivamente, 1,0 kg/s e 0,577 rad. Para essas condições foi avaliado o ponto de Hopf desse sistema em, aproximadamente, 54% de abertura da válvula *choke*, vide Figura 3.1. O ponto de Hopf é o momento onde há a bifurcação do gráfico, ou seja, onde o sistema passa de um estado assintótico para um ciclo limite. Quanto maior a diferença entre o máximo e o mínimo de pressão no ponto PDG, maior a severidade da golfada.

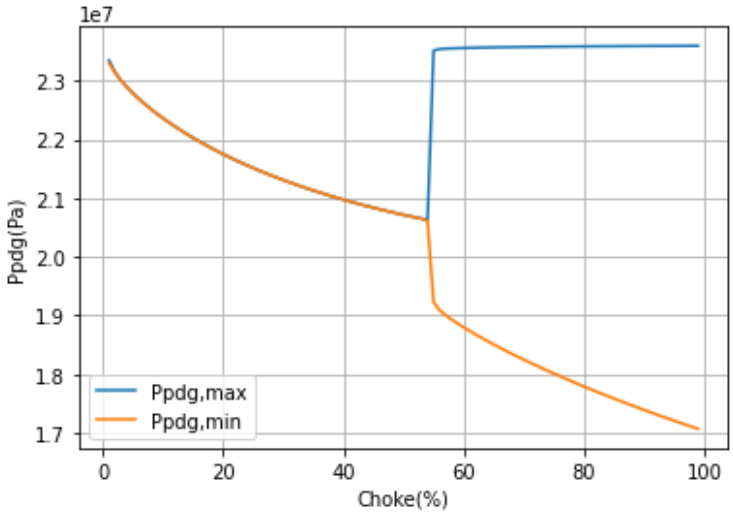

Figura 3.1: Diagrama de bifurcação com base na pressão no ponto PDG.

3.2 *Reinforcement Learning*

Como todos os problemas de *reinforcement learning* (RL), primeiramente é necessário definir o ambiente, os estados, as ações, as políticas e as recompensas para o sistema em questão. Após definidos esses pontos, escolhe-se o algoritmo de RL a ser utilizado, visto que as definições do sistema limitam a utilização de determinados métodos.

O ambiente no presente problema está majoritariamente definido no modelo empregado. Os estados desse ambiente foram definidos como os equilíbrios alcançados para cada abertura da *choke.* A abertura da válvula é um ponto importante, pois, além de definir o estado atual do ambiente, também é nela onde as ações atuam. Abaixo de 2% de abertura de válvula, o integrador para resolução do modelo apresenta problemas, além de, na prática, ser um valor inviável, pois fecharia o poço. Limitou-se então o mínimo e o máximo de abertura da *choke* em, respectivamente, 2% e 98%. O limite superior em 98% foi decidido arbitrariamente, com o intuito de evitar cenários de saturação de válvula e gerar possíveis riscos ao processo. Numericamente, dividiram-se os possíveis estados do ambiente em um plano de duas dimensões, em um eixo a abertura de válvula e no outro o valor da pressão no ponto PDG. Essa consideração objetiva localizar melhor o agente no ambiente, de modo a não sobrepor estados, afetando a avaliação das ações a serem realizadas. Optou-se pela pressão no ponto PDG, pois essa variável não está atrelada a abertura de válvula, ou seja, uma mesma abertura de válvula pode acarretar diferentes pressões no ponto PDG dependendo da trajetória executada.

Como mencionado anteriormente, a ação é atuar na abertura da válvula *choke.* Diehl et al. (2019) no seu estudo para incremento de produção em um poço ultra profundo da Petrobrás, comenta que a velocidade lenta do atuador dessa válvula faz com que essa demore 6 minutos para abrir totalmente. Levando essa dinâmica em consideração, optou-se por restringir os movimentos em, no máximo, 17% de variação absoluta por minuto. Como o método avalia todas as ações a serem realizadas, foi utilizado um conjunto discreto de ações, ou seja, movimentos de -17% a 17% de abertura com intervalos de 0,2%, incluindo o movimento zero, totalizando 171 possíveis ações. Tais valores foram escolhidos com o intuito de permitir adequada liberdade do algoritmo, sem um peso computacional excessivo.

A política pode ser entendida como a probabilidade de se tomar uma ação em determinado estado. Seria possível definir ela manualmente, contudo, no problema em questão, o algoritmo do método de RL selecionado define e realiza constantes alterações na política, com base nas recompensas obtidas. No começo, o algoritmo realiza ações aleatórias, ou seja, possui um caráter totalmente exploratório que decai à medida que ele vai recebendo informações e atualizando as políticas, até se obter a trajetória que gera a maior recompensa.

Para definir a recompensa é preciso entender primeiramente o episódio avaliado. Um episódio foi composto de três períodos: um período inicial de ações, um período intermediário de folga e um período final de avaliação, vide Figura 3.2. O período inicial de ações tem duração de uma hora, onde a cada minuto o agente realiza uma ação, em outras palavras, a cada minuto há uma atuação no output da válvula *choke*. O período intermediário dura três horas e foi implementado para permitir o sistema alcançar um equilíbrio após as ações realizadas. O período de avaliação dura seis horas, nele é avaliado, principalmente, a amplitude da pressão no ponto PDG para verificar a estabilidade e produtividade do sistema.

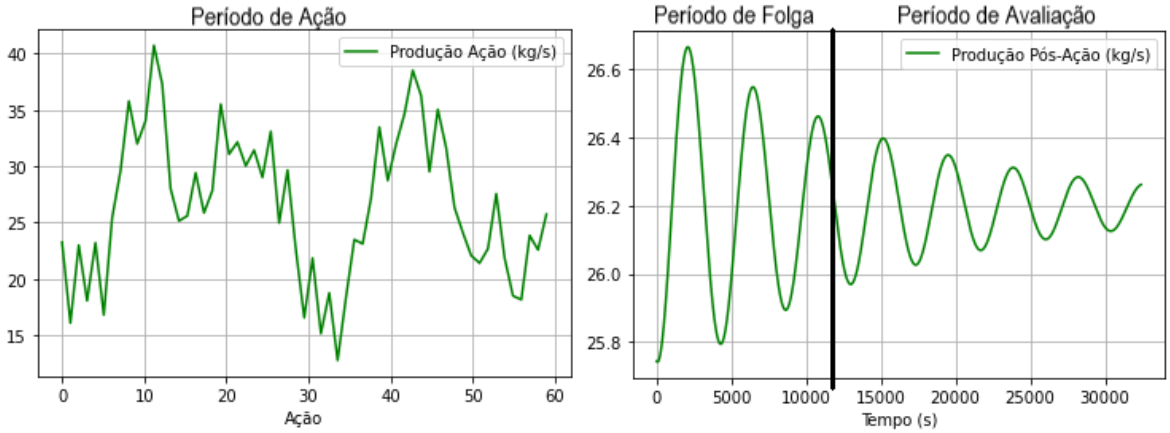

Figura 3.2: Exemplificação de um episódio com base na produção.

A recompensa é definida pela produção realizada no período de ações, somada com a produção no período de avaliação. Da recompensa total é descontada a amplitude da pressão no PDG para desencorajar o agente a levar o sistema a um estado com golfadas, vide Eq (3.1).

$$\mathrm{R_T} = \sum_n R_a + \left(\mathrm{W_f} - \Delta_{\mathrm{Ppdg}}\right) \tag{3.1}$$

R_T é a recompensa total do episódio, R_a é a recompensa de cada ação, n é o número de ações, W_f é a produção no período final de avaliação e Δ_{Ppdg} é a amplitude da pressão no ponto PDG, durante o período de avaliação.

Além disso, foi implementado uma supressão de movimento para evitar uso excessivo da válvula durante o período de ações. Essa supressão é feita com base nas variações absolutas da abertura da válvula entre cada ação, multiplicado por uma constante. Desconta-se esse percentual da recompensa da ação, vide Eq. (3.2).

$$\mathrm{R_a} = \mathrm{W_a} * (1 - |\omega_a - \omega_{a-1}| * \mathrm{C}) \tag{3.2}$$

R_a é a recompensa da ação, W_a é a produção durante aquela ação, ω_a é a abertura da válvula ao final da ação atual, ω_{a-1} é a abertura da válvula ao final da ação passada e C é a constante de supressão. A constante C é um parâmetro de ajuste e foram testados os valores de 0, 1, 2, 3 e 6 para avaliação sob diferentes intensidades de supressão.

Com base nas definições realizadas, o algoritmo de RL escolhido para aplicação foi o método *actor-critic.* Essa decisão foi tomada devido ao sistema de produção de petróleo poder ser configurado como um processo de decisão de *Markov* estacionário e possuir um ambiente contínuo. Além disso, a ordem das ações aplicadas na válvula *choke* pode levar o sistema à uma operação estável acima do ponto de *Hopf*, em outras palavras, levam a um estado pouco explorado caso seguida uma política constante.

Para aplicação do método, utilizou-se como base um algoritmo *actor-critic* (AC) desenvolvido por Nandan (2020) com base em *Keras*. Implementou-se o algoritmo AC no software *Python,* a partir da distribuidora *Anaconda,* com interface *Spyder,* exposto no Apêndice B, com auxílio das bibliotecas *Gym*, para criação do ambiente, e *Matplotlib,* para construção dos gráficos. Primeiramente se resolve o modelo *FOWM* para garantir um estado estacionário antes do início das iterações. Para os passos de ação, foi construído um ambiente em *Gym* representando uma ação na válvula. Esse ambiente é avaliado sessenta vezes e cada avaliação equivale a uma ação de um minuto, totalizando nosso período de ações. O período de avaliação não está contemplado no ambiente em *Gym*, em vez disso, ele foi considerado dentro da iteração de um episódio no algoritmo do AC. Aglomeraram-se os períodos intermediário e final com o intuito de não contabilizar um passo para o período de folga, visto que não é objetivo avaliar o que acontece durante ele. Calcula-se a recompensa do episódio e, em seguida, com auxílio de uma rede neural estática do tipo *feed foward,* é feita a avaliação das recompensas e atualização das probabilidades de ações. A rede neural recebe como *input* os sinais de estado. Esse sinal passa para uma camada comum de 128 neurônios com ativação de retificação linear (“*relu”)* que alimenta as duas camadas de *output*, uma camada para o *actor,* de densidade relativa ao número de ações e com ativação a partir de função exponencial normalizada (“*softmax”*), e uma camada para o *critic* com densidade unitária. O *actor* retorna a probabilidade das ações, dentro do espaço de ações, e o *critic* retorna uma estimativa das recompensas totais futuras. A política é atualizada via *backpropagation* e um novo episódio é iniciado.

Considerando o objetivo de explorar muitos caminhos, o algoritmo foi configurado para realizar um número de iterações definido. Explorou-se diferentes números de iterações, sendo eles 10000, 20000, 30000 e 60000, com o intuito de avaliar o impacto dessa variável no resultado do método. Entende-se uma iteração,

ou episódio, como as dez horas totais observadas no modelo, sua respectiva avaliação e atualização da política. Todas as execuções foram feitas no computador com processador *Intel(R) Core(TM) i5-9400F CPU @ 2.90GHz* e 16gb de memória RAM.

3.3 Otimização dinâmica

Adaptou-se o problema dinâmico em questão utilizando o princípio de *single-shooting* (SS) (Michalik et al., 2009). O espaço foi discretizado em um vetor de 60 variáveis, utilizado para representar as ações na válvula *choke* durante o período de ações. SS parte de um problema de valor inicial e tenta alcançar um valor objetivo com base nas derivadas para diferentes conjuntos de variáveis. Aplica-se então um *solver* para maximizar o valor objetivo, no caso, o valor da recompensa, atuando nas variáveis do vetor de entrada. Para comparação e avaliação do *reinforcement learning* (RL) se utilizou dois otimizadores disponíveis no pacote de otimização do S*cipy*: *Minimize,* utilizando o método *COBYLA,* e *Differential Evolution* (DE). O método *COBYLA* minimiza uma função escalar usando uma otimização restringida por aproximação linear. Otimização por DE objetiva o mínimo global e possui uma natureza estocástica, sem utilização de gradientes para encontrar o mínimo. Ambos utilizam como base o episódio utilizado para RL, vide Figura 3.2.

A função a ser minimizada em ambos os casos é definida como o inverso da recompensa, ou seja, o somatório da produção durante as ações e no período de avaliação. Dessa recompensa se desconta a variação de pressão no ponto PDG, durante o período de avaliação, para evitar direcionar a resolução à um cenário com golfadas. Além disso, essa recompensa está sujeita a uma supressão de movimento que consiste na soma das variações absolutas de abertura de válvula durante as ações, multiplicada por uma constante, vide Eq. (3.3):

$$\mathrm{R_T} = \left(\sum_n W_a + \left(\mathrm{W_f} - \Delta_{\mathrm{Ppdg}}\right)\right) * \left(1 - \sum_n \left(|\omega_\mathrm{a} - \omega_{\mathrm{a-1}}|\right) * \mathrm{C}\right) \qquad (3.3)$$

onde R_T é a recompensa total, n é o número de ações, W_a é a produção de cada ação, W_f é a produção no período final de avaliação, Δ_{Ppdg} é a amplitude da pressão no ponto PDG durante o período de avaliação. $|(\omega_a-\omega_{a-1})|$ é a variação absoluta da abertura de válvula entre ação atual ω_a, e a ação passada ω_{a-1}. C é a constante de

supressão, parâmetro avaliado com diferentes valores. No caso do método *COBYLA*, testou-se valores de constante de 0, 0.01, 0.04, 0.06, 0.08, 0.25, 0.5 e 0.75. Para o método DE, avaliou-se constantes em 0.04 e 0.25.

O valor de cada uma das 60 variáveis representa o delta aplicado na válvula *choke* e cada uma possui limite inferior e superior em, respectivamente, -17% e 17%. Em todas as execuções, ambos os métodos utilizaram como chute inicial um vetor de zeros, ou seja, nenhuma ação é feita na válvula. Diferente do RL que executa um número definido de iterações, esses dois otimizadores podem convergir em um resultado antes de terminar todas as iterações. O código conforme implementado para o COBYLA está disposto no Apêndice C. No caso do método DE, optou-se por uma população de 120 indivíduos e foi limitado o número de iterações máximas em 200, equivalente à aproximadamente 25000 avaliações de função, com o intuito de deixar o tempo de execução competitivo com o método AC. O código conforme implementado está explicito no Apêndice D. Todas as execuções foram realizadas no mesmo computador utilizado para o RL.

4 RESULTADOS

Para todas as otimizações, definiu-se a abertura inicial da válvula *choke* em 40%. Esse valor é arbitrário, porém se optou por uma abertura abaixo do ponto de Hopf, considerando como ponto de partida estável. Nenhuma execução apresentou golfadas severas. Um sumário dos resultados obtidos no estudo está exposto na Tabela 4.1. Supressão de movimento indica o valor da constante de supressão, relativo à cada método. Quanto maior esse número, mais intensa a supressão. A produção final faz referência à vazão média de óleo após estabilização do período de avaliação.

Tabela 4.1: Resultado geral das melhores otimizações obtidas.

Método	COBYLA	COBYLA	Differential Evolution	Differential Evolution	Actor-Critic
Constante Supressão de Movimento	0	0,25	0,04	0,25	3
Abertura Choke Final (%)	74,50	73,70	68,68	70,66	72,8
Produção Final (kg/s)	26,33	26,25	25,69	25,92	26,16
Tempo de Execução (min)	5,70	5,95	494,6	411,4	696,0

Considerando que todos os métodos alcançaram um estado sem golfadas severas, todos estão aptos a serem comparados. A abertura final da *choke* está diretamente correlacionada com a produção final alcançada, permitindo ser comparado apenas um desses resultados. O método que alcançou a maior produção final foi o *COBYLA* em 26,25 kg/s para constante de supressão 0,25, além de possuir a execução mais rápida.

4.1 *COBYLA*

Dada a agilidade do método *COBYLA* para resolução do problema, ele foi usado como triagem para avaliar o impacto das constantes de supressão de movimento. As constantes testadas e os resultados obtidos estão expostos na Tabela

4.2. Não foi comparado tempo de execução por se tratar do mesmo método e todos possuírem tempos muito próximos, na faixa de seis minutos. A tendência é, à medida que a constante de supressão aumenta, menor a abertura de válvula alcançada no final e, por consequência, menor a produção final. A partir da constante 0,5 os resultados são significativamente menores, indicando uma supressão rigorosa a ponto de restringir a obtenção de maiores recompensas. A Figura 4.1 expõe as ações realizadas na válvula com constante de supressão em 0,5. Similarmente, a Figura 4.2 expõem as ações na válvula, porém com constante de supressão em 0,75.

Tabela 4.2: Resultados do método *COBYLA* para diferentes constantes de supressão de movimento.

Constante Supressão de Movimento	**0**	**0,01**	**0,04**	**0,06**	**0,08**	**0,25**	**0,5**	**0,75**
Abertura Choke Final (%)	74,50	72,35	72,33	72,10	71,77	73,70	61,7	50,56
Produção Final (kg/s)	26,33	26,18	26,13	26,08	26,04	26,25	24,80	23,05

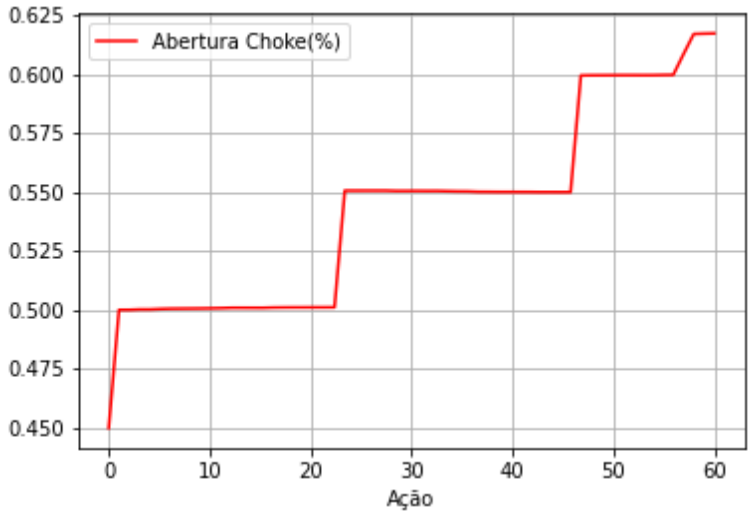

Figura 4.1: Gráfico da abertura da válvula *choke* em cada ação para o método *COBYLA* com constante de supressão 0,5.

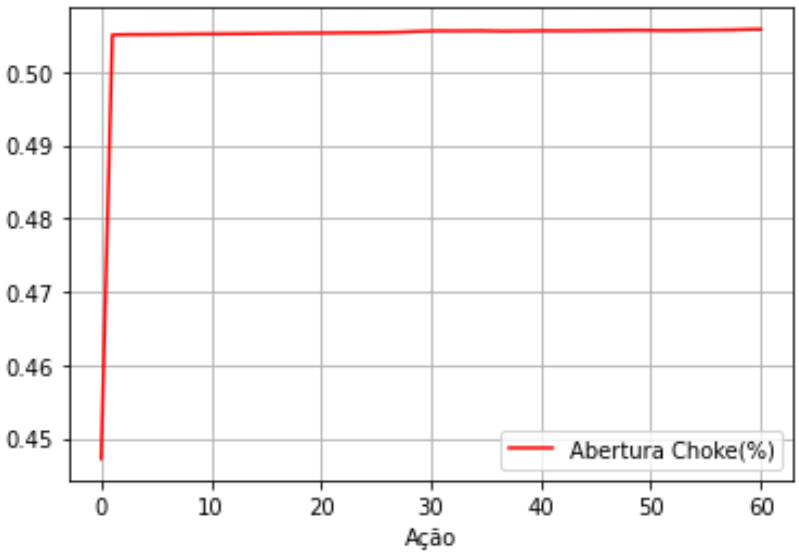

Figura 4.2: Gráfico da abertura da válvula *choke* em cada ação para o método *COBYLA* com constante de supressão 0,75.

Ao se comparar os dois melhores resultados, com constante 0 e com constante 0,25, percebe-se que, apesar de alcançar 0,8% a mais de abertura, a constante nula realiza frequentes movimentos na válvula choke, vide Figura 4.3. Por outro lado, o movimento com constante 0,25 é mais suave e a restrição não prejudica a obtenção de uma recompensa comparativamente alta, vide Figura 4.4.

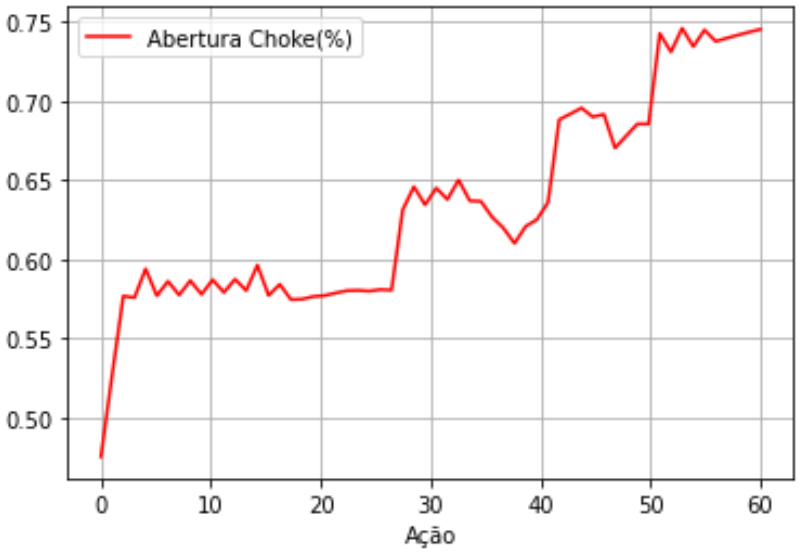

Figura 4.3: Gráfico da abertura da válvula *choke* em cada ação para o método *COBYLA* com constante de supressão 0.

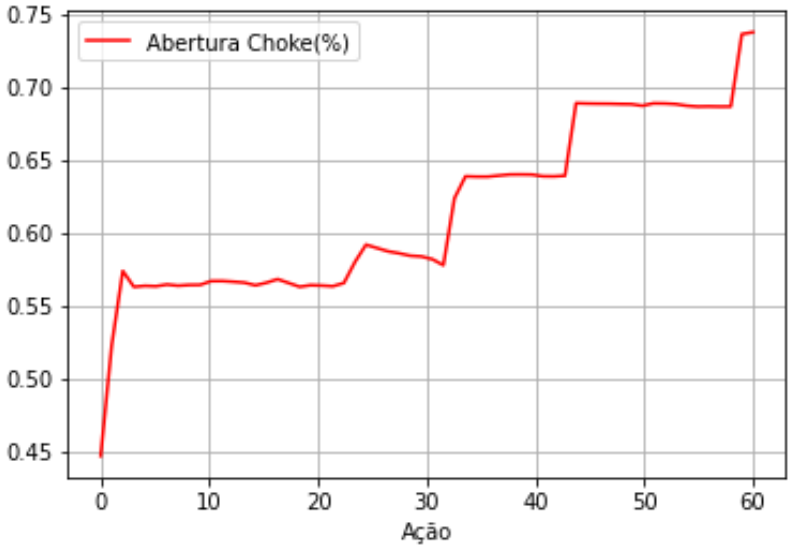

Figura 4.4: Gráfico da abertura da válvula *choke* em cada ação para o método *COBYLA* com constante de supressão 0,25.

Ressalta-se que os resultados do *COBYLA* apresentaram a tendência de realizar degraus. Além disso, nenhuma configuração levou o sistema a golfar. A Figura 4.5 apresenta a pressão no ponto PDG, após o período de ações de 1 hora, com constante de supressão em 0. O resultado para o mesmo período com constante de supressão 0,25 está exposto na Figura 4.6. Na Figura 4.7 está evidenciado a vazão de óleo após término das ações sem supressão e, semelhantemente, a Figura 4.8 expõem a produção após ações com constante de supressão 0,25. Todas as constantes apresentaram comportamentos oscilatórios com tendência à estabilização da produção e pressão no ponto PDG.

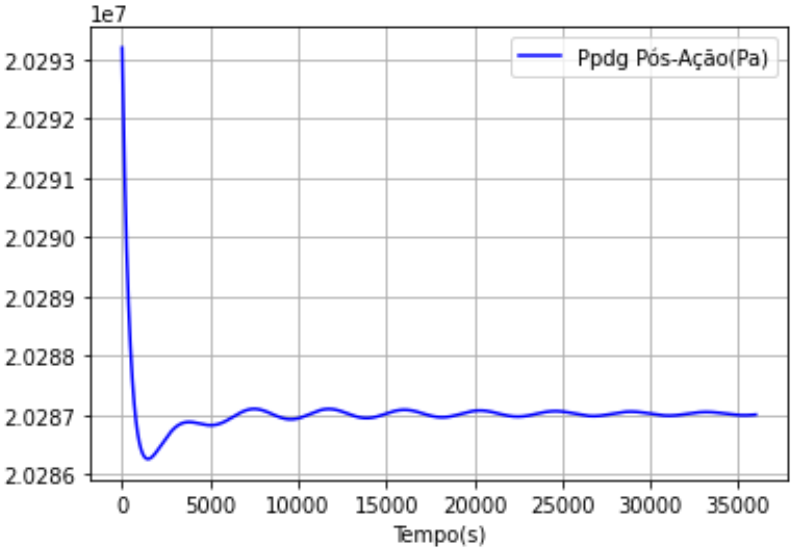

Figura 4.5: Gráfico de pressão no ponto PDG após ações com constante de supressão 0 no método *COBYLA.*

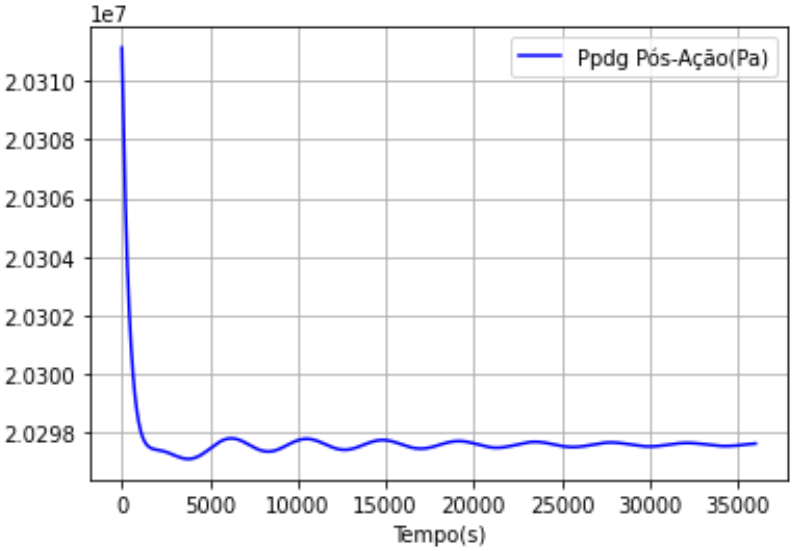

Figura 4.6: Gráfico de pressão no ponto PDG após ações com constante de supressão 0,25 no método *COBYLA.*

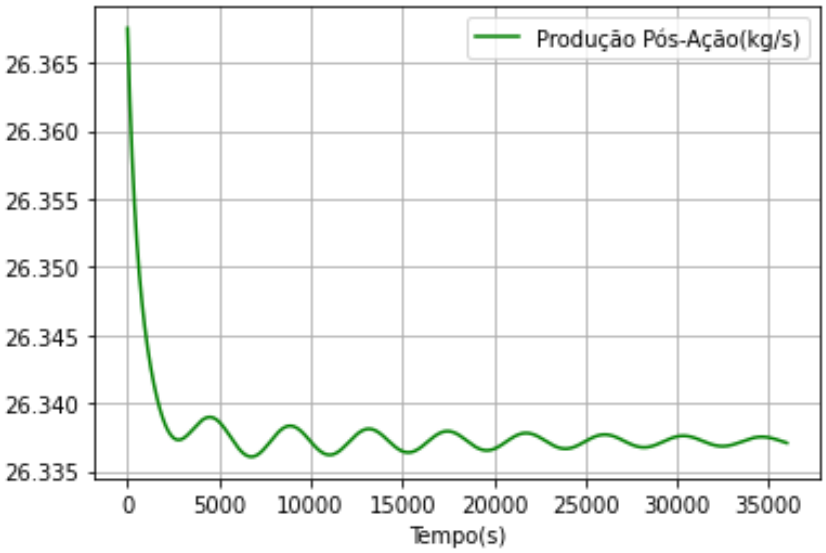

Figura 4.7: Gráfico da produção após ações com constante de supressão 0 no método *COBYLA.*

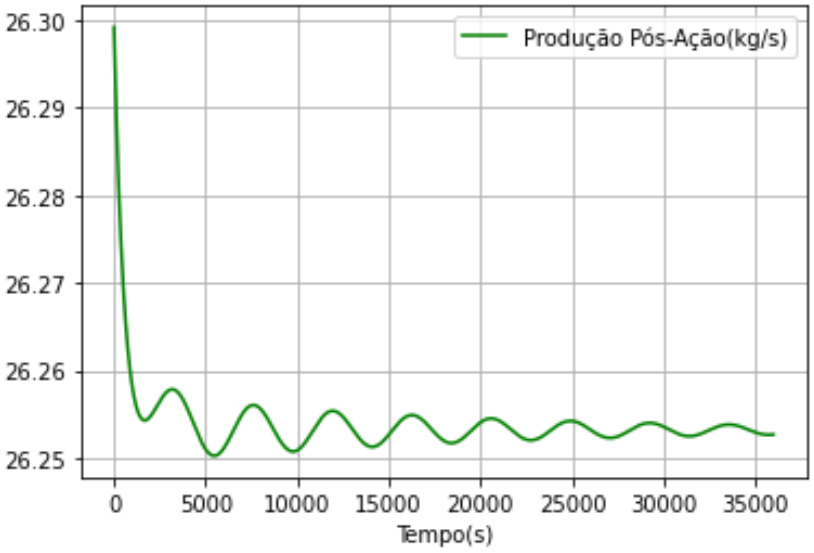

Figura 4.8: Gráfico da produção após ações com constante de supressão 0,25 no método *COBYLA.*

4.2 *Differential Evolution*

Frente aos resultados expostos na Tabela 4.2, para as constantes de supressão de movimento do método *COBYLA*, e dado tempo de execução longo para o método *Differential Evolution* (DE), optou-se por utilizar somente duas constantes de supressão, sendo elas 0,04 e 0,25. A ideia é explorar outro modelo com a melhor constante, diferente de zero, encontrado para o *COBYLA,* 0,25. Também foi escolhida uma constante mais suave, 0,04, para avaliar o impacto da intensidade da supressão. Evitou-se a execução do método com constante nula, pois é interesse encontrar um

padrão de ações menos súbitas e essa suavidade é obtida com maiores constantes de supressão, comprovado pelos resultados do *COBYLA.*

Ambos resultados não encontraram uma solução definitiva antes de exceder o limite máximo de iterações. O resultado está exposto na Tabela 4.3. Obteve-se um incremento de 1,98% de abertura de válvula com a constante de supressão em 0,25. Contudo, dada inicialização randômica, não é possível inferir que a constante de supressão impacta positivamente no valor final de abertura da *choke*. Por outro lado, a constante de supressão teve impacto significativo na suavização das ações, vide Figura 4.9, para uma constante de supressão de 0,04, e Figura 4.10, para a constante de supressão de 0,25. Além disso, os gráficos de produção e pressão no ponto PDG pós ações apresentaram comportamentos semelhantes e sem golfadas. A Figura 4.11 apresenta a produção após ações com constante de supressão 0,04, enquanto a Figura 4.12 apresenta o mesmo gráfico para constante de supressão 0,25. O comportamento da pressão no ponto PDG após o período de ações com uma constante de supressão de 0,04 está evidenciado na Figura 4.13. A Figura 4.14 representa a pressão no ponto PDG após ações quando considerada uma constante de supressão de 0,25.

Tabela 4.3: Resultados do método DE para diferentes constantes de supressão de movimento.

Constante Supressão de Movimento	**0,04**	**0,25**
Abertura Choke Final (%)	68,68	70,66
Produção Final (kg/s)	25,69	25,92

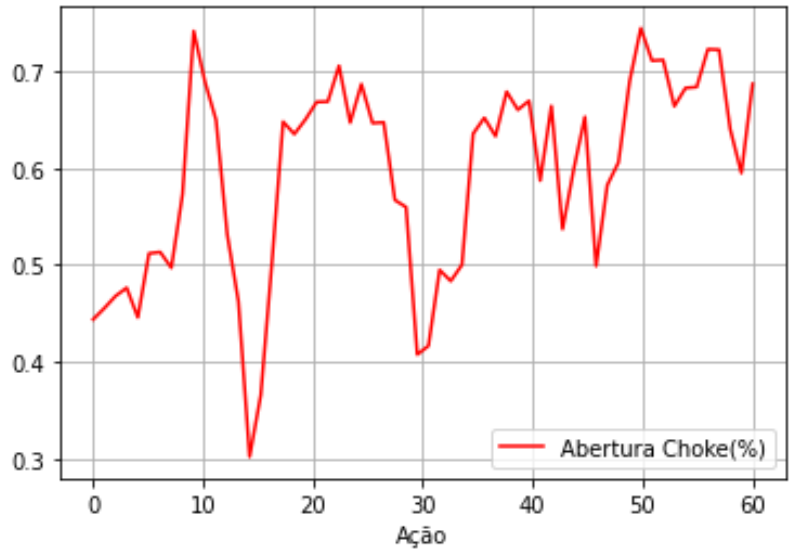

Figura 4.9: Gráfico da abertura da válvula *choke* em cada ação para o método DE com constante de supressão 0,04.

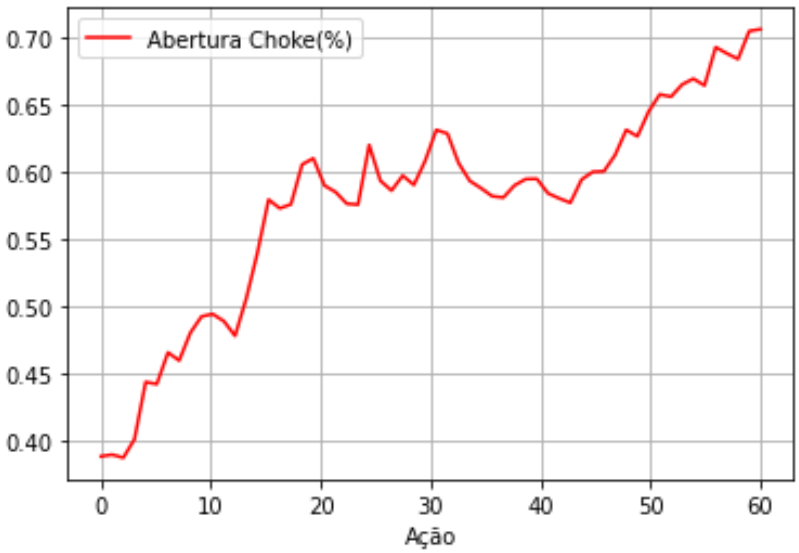

Figura 4.10: Gráfico da abertura da válvula *choke* em cada ação para o método DE com constante de supressão 0,25.

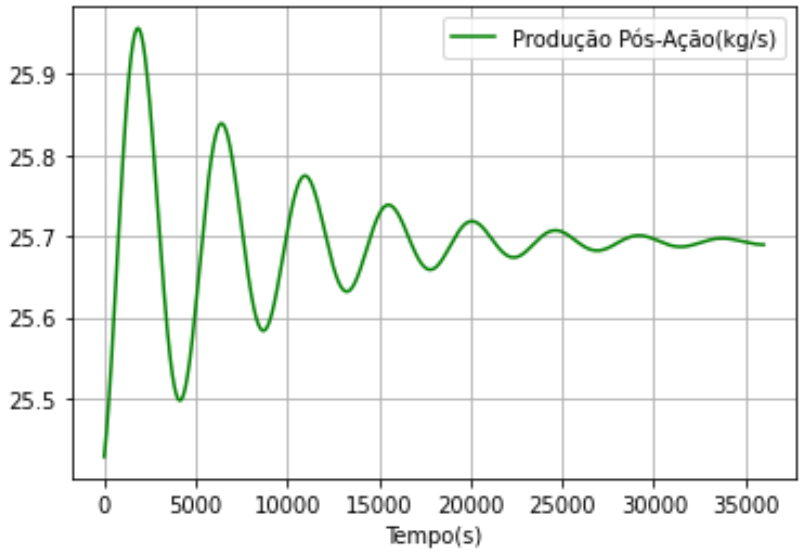

Figura 4.11: Gráfico da produção pós período de ação para o método DE com constante de supressão 0,04.

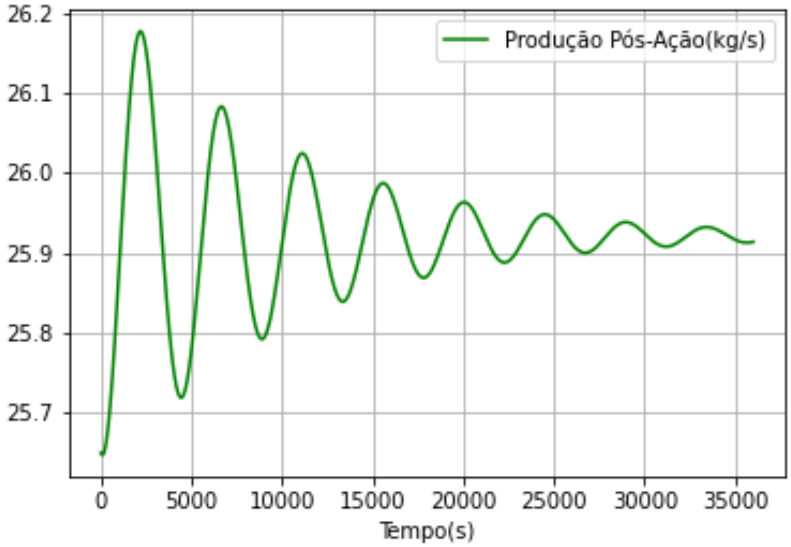

Figura 4.12: Gráfico da produção pós período de ação para o método DE com constante de supressão 0,25.

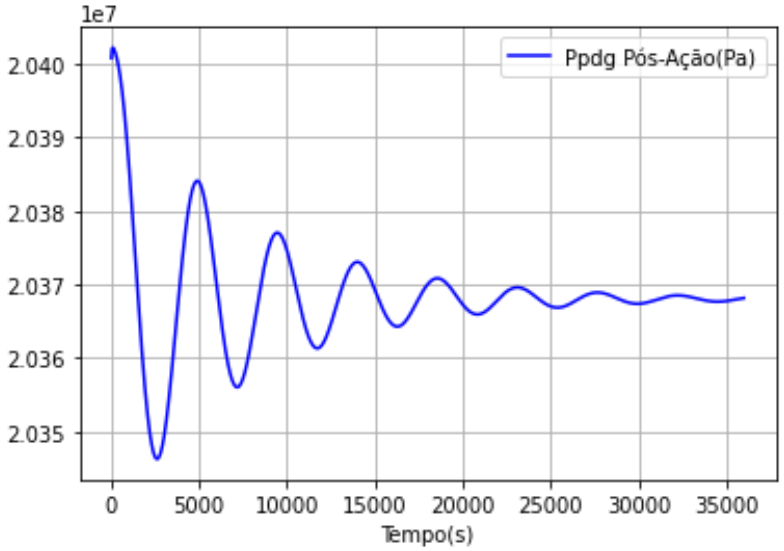

Figura 4.13: Gráfico da pressão no ponto PDG após ações para o método DE com constante de supressão 0,04.

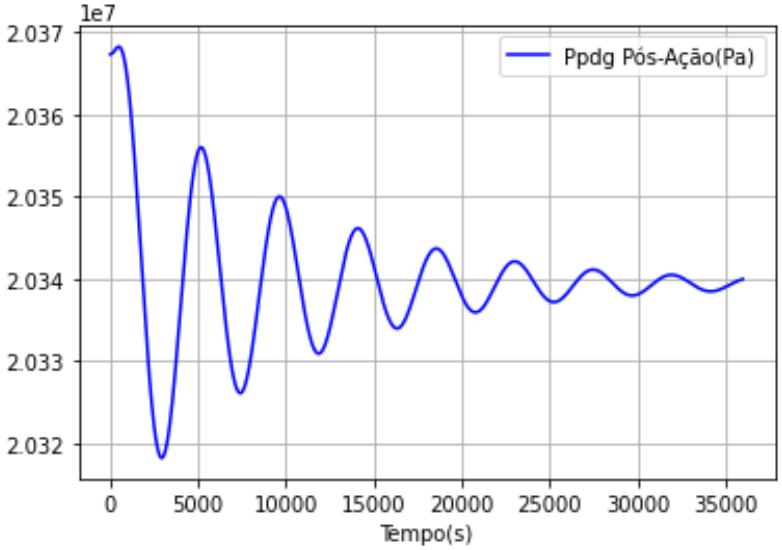

Figura 4.14: Gráfico da pressão no ponto PDG após ações para o método DE com constante de supressão 0,25.

4.3 *Actor-Critic*

A aplicação do método *actor-critic* (AC) teve como objetivo avaliar dois parâmetros: número de iterações e constante de supressão de movimento. Primeiramente, antes de se avaliar a constante, é necessário encontrar um número mínimo de iterações necessárias para o método sair de um momento totalmente exploratório e começar a gerar resultados viáveis. Essa avaliação se deu a partir do acompanhamento da amplitude de pressão no ponto PDG no final de cada iteração.

A Figura 4.15 indica que, nas primeiras 10000 iterações, o método realizou ações que, ao final, resultaram somente no máximo ou no mínimo de abertura de válvula, ou seja, na maior ou na menor amplitude de pressão no ponto PDG, respectivamente.

Ao dobrar o número de iterações, percebe-se que, nas iterações finais, o método começa a resultar em amplitudes diferentes, ou seja, aberturas finais de válvula diferentes, vide área em vermelho na Figura 4.16. A diferença entre os comportamentos exploratórios se dá pela aleatoriedade das ações durante a fase de treinamento do método. Com o intuito de avaliar a constante de supressão para um resultado mais aprimorado, optou-se por 30000 iterações como base para a comparação, resultando na Figura 4.17.

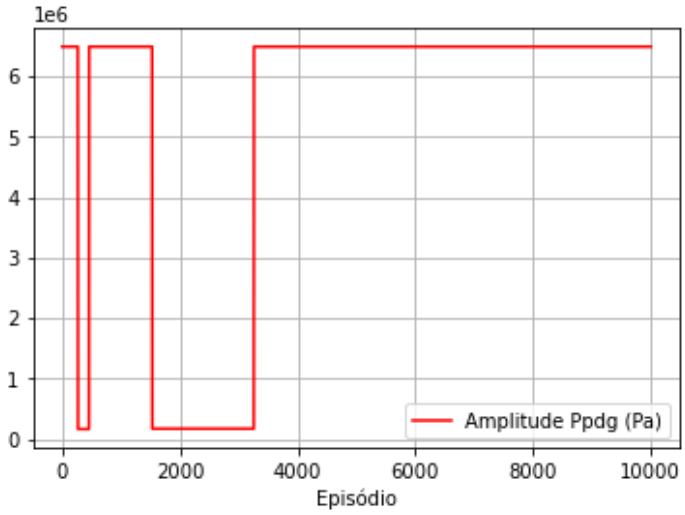

Figura 4.15: Amplitude de pressão no ponto PDG ao término do episódio para 10000 iterações do método AC.

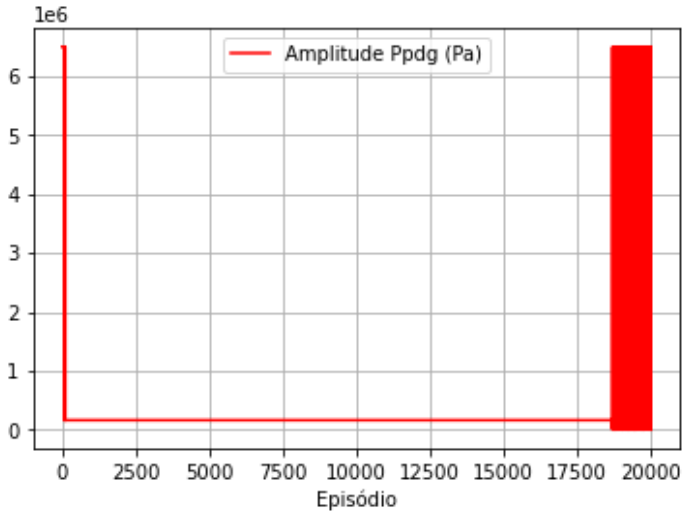

Figura 4.16: Amplitude de pressão no ponto PDG ao término do episódio para 20000 iterações do método AC.

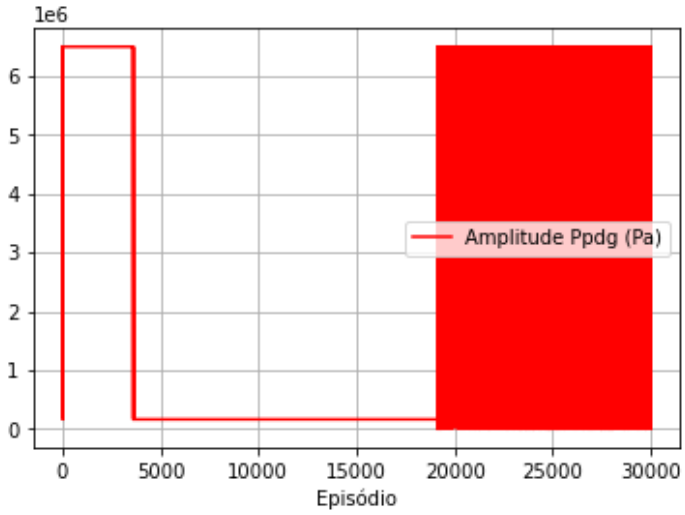

Figura 4.17: Amplitude de pressão no ponto PDG ao término do episódio para 30000 iterações do método AC.

O resultado para diferentes constantes de supressão de movimento para o método AC com 30000 iterações está exposto na Tabela 4.4. Diferente do *COBYLA*, o resultado do método AC apresenta tendência de se beneficiar com o incremento da constante de supressão. Contudo, o incremento da constante de 3 para 6 não apresentou suavização dos movimentos significativa, vide Figura 4.18 para a constante 3 e Figura 4.19 para a constante 6. Além disso, matematicamente, valores de constante acima de 6 influenciam significativamente no range de ações, dada maior variação absoluta ser 17%, vide Eq. (3.2). Com o intuito de se preservar os limites estabelecidos para as ações, optou-se pela constante de supressão 3 para ser a base do incremento de iterações.

Tabela 4.4: Resultados do método *AC* para diferentes constantes de supressão de movimento em 30000 iterações.

Constante Supressão de Movimento	**0**	**1**	**2**	**3**	**6**
Abertura Choke Final (%)	61,40	66,20	72,4	72,8	72,2
Produção Final (kg/s)	24,75	25,40	26,10	26,16	26,08

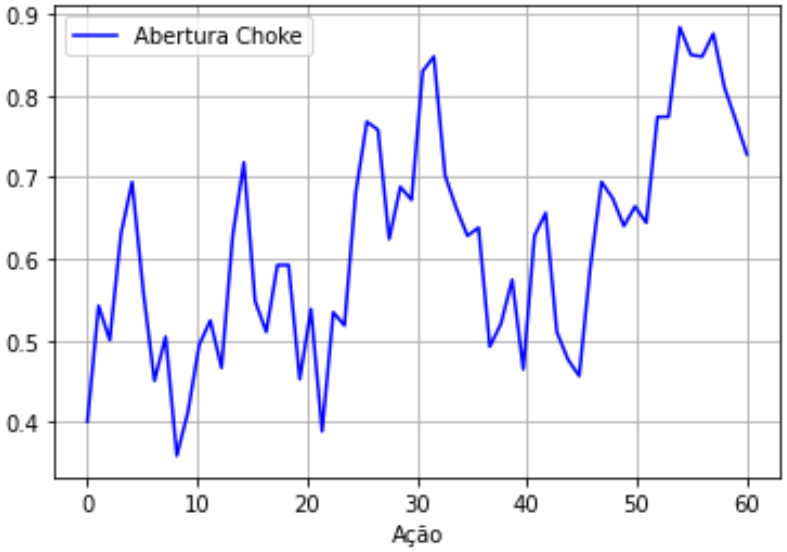

Figura 4.18: Gráfico da abertura da válvula *choke* em cada ação para o método *AC* com constante de supressão 3 em 30000 iterações.

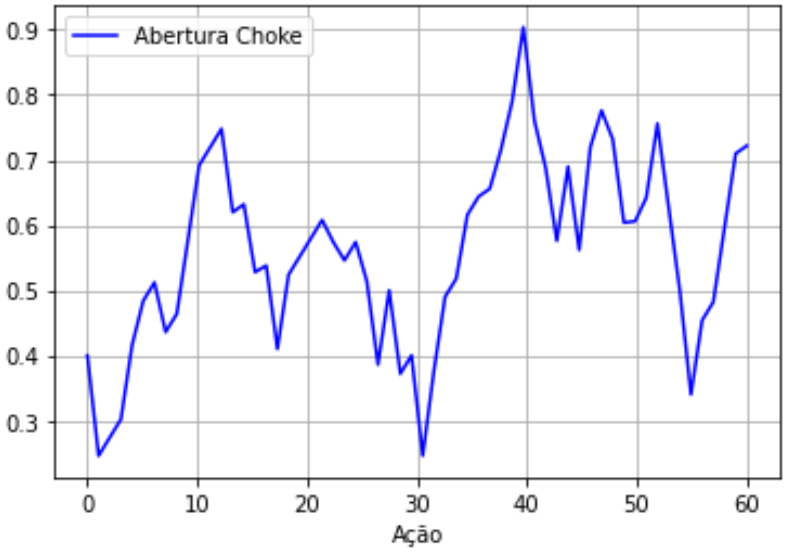

Figura 4.19: Gráfico da abertura da válvula *choke* em cada ação para o método *AC* com constante de supressão 6 em 30000 iterações.

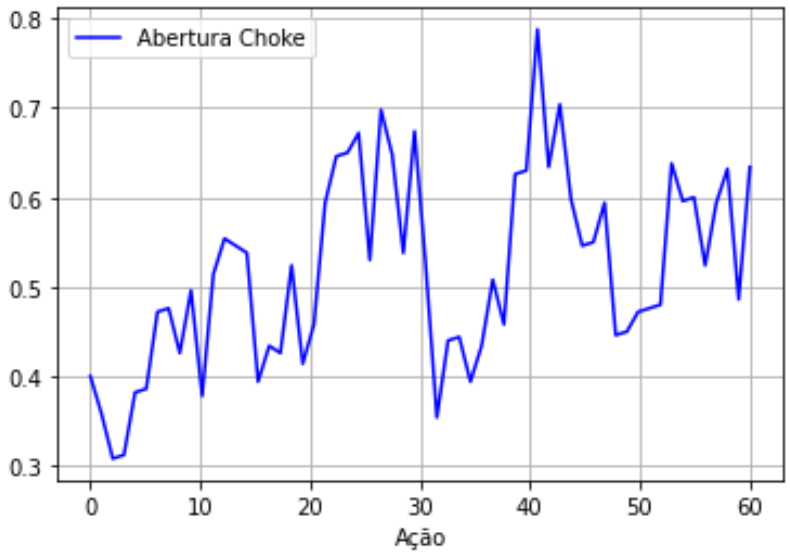

Figura 4.20: Gráfico da abertura da válvula *choke* em cada ação para o método *AC* com constante de supressão 3 em 60000 iterações.

Para avaliar qual o impacto do número de episódios na obtenção de resultados, realizou-se uma execução de 60000 iterações com fator de supressão de movimento em 3 para avaliar o horizonte dos resultados. A comparação entre 30000 iterações e 60000 iterações está exposta na Tabela 4.5. A execução de 30000 iterações alcançou uma abertura de válvula 9,4% maior que a de 60000 iterações. Esse resultado aponta que o número de iterações não está sendo um fator determinante para a efetividade da otimização. Sabe-se que há uma aleatoriedade intrínseca ao método AC, porém essa deveria ser gradativamente reduzida em detrimento ao treinamento do método. É valido considerar que, se continuada, eventualmente a execução de 60000 iterações alcançaria um resultado similar ou melhor que o de 30000, dado caráter exploratório da metodologia, porém o tempo de execução seria ainda maior. Outro ponto que pode influenciar nesse quesito é a quantidade elevada de estados e ações. O período de treino pode acabar explorando ações que direcionam o agente a estressar um caminho não ótimo, até que ele descubra possíveis rotas melhores. Cenários que isso ocorra aumentam consideravelmente o número de iterações para alcançar melhores resultados. O gráfico das ações, resultado das 30000 iterações, está exposto na Figura 4.18. As ações resultantes das 60000 iterações estão representadas na Figura 4.20. A Figura 4.21 apresenta o comportamento da pressão no ponto PDG, após ações, do resultado das 30000 iterações. A pressão no ponto PDG do resultado das 60000 iterações está na Figura 4.22. A Figura 4.23 expõem a produção após ações

para o resultado das 30000 iterações e, similarmente, a Figura 4.24 representa o mesmo para a execução de 60000 iterações.

Tabela 4.5: Comparação dos resultados do método AC para diferentes número de iterações com constante de supressão 3.

Número de Iterações	**30000**	**60000**
Constante Supressão de Movimento	3	3
Abertura Final da *Choke* (%)	72,8	63,4
Produção Final (kg/s)	26,16	25,03
Golfadas	Não	Não
Tempo de execução (h)	11,6	21,6

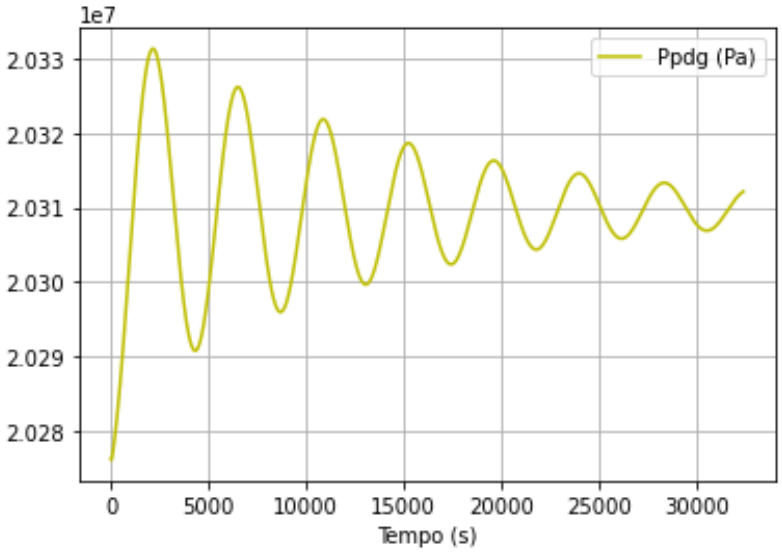

Figura 4.21: Gráfico da pressão no ponto PDG para o método AC com constante de supressão 3 em 30000 iterações

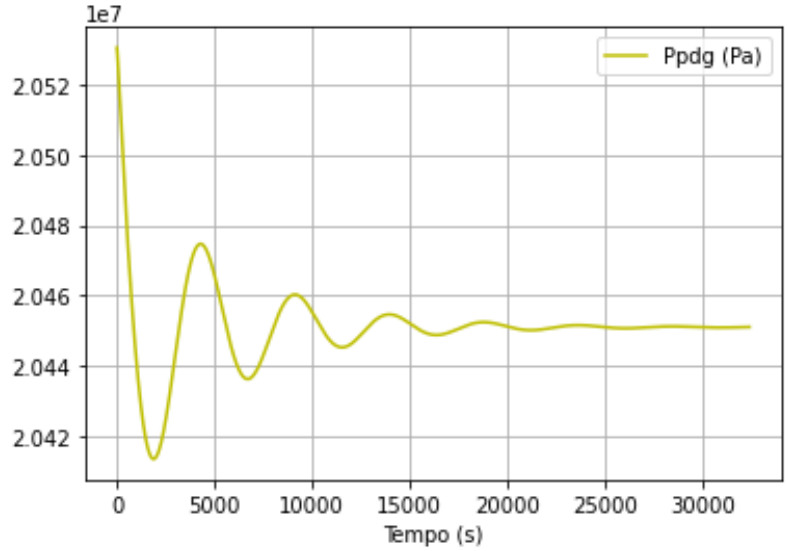

Figura 4.22: Gráfico da pressão no ponto PDG para o método AC com constante de supressão 3 em 60000 iterações

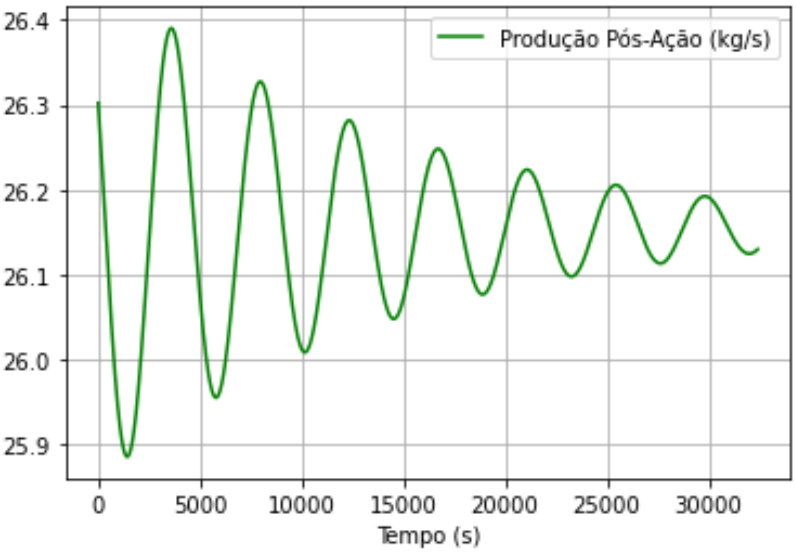

Figura 4.23: Gráfico da produção pós ações para o método AC com constante de supressão 3 em 30000 iterações.

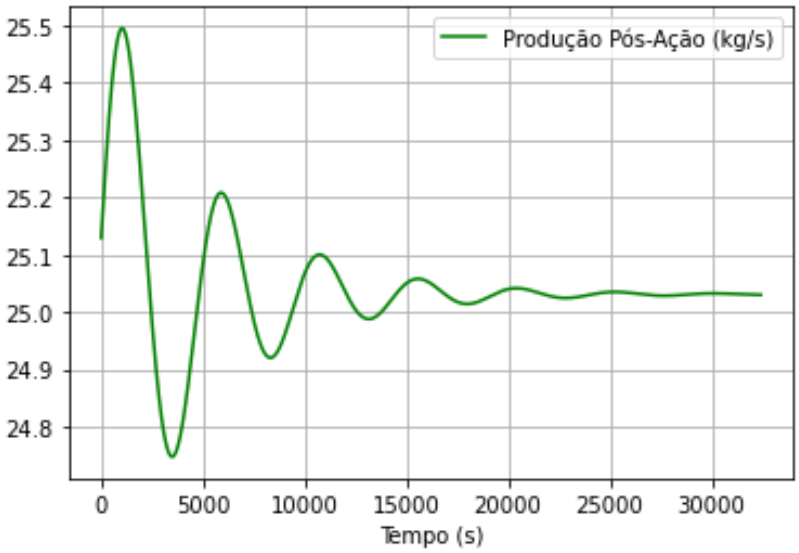

Figura 4.24: Gráfico da produção pós ações para o método AC com constante de supressão 3 em 60000 iterações.

5 CONCLUSÕES E TRABALHOS FUTUROS

O estudo aplicou a metodologia de *reinforcement learning* (RL) para otimizar a produção de petróleo e analisar a viabilidade quando comparado com a otimização dinâmica, via métodos *COBYLA* e *Differential Evolution* (DE), disponíveis atualmente no pacote de otimização do *scipy*. Os testes foram realizados com base no modelo simplificado para poços de elevação de petróleo (Diehl et al., 2017).

Explorou-se uma configuração de algoritmo *actor-critic* (AC), no qual a política é atualizada com base nas recompensas obtidas, aplicando uma rede neural estática. O método alcançou o objetivo de gerar resultados viáveis, alcançando aberturas de válvula de 72,8%, sem golfar. Contudo, o tempo de execução para obtenção desses resultados foi de 11,6 horas, valor demasiadamente elevado quando comparado aos outros métodos. O método DE, também lento, apresentou o resultado menos otimizado de 70,68% de abertura em 6,85 horas de execução. Em questão de agilidade e valores, o método *COBYLA* apresentou as melhores características, alcançando abertura de válvula de 73,70%, sem golfar, com o tempo de execução de aproximadamente 5,95 minutos. Com as configurações do método AC aplicadas nesse estudo, entende-se não ser efetivo utilizar essa metodologia em detrimento ao método *COBYLA.*

RL, por definição, é uma metodologia com um caráter exploratório, ou seja, é esperado que o método possua um maior tempo de execução. Dito isso, a não

dependência do AC com o número de iterações direciona possíveis aprimoramentos a serem feitos no algoritmo utilizado, para aumentar a efetividade do método. A utilização de uma rede neural estática é um forte limitador, principalmente quando aplicada em um sistema dinâmico como o desse estudo. Além disso, não há evidência de que o conjunto abertura de válvula e pressão no ponto PDG, como estado observado, é o mais otimizado para o problema em questão.

Trabalhos futuros incluem a aplicação de novas arquiteturas de redes neurais, a exemplo de LSTM, fator que pode agilizar significativamente o tempo de execução. Não só, mas a realização de um estudo para identificar qual o sinal de estado mais expressivo para o problema em questão, dado importância desse valor para treinamento do método AC. Outro ponto é o episódio explorado no estudo estar livre de perturbações esporádicas. Julga-se válido avaliar a resposta do método AC para um sistema com irregularidades, dada maior semelhança com possíveis cenários reais. Além, avaliar outros métodos de RL, que não AC, e como esses respondem ao problema em questão.

Entende-se que esse estudo obteve sucesso em dar início a aplicação e estudos de RL para a otimização da produção do petróleo. Evidencia-se com base nesse estudo que ainda há diversas fronteiras a serem exploradas quando o assunto é *machine learning,* em especial, *reinforcement learning,* e que há espaço para esses conceitos dentro da indústria petrolífera.

REFERÊNCIAS

Alves Goulart, D., & Dutra Pereira, R. (2020). Autonomous pH control by reinforcement learning for electroplating industry wastewater. *Computers and Chemical Engineering*, *140*. https://doi.org/10.1016/j.compchemeng.2020.106909

AYODELE, T. O. (2010). *New Advances in Machine Learning*. https://books.google.com.br/books?hl=pt-BR&lr=&id=XAqhDwAAQBAJ&oi=fnd&pg=PA19&dq=machine+learning+types&ots=r2Hj6WAfLt&sig=nX6n_2N7QqagtYyZY1lLQYgLvoQ#v=onepage&q=machine learning types&f=false

Bagherzadeh, F., Mehrani, M. J., Basirifard, M., & Roostaei, J. (2021). Comparative study on total nitrogen prediction in wastewater treatment plant and effect of various feature selection methods on machine learning algorithms performance. *Journal of Water Process Engineering*, *41*. https://doi.org/10.1016/j.jwpe.2021.102033

Barman, R. K., Mukhopadhyay, A., Maulik, U., & Das, S. (2019). Identification of infectious disease-associated host genes using machine learning techniques. *BMC Bioinformatics*, *20*(1). https://doi.org/10.1186/s12859-019-3317-0

Bi, S., Zhang, B., Mu, L., Ding, X., & Wang, J. (2020). Optimization of tobacco drying process control based on reinforcement learning. *Drying Technology*, *38*(10), 1291–1299. https://doi.org/10.1080/07373937.2019.1633662

Décio, D.-G., Diretores, O., Amaral, A., Amorelli, D., Kury, F., & Cechi, J. C. (2018). *AGÊNCIA NACIONAL DO PETRÓLEO, GÁS NATURAL E BIOCOMBUSTÍVEIS*.

Di Meglio, F., Petit, N., Alstad, V., & Kaasa, G. O. (2012). Stabilization of slugging in oil production facilities with or without upstream pressure sensors. *Journal of Process Control*, *22*(4), 809–822. https://doi.org/10.1016/j.jprocont.2012.02.014

Diehl, F. C., Anzai, T. K., Almeida, C. S., Von Meien, O. F., Neto, S. S., Rosa, V. R., Campos, M. C. M. M., Reolon, F., Gerevini, G., Ranzan, C., Farenzena, M., & Trierweiler, J. O. (2017). Fast Offshore Wells Model (FOWM): A practical dynamic model for multiphase oil production systems in deepwater and ultra-deepwater scenarios. *Computers and Chemical Engineering*, *99*, 304–313. https://doi.org/10.1016/j.compchemeng.2017.01.036

Diehl, F. C., Machado, T. O., Anzai, T. K., Almeida, C. S., Moreira, C. A., Nery, G. A., Campos, M. C. M. M., Farenzena, M., & Trierweiler, J. O. (2019). 10% increase in oil production through a field applied APC in a petrobras ultra-deepwater well. *Control Engineering Practice*, *91*, 104108. https://doi.org/10.1016/j.conengprac.2019.104108

Foroughi, B., Shahrouzi, J. R., & Nemati, R. (2021). Detection of Gasoline Adulteration Using Modified Distillation Curves and Artificial Neural Network. *Chemical Engineering and Technology*, *44*(3), 527–534. https://doi.org/10.1002/ceat.202000217

Foss, B., Knudsen, B. R., & Grimstad, B. (2018). Petroleum production optimization – A static or dynamic problem? *Computers and Chemical Engineering*, *114*, 245–253. https://doi.org/10.1016/j.compchemeng.2017.10.009

Galimberti, A., Bruno, A., Agostinetto, G., Casiraghi, M., Guzzetti, L., & Labra, M. (2021). Fermented food products in the era of globalization: tradition meets biotechnology innovations. In *Current Opinion in Biotechnology* (Vol. 70, pp. 36–41). Elsevier Ltd. https://doi.org/10.1016/j.copbio.2020.10.006

Grondman, I., Busoniu, L., Lopes, G. A. D., & Babuška, R. (2012). A survey of actor-critic

reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, *42*(6), 1291–1307. https://doi.org/10.1109/TSMCC.2012.2218595

Gym. (2021). https://gym.openai.com/docs/

Hahne, F., Huber, W., Gentleman, R., Falcon, S., Gentleman, R., & Carey, V. J. (2008). Unsupervised Machine Learning. In *Bioconductor Case Studies* (pp. 137–157). Springer New York. https://doi.org/10.1007/978-0-387-77240-0_10

Jahanshahi, E. (2013). Control Solutions for Multiphase Flow. In *Department of Chemical Engineering: Vol. PhD* (Issue October).

Ji, C., Jiao, Z., Yuan, S., El-Halwagi, M. M., & Wang, Q. (2021). Development of novel combustion risk index for flammable liquids based on unsupervised clustering algorithms. *Journal of Loss Prevention in the Process Industries*, *70*. https://doi.org/10.1016/j.jlp.2021.104422

Kotsiantis, S. B. (2007). Supervised Machine Learning: A Review of Classification Techniques. In *Informatica* (Vol. 31). https://books.google.com.br/books?hl=en&lr=&id=vLiTXDHr_sYC&oi=fnd&pg=PA3&dq=supervised+machine+learning+a+review+of+classification+techniques&ots=CZlzut0lkk&sig=yg3VmaVAuVqqyFCRuIfKULEfFqY

Li, H., & Misra, S. (2021). Reinforcement learning based automated history matching for improved hydrocarbon production forecast. *Applied Energy*, *284*(April 2020), 116311. https://doi.org/10.1016/j.apenergy.2020.116311

Michalik, C., Hannemann, R., & Marquardt, W. (2009). Incremental single shooting-A robust method for the estimation of parameters in dynamical systems. *Computers and Chemical Engineering*, *33*(7), 1298–1305. https://doi.org/10.1016/j.compchemeng.2009.02.002

Nandan, A. (2020). *Actor Critic Method*. https://keras.io/examples/rl/actor_critic_cartpole/

Oh, D. H., Adams, D., Vo, N. D., Gbadago, D. Q., Lee, C. H., & Oh, M. (2021). Actor-critic reinforcement learning to estimate the optimal operating conditions of the hydrocracking process. *Computers and Chemical Engineering*, *149*. https://doi.org/10.1016/j.compchemeng.2021.107280

Ohyama, J., Kinoshita, T., Funada, E., Yoshida, H., Machida, M., Nishimura, S., Uno, T., Fujima, J., Miyazato, I., Takahashi, L., & Takahashi, K. (2021). Direct design of active catalysts for low temperature oxidative coupling of methaneviamachine learning and data mining. *Catalysis Science and Technology*, *11*(2), 524–530. https://doi.org/10.1039/d0cy01751e

Overview — Matplotlib 3.4.2 documentation. (2021). https://matplotlib.org/stable/contents.html

Ren, F., Rabault, J., & Tang, H. (2020). Applying deep reinforcement learning to active flow control in turbulent conditions. In *arXiv*. arXiv. https://doi.org/10.1063/5.0037371

scipy.optimize.differential_evolution — SciPy v1.6.3 Reference Guide. (2021). https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.differential_evolution.html

scipy.optimize.minimize — SciPy v1.6.3 Reference Guide. (2021). https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html#scipy.optimize.minimize

Soori, T., Rassoulinejad-Mousavi, S. M., Zhang, L., Rokoni, A., & Sun, Y. (2021). A machine learning approach for estimating surface tension based on pendant drop images. *Fluid Phase Equilibria*, 113012. https://doi.org/10.1016/j.fluid.2021.113012

Sun, L. W., Li, H., Zhang, X. Q., Gao, H. B., & Luo, M. B. (2020). Identifying Conformation States of Polymer through Unsupervised Machine Learning. *Chinese Journal of Polymer Science (English Edition)*, *38*(12), 1403–1408. https://doi.org/10.1007/s10118-020-2442-6

Sutton, R. S., & Barto, A. G. (2017). Reinforcement learning : an introduction 2nd (19 June, 2017). *Neural Networks IEEE Transactions On*, *9*(5).

Yoo, H., Kim, B., Kim, J. W., & Lee, J. H. (2021). Reinforcement learning based optimal control of batch processes using Monte-Carlo deep deterministic policy gradient with phase segmentation. *Computers and Chemical Engineering*, *144*. https://doi.org/10.1016/j.compchemeng.2020.107133

Yu, J., Rui, Y., Tang, Y. Y., & Tao, D. (2014). High-order distance-based multiview stochastic learning in image classification. *IEEE Transactions on Cybernetics*, *44*(12), 2431–2442. https://doi.org/10.1109/TCYB.2014.2307862

Yu, K. H., Chen, Y. A., Jaimes, E., Wu, W. C., Liao, K. K., Liao, J. C., Lu, K. C., Sheu, W. J., & Wang, C. C. (2021). Optimization of thermal comfort, indoor quality, and energy-saving in campus classroom through deep Q learning. *Case Studies in Thermal Engineering*, *24*. https://doi.org/10.1016/j.csite.2021.100842

APÊNDICE A – MODELO FOWM

O modelo em questão utiliza as variáveis expostas na Figura A.0.1. Na base, ele pode ser entendido como um balanço de massa entre as etapas de elevação de petróleo, de acordo com as equações:

$$\frac{dm_{ga}}{dt} = W_{gc} - W_{iv} \quad \text{(A.1)}$$

$$\frac{dm_{gt}}{dt} = W_r \alpha_{gw} + W_{iv} - W_{whg} \quad \text{(A.2)}$$

$$\frac{Dm_{lt}}{dt} = W_r(1 - \alpha_{gw}) - W_{whl} \quad \text{(A.3)}$$

$$\frac{Dm_{gb}}{dt} = (1 - E)W_{whg} - W_g \quad \text{(A.4)}$$

$$\frac{dm_{gr}}{dt} = EW_{whg} + W_g - W_{gout} \quad \text{(A.5)}$$

$$\frac{dm_{lr}}{dt} = W_{whl} - W_{lout} \quad \text{(A.6)}$$

onde m_{ga} é a massa de gás no anular, m_{gt} é a massa de gás no *tubing*, m_{lt} é massa de líquido no *tubing*, m_{gb} é a massa de gás na bolha na linha de transporte, m_{gr} e m_{lr} são, respectivamente, a massa de gás e de líquido na linha de elevação. *E* é a fração de gás que não se prende na bolha e α_{gw} é a fração mássica de gás na pressão e temperatura do reservatório.

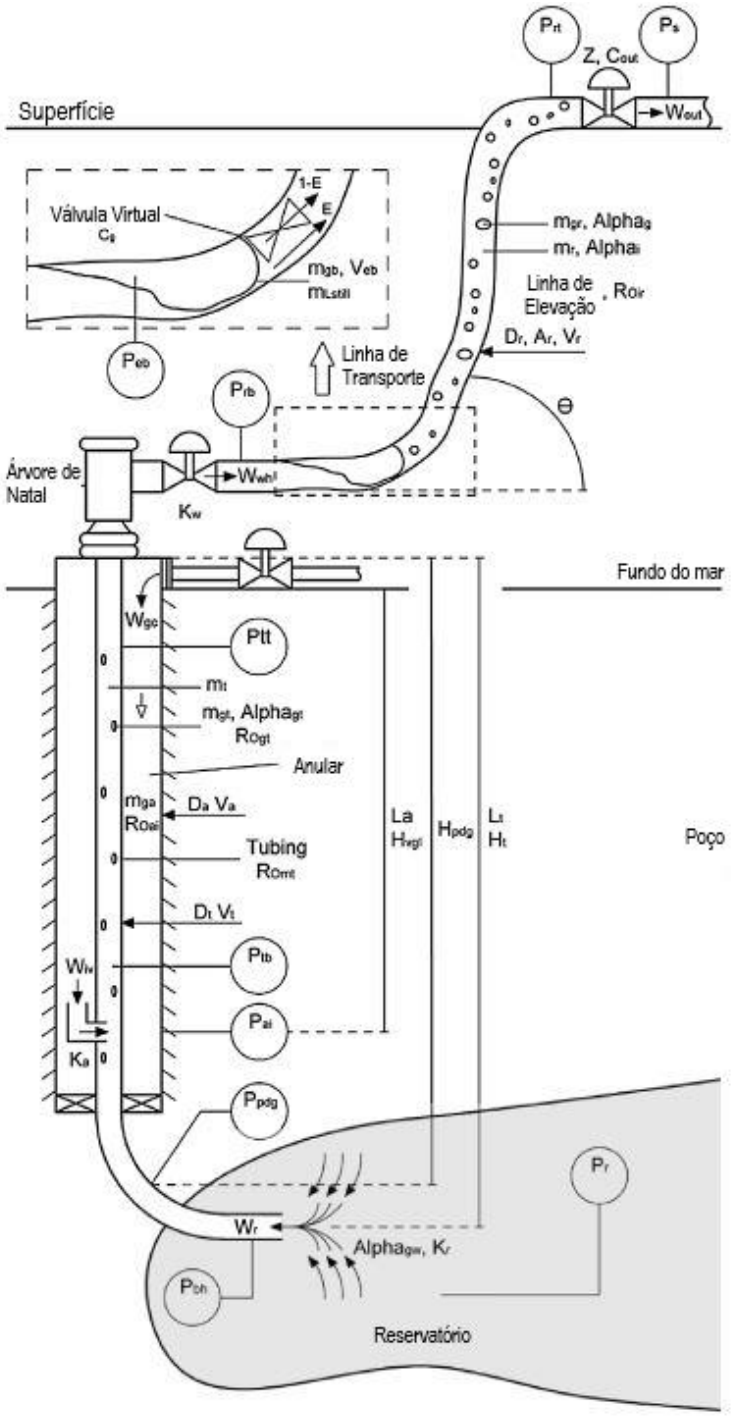

Figura A.0.1: Sistema de óleo e gás em alto mar modelado por FOWM.

Fonte: Diehl et al. (2017)

As vazões de fluidos são obtidas a partir de:

$$W_{iv} = K_a \sqrt{\rho_{ai}(P_{ai} - P_{tb})} \tag{A.7}$$

$$W_r = K_r \left[1 - \left(0{,}2 \frac{P_{bh}}{P_r}\right) - \left(0{,}8 \frac{P_{bh}}{P_r}\right)^2\right] \tag{A.8}$$

$$W_{whg} = K_w \alpha_{gt} \sqrt{\rho_l (P_{tt} - P_{rb})} \quad \text{(A.9)}$$

$$W_{whl} = K_w (1 - \alpha_{gt}) \sqrt{\rho_l (P_{tt} - P_{rb})} \quad \text{(A.10)}$$

$$W_g = C_g (P_{eb} - P_{rb}) \quad \text{(A.11)}$$

$$W_{gout} = \alpha_g C_{out} z \sqrt{\rho_l (P_{rt} - P_s)} \quad \text{(A.12)}$$

$$W_{lout} = \alpha_l C_{out} z \sqrt{\rho_l (P_{rt} - P_s)} \quad \text{(A.13)}$$

onde W_{gc} é a massa de gás para elevação entrando no anular, W_{iv} é a massa de gás que flui do anular para o *tubing*, W_r é a vazão do reservatório para o buraco de fundo, W_{whg} e W_{whl} são, respectivamente, a vazão de gás e líquido na árvore de natal, W_g é a vazão na válvula virtual, W_{gout} e W_{lout} são, respectivamente, a vazão de gás e líquido pela válvula *choke* no topo da plataforma. K_a é coeficiente de vazão entre o anular e o tubing, K_r é parâmetro diretamente proporcional à produção do reservatório ao *tubing*, K_w é o coeficiente de vazão na árvore de natal.

A densidade do anular ρ_{ai} é dada pela Eq. (A.14), ρ_L é a densidade do líquido e é considerada constante. Das Eq. (A.14) – (A.17), α_{gt} é a fração mássica no *tubing*, enquanto α_{gr} e α_{lr} são a fração de gás e líquido na linha de transporte. As constantes da válvula virtual e da válvula *choke* são representadas como C_g e C_{out}. A fração de abertura da *choke* é dada por z. P_r é a pressão na entrada do buraco de fundo, P_S é a pressão no separador gravitacional na plataforma. As outras pressões utilizadas nas Eq. (3.7) – (3.13) são descritas pelas Eq. (3.18) – (3.25):

$$\rho_{ai} = \frac{MP_{ai}}{RT} \quad \text{(A.14)}$$

$$\alpha_{gt} = \frac{m_{gt}}{m_{gt} + m_{lt}} \quad \text{(A.15)}$$

$$\alpha_{gr} = \frac{m_{gr}}{m_{gr} + m_{lr}} \quad \text{(A.16)}$$

$$\alpha_{lr} = 1 - \alpha_{gr} \quad \text{(A.17)}$$

$$P_{ai} = \left(\frac{RT}{V_a M} + \frac{g L_a}{V_a} \right) m_{ga} \quad \text{(A.18)}$$

$$P_{tb} = P_{tt} + \rho_{mt} g H_{vgl} \quad \text{(A.19)}$$

$$P_{bh} = P_{pdg} + \rho_{mres} g (H_t - H_{pdg}) \quad \text{(A.20)}$$

$$P_{pdg} = P_{tb} + \rho_{mres} g (H_{pdg} - H_{vgl}) \quad \text{(A.21)}$$

$$P_{tt} = \frac{\rho_{gt} RT}{M} \quad \text{(A.22)}$$

$$P_{rb} = P_{rt} + \frac{(m_{lr} + m_{L,still}) g \sin(\theta)}{A_{ss}} \quad \text{(A.23)}$$

$$P_{eb} = \frac{m_{gb} RT}{M V_{eb}} \quad \text{(A.24)}$$

$$P_{rt} = \frac{m_{gr} RT}{M(\omega_u V_{ss} - \frac{m_{lr} + m_{L,still}}{\rho_l})} \quad \text{(A.25)}$$

onde P_{ai} é a pressão no ponto de injeção de gás no anular, P_{tb} é a pressão no ponto de injeção de gás na lateral do *tubing*, P_{bh} é a pressão no buraco de fundo, P_{pdg} é a pressão no fundo da coluna de produção (PDG), P_r é a pressão do reservatório, P_{tt} é a pressão no topo do *tubing,* P_{rb} é a pressão na linha de transporte antes da bolha, P_{eb} é a pressão da bolha e P_{rt} é a pressão no topo da linha de elevação. Estima-se o comportamento do gás como ideal, T é a temperatura média do sistema de produção, M é a massa molecular do gás, R é a constante universal dos gases e g é a aceleração da gravidade. As propriedades de densidade e dimensionamento restantes são dadas pelas seguintes equações:

$$\rho_{mt} = \frac{m_{gt} + m_{lt}}{V_{gt}} \quad \text{(A.26)}$$

$$\rho_{gt} = \frac{m_{gt}}{V_{gt}} \quad \text{(A.27)}$$

$$V_{gt} = V_t - \frac{m_{lt}}{\rho_l} \quad \text{(A.28)}$$

$$A_{ss} = \frac{\pi D_{ss}^{\,2}}{4} \quad \text{(A.29)}$$

$$V_{ss} = \frac{\pi D_{ss}^{\,2} L_r}{4} + \frac{\pi D_{ss}^{\,2} L_{fl}}{4} \quad \text{(A.30)}$$

$$V_a = \frac{\pi D_a^{\,2} L_a}{4} \quad \text{(A.31)}$$

$$V_t = \frac{\pi D_t^{\,2} L_t}{4} \tag{A.32}$$

onde V_a é o volume do anular, L_a é o comprimento do anular, ρ_{mt} é a densidade da mistura no *tubing*, ρ_{gt} é a densidade do gás no *tubing*. Considera-se a mistura no reservatório como constante e com densidade ρ_{mres}. H_{vgl} é a altura do ponto de inserção de gás no *tubing* e a árvore de natal, H_{pdg} é a altura do ponto PDG até a árvore de natal e H_t é a altura do buraco de fundo até a árvore de natal. A inclinação média da linha de elevação é dada por Θ. $m_{L,still}$ é a mínima massa de líquido na linha de transporte, V_{eb} é o volume da bolha e ω_u é parâmetro de posicionamento da bolha.

APÊNDICE B – *ACTOR-CRITIC* EM *PYTHON*

Da Figura B.0.1 a Figura B.0.17 é explicitado o código utilizado em *Python* para aplicar o método *actor-critic* (AC) na otimização de produção de poços de petróleo. Utilizou-se como base as equações definidas pelo modelo *FOWM*.

```
import numpy as np
import math
from matplotlib import pyplot as plt
from scipy.integrate import odeint
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from gym import Env
from gym.spaces import Discrete,Box
import random
import time
start_time = time.time()
```

Figura B.0.1: Definição das bibliotecas utilizadas no método AC.

```
#--------------------FOWM----------------------

# Variáveis Definidas
teta = 0.577 # (rad) = 56°
zinicial = 0.4 # Abertura inicial da válvula choke
z = zinicial # Abetura da válvula choke
Wgc = 1.0 # kg/s, massa de gás injetado no tubo

# Propriedades gases
g = 9.81 #m/s2
R = 8.3145 #J/mol.K

# Parâmetros-Caso 3-Poço A
Mlstill = 62.22
Cg = 0.001137
Cout = 0.002039
Veb = 60.98
E = 0.1545
Kw = 0.0006876
Ka = 0.00002293
Kr = 126.9
wu = 2.780

# Variáveis Poço A
rol = 900 #kg/m3
Pr = 22500000 #Pascal = 225 bar
Ps = 1000000 #Pascal = 10 bar
alphagw = 0.0188
romres = 892 #kg/m3
M = 0.018 #kg/mol
T = 298 #K
Lr = 1569 #m
Lfl = 2928 #m
Lt = 1638 #m
La = 1118 #m
Ht = 1279 #m
Hpdg = 1117 #m
Hvgl = 916 #m
Dss = 0.15 #m
Dt = 0.15 #m
Da = 0.14 #m

# Dimensões - Eq.Constantes
Ass = math.pi * math.pow(Dss, 2)/4
Vss = math.pi * math.pow(Dss, 2) * Lr/4 + math.pi * math.pow(Dss, 2) *
Lfl/4
Va = math.pi * math.pow(Da, 2) * La/4
Vt = math.pi * math.pow(Dt, 2) * Lt/4
```

Figura B.0.2: Definição dos parâmetros e equações constantes do modelo FOWM.

```
# Propriedades dos Fluidos
def alphagt(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return Mgt/(Mgt + Mlt)

def alphagr(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return Mgr/(Mgr + Mlr)

def Pai(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return (R*T/(Va*M) + g*La/Va)*Mga

def Prb(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return Prt(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) + (Mlr + Mlstill) * g *
math.sin(teta)/Ass

def Prt(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return Mgr * R * T/(M * (wu * Vss - (Mlr + Mlstill)/rol))

def Peb(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return Mgb * R * T/(M * Veb)

def romt(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return (Mgt + Mlt)/Vt

def Vgt(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return Vt - Mlt/rol

def rogt(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return Mgt/Vgt(Mga,Mgt,Mlt,Mgb,Mgr,Mlr)

def roai(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return M * Pai(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) / (R * T)

def Ptt(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return rogt(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) * R * T/M

def alphalr(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return 1 - alphagr(Mga,Mgt,Mlt,Mgb,Mgr,Mlr)

def Ptb(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return Ptt(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) +
romt(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) * g * Hvgl

def Pbh(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return Ppdg(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) + romres * g * (Ht - Hpdg)

def Ppdg(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return Ptb(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) + romres * g * (Hpdg - Hvgl)
```

Figura B.0.3: Equações das propriedades dos fluidos do modelo FOWM.

```
# Vazões dos Fluidos
def Wiv(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return Ka * np.sqrt(max(0,roai(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) *
(Pai(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) - Ptb(Mga,Mgt,Mlt,Mgb,Mgr,Mlr))))

def Wr(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return Kr * (1 - (0.2 * Pbh(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) / Pr) -
math.pow((0.8 * Pbh(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) / Pr), 2))

def Wwhg(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return Kw * np.sqrt(max(0, rol * (Ptt(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) -
Prb(Mga,Mgt,Mlt,Mgb,Mgr,Mlr)))) * alphagt(Mga,Mgt,Mlt,Mgb,Mgr,Mlr)

def Wwhl(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return Kw * np.sqrt(max(0,rol * (Ptt(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) -
Prb(Mga,Mgt,Mlt,Mgb,Mgr,Mlr)))) * (1 -
alphagt(Mga,Mgt,Mlt,Mgb,Mgr,Mlr))

def Wg(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return Cg * (Peb(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) -
Prb(Mga,Mgt,Mlt,Mgb,Mgr,Mlr))

def Wgout(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return alphagr(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) * Cout * z *
np.sqrt(max(0,rol * (Prt(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) - Ps)))

def Wlout(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return alphalr(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) * Cout * z *
np.sqrt(max(0,rol * (Prt(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) - Ps)))
```

Figura B.0.4: Equações de vazões dos fluidos do modelo FOWM.

```
# Balanço de Massa
def Dm(m,t):

    Dm = np.zeros(6)

    Mga = m[0]
    Mgt = m[1]
    Mlt = m[2]
    Mgb = m[3]
    Mgr = m[4]
    Mlr = m[5]

    Dmga = Wgc - Wiv(Mga,Mgt,Mlt,Mgb,Mgr,Mlr)
    Dmgt = Wr(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) * alphagw +
Wiv(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) - Wwhg(Mga,Mgt,Mlt,Mgb,Mgr,Mlr)
    Dmlt = Wr(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) * (1 - alphagw) -
Wwhl(Mga,Mgt,Mlt,Mgb,Mgr,Mlr)
    Dmgb = (1 - E) * Wwhg(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) -
Wg(Mga,Mgt,Mlt,Mgb,Mgr,Mlr)
    Dmgr = E * Wwhg(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) +
Wg(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) - Wgout(Mga,Mgt,Mlt,Mgb,Mgr,Mlr)
    Dmlr = Wwhl(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) -
Wlout(Mga,Mgt,Mlt,Mgb,Mgr,Mlr)

    Dm[0] = Dmga
    Dm[1] = Dmgt
    Dm[2] = Dmlt
    Dm[3] = Dmgb
    Dm[4] = Dmgr
    Dm[5] = Dmlr

    return Dm

# Condições iniciais:
m = np.zeros(6)
m[0] = 3372
m[1] = 1390
m[2] = 14674
m[3] = 6489
m[4] = 2620
m[5] = 27000
```

Figura B.0.5: Balanço de massa e condições iniciais do modelo *FOWM*.

```
# Resolução Balanço de Massa
tfinal = 36000 #tempo longo para estabilização do sistema pré-
otimização
t = np.linspace(0, tfinal, num=tfinal+1)
sol = odeint(Dm,m,t)

# Definindo novos valores de massa para início da otimização
m[0] = sol[tfinal,0]
m[1] = sol[tfinal,1]
m[2] = sol[tfinal,2]
m[3] = sol[tfinal,3]
m[4] = sol[tfinal,4]
m[5] = sol[tfinal,5]
```

Figura B.0.6: Resolução do balanço de massa e definição dos valores para início da otimização pelo método AC.

```
# ------------Reinforcement Learning Actor Critic------------------

# Definições para gráficos e análises
passoz = []
steppassoz = []
recompensatotal = []
recompensaaux = Wlout(sol[tfinal,0], sol[tfinal,1], sol[tfinal,2],
sol[tfinal,3], sol[tfinal,4], sol[tfinal,5])

Ppdginicial = Ppdg(sol[tfinal,0], sol[tfinal,1], sol[tfinal,2],
sol[tfinal,3], sol[tfinal,4], sol[tfinal,5])
Ppdgamp = 0
graficoppdg = []
graficoppdgepisodio = []
graficoppdgepisodioaux = []

graficowlout = []
graficowloutaux = []
graficowloutaux.append(recompensaaux)
graficowloutreward = []
```

Figura B.0.7: Definições para gráficos e análises do método AC.

```
# Ambiente em Gym do Período de Ações
class FOWM(Env):
    # Inicialização do Ambiente
    def __init__(self):
        global z
        global steppassoz
        global m
        global Ppdginicial
        steppassoz.append(zinicial)
        self.action_space = Discrete(171) #Espaço de ações
        self.observation_space = Box(low=np.array([0.02, 0]),
high=np.array([0.98,np.inf]), dtype=np.float32) #Espaço dos estados
        self.state = (z,Ppdginicial) #Definição do estado inicial

    # Passo - Ação
    def step(self, action):
        global z
        global m
        global passoz
        global steppassoz
        global recompensaaux
        global Ppdgamp
        global graficowlout
        global graficowloutaux

        z = z + ((action - 85)/500) #aplicando ação na choke

        descontoc = 6 #constante de supressão de movimento

        done = False #variável para parada antecipada, nao utilizado.

        if z <= 0.02:
            z = 0.02
        elif z >= 0.98:
                z = 0.98

        # Definição do vetor tempo de uma ação
        steptfinal = 60
        stept = np.linspace(0, steptfinal, num=steptfinal+1)
        stepsol = odeint(Dm,m,stept) #Resolvendo o modelo para a ação.

        # Aplicando os resultados às massas.
        m[0] = stepsol[steptfinal,0]
        m[1] = stepsol[steptfinal,1]
        m[2] = stepsol[steptfinal,2]
        m[3] = stepsol[steptfinal,3]
        m[4] = stepsol[steptfinal,4]
        m[5] = stepsol[steptfinal,5]
```

Figura B.0.8: Criação do ambiente *Gym* e resolução de uma ação do método AC.

```
        # Para visualização gráfica da vazão durante ações.
        graficowloutaux.append(Wlout(stepsol[steptfinal,0],
stepsol[steptfinal,1], stepsol[steptfinal,2], stepsol[steptfinal,3],
stepsol[steptfinal,4], stepsol[steptfinal,5]))

        # Cálculo da variação absoluta entre abertura no final da
atual e abertura no final da ação passada
        steppassoz.append(z)
        stepatual = len(steppassoz)-1
        stepdesconto = abs(steppassoz[stepatual] -
steppassoz[stepatual-1])

        # Cálculo da pressão no ponto PDG para definição do estado.
        Ppdgstep = Ppdg(stepsol[steptfinal,0], stepsol[steptfinal,1],
stepsol[steptfinal,2], stepsol[steptfinal,3], stepsol[steptfinal,4],
stepsol[steptfinal,5])

        # Cálculo da pressão no ponto PDG para possível desconto na
recompensa, não utilizado.
        Ppdgamp = (max(Ppdg(stepsol[:,0], stepsol[:,1], stepsol[:,2],
stepsol[:,3], stepsol[:,4], stepsol[:,5])) - min(Ppdg(stepsol[:,0],
stepsol[:,1], stepsol[:,2], stepsol[:,3], stepsol[:,4],
stepsol[:,5])))

        # Cálculo da produção para ser usada na recompensa e futura
visualização.
        i=0
        Wloutreward = np.zeros(steptfinal+1)
        while i <= steptfinal:
            Wloutreward[i] = Wlout(stepsol[i,0], stepsol[i,1],
stepsol[i,2], stepsol[i,3], stepsol[i,4], stepsol[i,5])
            i+=1

        #Cálculo da recompensa do episódio.
        if z <= 0.02:
            reward = 0 #penalidade por alcançar o limite inferior
        else:
            if z >= 0.98:
                reward = 0 #penalidade por alcançar o limite superior
            else:
                reward = (np.mean(Wloutreward) * steptfinal)*(1-
stepdesconto*descontoc) #recompensa do episódio.
```

Figura B.0.9: Cálculos após realização da ação da ação no ambiente *Gym* para método AC.

```
        # Para visualização e acompanhamento
        recompensaaux = Wlout(stepsol[steptfinal,0],
        stepsol[steptfinal,1], stepsol[steptfinal,2],
        stepsol[steptfinal,3], stepsol[steptfinal,4],
        stepsol[steptfinal,5])

        # Parâmetro info padrão de espaço em Gym.
        info = {}

        # Definição do novo estado.
        self.state = (z,Ppdgstep)

        return self.state, reward, done, info

    # Visualização por ação. Não utilizada.
    def render(self):
        pass

    # Reinicialização do sistema para iniciar o proxímo episódio.
    def reset(self):
        global z
        global sol
        global m
        global Ppdginicial
        m[0] = sol[tfinal,0]
        m[1] = sol[tfinal,1]
        m[2] = sol[tfinal,2]
        m[3] = sol[tfinal,3]
        m[4] = sol[tfinal,4]
        m[5] = sol[tfinal,5]
        z = zinicial
        self.state = (z,Ppdginicial)
        return self.state
```

Figura B.0.10: Final do passo e configurações de reinicialização do ambiente *Gym* para método AC.

```
# Configuração de parâmetros para todo o algoritmo
seed = random.seed()
gamma = 1  # Fator de desconto para recompensas passadas
max_steps_per_episode = 61 # Numero de passos
env = FOWM() # Criando o ambiente
env.seed(seed)
eps = np.finfo(np.float32).eps.item()  # Menor número para que 1.0 +
eps != 1.0

# Gerando a rede neural
num_inputs = 2 # Número de variáveis de estado
num_actions = 171 # Número de ações possíveis
num_hidden = 128 # Número de neurônios da rede

inputs = layers.Input(shape=(num_inputs,))
common = layers.Dense(num_hidden, activation="relu")(inputs)
action = layers.Dense(num_actions, activation="softmax")(common)
critic = layers.Dense(1)(common)

# Definindo parâmetros da rede neural e iterações
model = keras.Model(inputs=inputs, outputs=[action, critic])
optimizer = keras.optimizers.Adam(learning_rate=0.01)
huber_loss = keras.losses.Huber()
action_probs_history = []
critic_value_history = []
rewards_history = []
running_reward = 0
episode_count = 0
```

Figura B.0.11: Configuração de parâmetros do método AC.

```
# Início das iterações
while episode_count < 30000: # Número máximo de iterações
    state = env.reset() # Reinicialização do ambiente
    episode_reward = 0
    with tf.GradientTape() as tape:
        for timestep in range(1, max_steps_per_episode):

            # Passos referentes ao período de ações
            if timestep < max_steps_per_episode - 1:

                state = tf.convert_to_tensor(state)
                state = tf.expand_dims(state, 0)

                # Predição das probabilidades de ações
                # e estimativa das recompensas futuras
                # com base no estado atual do ambiente
                action_probs, critic_value = model(state)
                critic_value_history.append(critic_value[0, 0])

                # Seleciona a ação de acordo com a distribuição das
                probabilidades
                action = np.random.choice(num_actions,
                                p=np.squeeze(action_probs))

                action_probs_history.append(tf.math.log(action_probs[0
                , action]))

                # Aplica a ação no ambiente
                state, reward, done, _ = env.step(action)
                rewards_history.append(reward)
                episode_reward += reward
```

Figura B.0.12: Início das iterações e período de ações do método AC.

```
# Passo referente ao período de avaliação.
else:

    state = tf.convert_to_tensor(state)
    state = tf.expand_dims(state, 0)

    # Predição das probabilidades de ações e estimativa
   das recompensas futuras com base no estado atual do
   ambiente
    action_probs, critic_value = model(state)
    critic_value_history.append(critic_value[0, 0])

    # Definindo o tempo dos períodos pós-ações
    steptfinal = 32400
    stept = np.linspace(0, steptfinal, num=steptfinal+1)
    stepsol = odeint(Dm,m,stept) #Resolvendo o modelo para
                                          esses períodos.

    # Aplicando o resultado final as massas
    m[0] = stepsol[steptfinal,0]
    m[1] = stepsol[steptfinal,1]
    m[2] = stepsol[steptfinal,2]
    m[3] = stepsol[steptfinal,3]
    m[4] = stepsol[steptfinal,4]
    m[5] = stepsol[steptfinal,5]

    # Calculo da amplite da pressão no ponto PDG - 10800s
     é o inicio do periodo de avaliação
      Ppdgamp = (max(Ppdg(stepsol[10800:,0],
     stepsol[10800:,1], stepsol[10800:,2],
     stepsol[10800:,3], stepsol[10800:,4],
     stepsol[10800:,5])) - min(Ppdg(stepsol[10800:,0],
     stepsol[10800:,1], stepsol[10800:,2],
     stepsol[10800:,3], stepsol[10800:,4],
     stepsol[10800:,5])))

    # Para visualização

   graficoppdgepisodioaux.append(Ppdg(stepsol[:,0],
   stepsol[:,1], stepsol[:,2], stepsol[:,3], stepsol[:,4],
   stepsol[:,5]))
```

Figura B.0.13: Resolução do período de avaliação do método AC.

```
            # Cálculo da produção no período de avaliação
            ee = 10800
            Wloutreward = np.zeros(steptfinal+1)
            i=0
            while i <= steptfinal:
                Wloutreward[i]=Wlout(stepsol[i,0], stepsol[i,1],
                stepsol[i,2], stepsol[i,3], stepsol[i,4],
                stepsol[i,5])
                i+=1

            # Recompensa do período de avaliação
            reward = max(0,((np.mean(Wloutreward[ee:]) *
                             (steptfinal - ee)) - Ppdgamp))

            # Para visualização
            graficowloutreward.append(Wloutreward)

            # Salvando a recompensa no histórico
            rewards_history.append(reward)
            episode_reward += reward

        if done:
            graficowloutreward.append(np.zeros(1))
            break
```

Figura B.0.14: Cálculo da recompensa do período de avaliação do método AC.

```
        # Para visualização
        running_reward = episode_reward

        # Cálculo do valor esperado a partir das recompensas
        # - A cada passo, qual foi a recompensa total recebida depois
        daquele passo
        # - Recompensas antigas são descontadas multiplicando-as por
        gamma - Não utilizado - gamma = 1
        # - Esses são os rótulos para o crítico
        returns = []
        discounted_sum = 0
        for r in rewards_history[::-1]:
            discounted_sum = r + gamma * discounted_sum
            returns.insert(0, discounted_sum)

        # Normalização
        returns = np.array(returns)
        returns = (returns - np.mean(returns)) / (np.std(returns) +
                                                                eps)
        returns = returns.tolist()

        # Cálculo do valor perdido para atualizar nossa rede
        history = zip(action_probs_history, critic_value_history,
                                                            returns)
        actor_losses = []
        critic_losses = []
        for log_prob, value, ret in history:

            diff = ret - value
            actor_losses.append(-log_prob * diff)  # actor loss

            critic_losses.append(
                huber_loss(tf.expand_dims(value, 0),
                   tf.expand_dims(ret, 0))
                )

        # Backpropagation
        loss_value = sum(actor_losses) + sum(critic_losses)
        grads = tape.gradient(loss_value, model.trainable_variables)
        optimizer.apply_gradients(zip(grads,
                                      model.trainable_variables))

        # Limpeza do histórico de perdas e de recompensas
        action_probs_history.clear()
        critic_value_history.clear()
        rewards_history.clear()

    # Contagem de episódios e linhas para visualização
    episode_count += 1
```

Figura B.0.15: Atualização da política e final do episódio do método AC.

```
    if episode_count % 1 == 0:
        template = "running reward: {:.2f} at episode {}"
        print(template.format(running_reward, episode_count))
        print("Abertura Choke Final: %.2f"%z)
        print("Vazão Final: %.2f"%recompensaaux)
        recompensatotal = recompensatotal + [running_reward]
        passoz.append(steppassoz)
        steppassoz = []
        steppassoz.append(zinicial)
        graficoppdg.append(Ppdgamp)
        graficowlout.append(graficowloutaux)
        graficowloutaux = []
        graficoppdgepisodio.append(graficoppdgepisodioaux)
        graficoppdgepisodioaux = []

# Avalia qual foi o episódio de maior recompensa
maioretapa = recompensatotal.index(max(recompensatotal))

# Imprime a maior recompensa
print("Maior Recompensa: %.2f"%max(recompensatotal))
print("Episodio: %.2f"%(maioretapa+1))
```

Figura B.0.16: Acompanhamentos de final de episódio e considerações para geração de gráficos do método AC.

```
# Geração dos gráficos
episodiografico = np.linspace(1, len(graficoppdg),
num=len(graficoppdg))
plt.figure(1)
plt.plot(episodiografico,graficoppdg,'r',label='Amplitude Ppdg (Pa)')
plt.legend(loc='best')
plt.xlabel('Episódio')
plt.grid()
plt.show()

graficoppdgepi = 0
for i in graficoppdgepisodio[maioretapa]:
    graficoppdgepi += i

tempoee = np.linspace(0, len(graficoppdgepi), num =
len(graficoppdgepi))
plt.figure(2)
plt.plot(tempoee,graficoppdgepi,'y',label='Ppdg (Pa)')
plt.legend(loc='best')
plt.xlabel('Tempo (s)')
plt.grid()
plt.show()

acaografico = np.linspace(0, len(passoz[maioretapa]),
num=len(passoz[maioretapa]))
plt.figure(3)
plt.plot(acaografico,passoz[maioretapa],'b',label='Abertura Choke')
plt.legend(loc='best')
plt.xlabel('Ação')
plt.grid()
plt.show()

acaografico2 = np.linspace(0, len(graficowlout[maioretapa]),
num=len(graficowlout[maioretapa]))
plt.figure(4)
plt.plot(acaografico2,graficowlout[maioretapa],'g',label='Produção
Ação (kg/s)')
plt.legend(loc='best')
plt.xlabel('Ação')
plt.grid()
plt.show()

plt.figure(5)
plt.plot(tempoee,graficowloutreward[maioretapa],'g',label='Produção
Pós-Ação (kg/s)')
plt.legend(loc='best')
plt.xlabel('Tempo (s)')
plt.grid()
plt.show()
```

Figura B.0.17: Geração dos gráficos do método AC.

APÊNDICE C – *MINIMIZE COBYLA* EM *PYTHON*

Da Figura C.0.1 a Figura C.0.14 é explicitado o código utilizado em *Python* para aplicar o método *COBYLA* na otimização de produção de poços de petróleo. Utilizou-se como base as equações definidas pelo modelo *FOWM*.

```
import numpy as np
import math
from matplotlib import pyplot as plt
from scipy.integrate import odeint
from scipy.optimize import minimize
import time
start_time = time.time()
```

Figura C.0.1: Definição das bibliotecas utilizadas pelo método *COBYLA*.

```
#definições
teta = 0.577 #rad = 56°C
zinicial = 0.40
z = zinicial #abertura choke valve
Wgc = 1.0 #kg/s, massa de gás injetado no tubo

#propriedades gases
g = 9.81 #m/s2
R = 8.3145 #J/mol.K

#parâmetros caso 3 poço A
Mlstill = 62.22
Cg = 0.001137
Cout = 0.002039
Veb = 60.98
E = 0.1545
Kw = 0.0006876
Ka = 0.00002293
Kr = 126.9
wu = 2.780
```

Figura C.0.2: Definição de parâmetros do modelo *FOWM.*

```
#variáveis poço A
rol = 900 #kg/m3
Pr = 22500000 #Pascal = 225 bar
Ps = 1000000 #Pascal = 10 bar
alphagw = 0.0188
romres = 892 #kg/m3
M = 0.018 #kg/mol
T = 298 #K
Lr = 1569 #m
Lfl = 2928 #m
Lt = 1638 #m
La = 1118 #m
Ht = 1279 #m
Hpdg = 1117 #m
Hvgl = 916 #m
Dss = 0.15 #m
Dt = 0.15 #m
Da = 0.14 #m

#Dimensões - Eq.Constantes
Ass = math.pi * math.pow(Dss, 2)/4
Vss = math.pi * math.pow(Dss, 2) * Lr/4 + math.pi * math.pow(Dss, 2) *
Lfl/4
Va = math.pi * math.pow(Da, 2) * La/4
Vt = math.pi * math.pow(Dt, 2) * Lt/4
```

Figura C.0.3: Variáveis e equações de dimensionamento do modelo *FOWM*.

```
# Vazões dos Fluidos
def Wiv(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return Ka * np.sqrt(max(0,roai(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) *
(Pai(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) - Ptb(Mga,Mgt,Mlt,Mgb,Mgr,Mlr))))

def Wr(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return Kr * (1 - (0.2 * Pbh(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) / Pr) -
math.pow((0.8 * Pbh(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) / Pr), 2))

def Wwhg(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return Kw * np.sqrt(max(0, rol * (Ptt(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) -
Prb(Mga,Mgt,Mlt,Mgb,Mgr,Mlr)))) * alphagt(Mga,Mgt,Mlt,Mgb,Mgr,Mlr)

def Wwhl(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return Kw * np.sqrt(max(0,rol * (Ptt(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) -
Prb(Mga,Mgt,Mlt,Mgb,Mgr,Mlr)))) * (1 -
alphagt(Mga,Mgt,Mlt,Mgb,Mgr,Mlr))

def Wg(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return Cg * (Peb(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) -
Prb(Mga,Mgt,Mlt,Mgb,Mgr,Mlr))

def Wgout(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return alphagr(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) * Cout * z *
np.sqrt(max(0,rol * (Prt(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) - Ps)))

def Wlout(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return alphalr(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) * Cout * z *
np.sqrt(max(0,rol * (Prt(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) - Ps)))
```

Figura C.0.4: Equações de vazões dos fluidos do modelo *FOWM*.

```
# Propriedades dos Fluidos
def alphagt(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return Mgt/(Mgt + Mlt)

def alphagr(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return Mgr/(Mgr + Mlr)

def Pai(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return (R*T/(Va*M) + g*La/Va)*Mga

def Prb(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return Prt(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) + (Mlr + Mlstill) * g *
math.sin(teta)/Ass

def Prt(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return Mgr * R * T/(M * (wu * Vss - (Mlr + Mlstill)/rol))

def Peb(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return Mgb * R * T/(M * Veb)

def romt(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return (Mgt + Mlt)/Vt

def Vgt(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return Vt - Mlt/rol

def rogt(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return Mgt/Vgt(Mga,Mgt,Mlt,Mgb,Mgr,Mlr)

def roai(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return M * Pai(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) / (R * T)

def Ptt(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return rogt(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) * R * T/M

def alphalr(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return 1 - alphagr(Mga,Mgt,Mlt,Mgb,Mgr,Mlr)

def Ptb(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return Ptt(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) +
romt(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) * g * Hvgl

def Pbh(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return Ppdg(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) + romres * g * (Ht - Hpdg)

def Ppdg(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return Ptb(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) + romres * g * (Hpdg - Hvgl)
```

Figura C.0.5: Equações de propriedades dos fluidos do modelo *FOWM.*

```
# Balanço de Massa
def Dm(m,t):

    Dm = np.zeros(6)

    Mga = m[0]
    Mgt = m[1]
    Mlt = m[2]
    Mgb = m[3]
    Mgr = m[4]
    Mlr = m[5]

    Dmga = Wgc - Wiv(Mga,Mgt,Mlt,Mgb,Mgr,Mlr)
    Dmgt = Wr(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) * alphagw +
Wiv(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) - Wwhg(Mga,Mgt,Mlt,Mgb,Mgr,Mlr)
    Dmlt = Wr(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) * (1 - alphagw) -
Wwhl(Mga,Mgt,Mlt,Mgb,Mgr,Mlr)
    Dmgb = (1 - E) * Wwhg(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) -
Wg(Mga,Mgt,Mlt,Mgb,Mgr,Mlr)
    Dmgr = E * Wwhg(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) +
Wg(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) - Wgout(Mga,Mgt,Mlt,Mgb,Mgr,Mlr)
    Dmlr = Wwhl(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) -
Wlout(Mga,Mgt,Mlt,Mgb,Mgr,Mlr)

    Dm[0] = Dmga
    Dm[1] = Dmgt
    Dm[2] = Dmlt
    Dm[3] = Dmgb
    Dm[4] = Dmgr
    Dm[5] = Dmlr

    return Dm

# Condições iniciais:
m = np.zeros(6)
m[0] = 3372
m[1] = 1390
m[2] = 14674
m[3] = 6489
m[4] = 2620
m[5] = 27000
```

Figura C.0.6: Equações do balanço de massa e condições iniciais do modelo *FOWM.*

```
# Resolução Balanço de Massa
tfinal = 36000 #tempo longo para estabilização do sistema pré-
otimização
t = np.linspace(0, tfinal, num=tfinal+1)
sol = odeint(Dm,m,t)

# Definindo novos valores de massa para início da otimização
m[0] = sol[tfinal,0]
m[1] = sol[tfinal,1]
m[2] = sol[tfinal,2]
m[3] = sol[tfinal,3]
m[4] = sol[tfinal,4]
m[5] = sol[tfinal,5]
```

Figura C.0.7: Resolução inicial do modelo *FOWM*.

```
#-----------------Minimize COBYLA-----------------

# Definição de parâmetros e valor inicial
acoes = 60
delta = np.zeros(acoes)

# Definindo os limites das variáveis
b = (-0.17,0.17)
bnds = []
i = 0
while i < acoes:
    bnds.append(b)
    i+=1
```

Figura C.0.8: Parâmetros do método *COBYLA*.

```
# Definindo a função a ser minimizada
def objective(delta):
    global z
    global m

    Wloutrewardtotal = []

    descontoc = 0.5 # Constante de supressão

    # Aplicação o vetor de ações
    i=0
    while i < acoes:
        z = z + delta[i]
        # z = delta[i]
        if z > 0.99:
            z = 0.99
        else:
            if z < 0.01:
                z = 0.01

        # Resolvendo o modelo para cada ação
        steptfinal = 60
        stept = np.linspace(0, steptfinal, num=steptfinal+1)
        stepsol = odeint(Dm,m,stept)

        # Definindo novo balanço
        m[0] = stepsol[steptfinal,0]
        m[1] = stepsol[steptfinal,1]
        m[2] = stepsol[steptfinal,2]
        m[3] = stepsol[steptfinal,3]
        m[4] = stepsol[steptfinal,4]
        m[5] = stepsol[steptfinal,5]

        # Calculando a recompensa da ação
        k=0
        Wloutreward = np.zeros(steptfinal+1)
        while k <= steptfinal:
            Wloutreward[k]=Wlout(stepsol[k,0], stepsol[k,1],
stepsol[k,2], stepsol[k,3], stepsol[k,4], stepsol[k,5])
            k+=1
        Wloutrewardtotal.append(np.mean(Wloutreward)*steptfinal)
        i+=1
```

Figura C.0.9: Início da função objetivo a ser minimizada pelo método *COBYLA* e período de ações.

```
# Período Pós Ações
    steptfinal = 32400 # Período de folga + período de avaliação
    stept = np.linspace(0, steptfinal, num=steptfinal+1)
    stepsol = odeint(Dm,m,stept) # Resolvendo o modelo para o todo o
período.

    # Aplicando os valores finais
    m[0] = stepsol[steptfinal,0]
    m[1] = stepsol[steptfinal,1]
    m[2] = stepsol[steptfinal,2]
    m[3] = stepsol[steptfinal,3]
    m[4] = stepsol[steptfinal,4]
    m[5] = stepsol[steptfinal,5]

    # Cálculo da amplitude de pressão no ponto PDG
    # 10800s = Início do período de avaliação
    Ppdgamp = (max(Ppdg(stepsol[10800:,0], stepsol[10800:,1],
stepsol[10800:,2], stepsol[10800:,3], stepsol[10800:,4],
stepsol[10800:,5])) - min(Ppdg(stepsol[10800:,0], stepsol[10800:,1],
stepsol[10800:,2], stepsol[10800:,3], stepsol[10800:,4],
stepsol[10800:,5])))

    # Cálculo da recompensa do período de avaliação
    ee=10800
    k=0
    Wloutreward = np.zeros(steptfinal+1-ee)
    while k <= (steptfinal - ee):
        Wloutreward[k] = Wlout(stepsol[k,0], stepsol[k,1],
stepsol[k,2], stepsol[k,3], stepsol[k,4], stepsol[k,5])
        k+=1
    Wloutrewardtotal.append(np.mean(Wloutreward)*(steptfinal-ee))
```

Figura C.0.10: Período pós ações da função objetivo a ser minimizada pelo método *COBYLA.*

```
# Cálculo da recompensa total
    recompensatotal = 0
    for i in Wloutrewardtotal:
        recompensatotal += i

    # Cálculo do somátorio das variações absolutas
    descontoz = 0
    for x in delta:
        descontoz = descontoz + abs(x)

    # Ajuste da recompensa total
    recompensatotal = (recompensatotal - Ppdgamp) * (1 - descontoz *
descontoc)

    # Para acompanhamento
    print(recompensatotal)

    # Reiniciando o sistema
    z = zinicial
    m[0] = sol[tfinal,0]
    m[1] = sol[tfinal,1]
    m[2] = sol[tfinal,2]
    m[3] = sol[tfinal,3]
    m[4] = sol[tfinal,4]
    m[5] = sol[tfinal,5]

    return -recompensatotal
```

Figura C.0.11: Cálculo da recompensa total e retorno da função objetivo a ser minimizada pelo método *COBYLA.*

```
# Definição dos limites de cada variável por restrição
# Particularidade do COBYLA
cons = []
for factor in range(len(bnds)):
    lower, upper = bnds[factor]
    l = {'type': 'ineq',
          'fun': lambda delta, lb=lower, i=factor: delta[i] - lb}
    u = {'type': 'ineq',
          'fun': lambda delta, ub=upper, i=factor: ub - delta[i]}
    cons.append(l)
    cons.append(u)

# Minimizando a função objetivo
sol = minimize(objective,delta,method='COBYLA',constraints=cons,
options={'rhobeg': 0.05, 'maxiter': 20000, 'disp': False, 'catol':
0.0})

# Visualização da resposta
print(sol)
```

Figura C.0.12: Limitação das variáveis e chamada da otimização pelo método *COBYLA.*

```
# Configuração dos vetores para gráficos a partir da solução
graficoacoes = np.linspace(0, acoes, num=acoes)
graficosol = []
graficoppdg = []
graficowlout = []

z = zinicial
for i in sol.x:
    z += i

    steptfinal = 60
    stept = np.linspace(0, steptfinal, num=steptfinal+1)
    stepsol = odeint(Dm,m,stept)
    m[0] = stepsol[steptfinal,0]
    m[1] = stepsol[steptfinal,1]
    m[2] = stepsol[steptfinal,2]
    m[3] = stepsol[steptfinal,3]
    m[4] = stepsol[steptfinal,4]
    m[5] = stepsol[steptfinal,5]

    graficoppdg.append(Ppdg(stepsol[steptfinal,0],
stepsol[steptfinal,1], stepsol[steptfinal,2], stepsol[steptfinal,3],
stepsol[steptfinal,4], stepsol[steptfinal,5]))
    graficowlout.append(Wlout(stepsol[steptfinal,0],
stepsol[steptfinal,1], stepsol[steptfinal,2], stepsol[steptfinal,3],
stepsol[steptfinal,4], stepsol[steptfinal,5]))
    graficosol.append(z)

steptfinal = 36000
stept = np.linspace(0, steptfinal, num=steptfinal+1)
stepsol = odeint(Dm,m,stept)

m[0] = stepsol[steptfinal,0]
m[1] = stepsol[steptfinal,1]
m[2] = stepsol[steptfinal,2]
m[3] = stepsol[steptfinal,3]
m[4] = stepsol[steptfinal,4]
m[5] = stepsol[steptfinal,5]

graficoppdgpos = Ppdg(stepsol[:,0], stepsol[:,1], stepsol[:,2],
stepsol[:,3], stepsol[:,4], stepsol[:,5])
graficowloutpos = []
i=0
while i <= steptfinal:
    graficowloutpos.append(Wlout(stepsol[i,0], stepsol[i,1],
stepsol[i,2], stepsol[i,3], stepsol[i,4], stepsol[i,5]))
    i+=1
graficopos = np.linspace(0, steptfinal, num=steptfinal+1)
```

Figura C.0.13: Configuração dos vetores para gráficos a partir da solução do método *COBYLA.*

```
# Geração dos gráficos
plt.figure(1)
plt.plot(graficoacoes,graficosol,'r',label='Abertura Choke(%)')
plt.legend(loc='best')
plt.xlabel('Ação')
plt.grid()
plt.show()

plt.figure(2)
plt.plot(graficoacoes,graficowlout,'g',label='Produção Ação(kg/s)')
plt.legend(loc='best')
plt.xlabel('Ação')
plt.grid()
plt.show()

plt.figure(3)
plt.plot(graficopos,graficowloutpos,'g',label='Produção Pós-
Ação(kg/s)')
plt.legend(loc='best')
plt.xlabel('Tempo(s)')
plt.grid()
plt.show()

plt.figure(4)
plt.plot(graficoacoes,graficoppdg,'b',label='Ppdg Ação(Pa)')
plt.legend(loc='best')
plt.xlabel('Ação')
plt.grid()
plt.show()

plt.figure(5)
plt.plot(graficopos,graficoppdgpos,'b',label='Ppdg Pós-Ação(Pa)')
plt.legend(loc='best')
plt.xlabel('Tempo(s)')
plt.grid()
plt.show()

# Impressão do tempo de execução
print("--- %s seconds ---" % (time.time() - start_time))
```

Figura C.0.14: Geração dos gráficos do método *COBYLA.*

APÊNDICE D – *DIFFERENTIAL EVOLUTION* EM *PYTHON*

Da Figura D.0.1 a Figura D.0.14 é explicitado o código utilizado em *Python* para aplicar o método *Differential Evolution* (DE) na otimização de produção de poços de petróleo. Utilizou-se como base as equações definidas pelo modelo *FOWM*.

```
import numpy as np
import math
from matplotlib import pyplot as plt
from scipy.integrate import odeint
from scipy.optimize import differential_evolution
import time
start_time = time.time()
```

Figura D.0.1: Definição das bibliotecas para o método DE.

```
#definições
teta = 0.577 #rad = 56°C
zinicial = 0.40
z = zinicial #abertura choke valve
Wgc = 1.0 #kg/s, massa de gás injetado no tubo

#propriedades gases
g = 9.81 #m/s2
R = 8.3145 #J/mol.K

#parâmetros caso 3 poço A
Mlstill = 62.22
Cg = 0.001137
Cout = 0.002039
Veb = 60.98
E = 0.1545
Kw = 0.0006876
Ka = 0.00002293
Kr = 126.9
wu = 2.780
```

Figura D.0.2: Definição de parâmetros para o modelo *FOWM.*

```
#variáveis poço A
rol = 900 #kg/m3
Pr = 22500000 #Pascal = 225 bar
Ps = 1000000 #Pascal = 10 bar
alphagw = 0.0188
romres = 892 #kg/m3
M = 0.018 #kg/mol
T = 298 #K
Lr = 1569 #m
Lfl = 2928 #m
Lt = 1638 #m
La = 1118 #m
Ht = 1279 #m
Hpdg = 1117 #m
Hvgl = 916 #m
Dss = 0.15 #m
Dt = 0.15 #m
Da = 0.14 #m

#Dimensões - Eq.Constantes
Ass = math.pi * math.pow(Dss, 2)/4
Vss = math.pi * math.pow(Dss, 2) * Lr/4 + math.pi * math.pow(Dss, 2) *
Lfl/4
Va = math.pi * math.pow(Da, 2) * La/4
Vt = math.pi * math.pow(Dt, 2) * Lt/4
```

Figura D.0.3: Definição de variáveis e equações constantes do modelo *FOWM.*

```
# Vazões dos Fluidos
def Wiv(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return Ka * np.sqrt(max(0,roai(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) *
(Pai(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) - Ptb(Mga,Mgt,Mlt,Mgb,Mgr,Mlr))))

def Wr(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return Kr * (1 - (0.2 * Pbh(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) / Pr) -
math.pow((0.8 * Pbh(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) / Pr), 2))

def Wwhg(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return Kw * np.sqrt(max(0, rol * (Ptt(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) -
Prb(Mga,Mgt,Mlt,Mgb,Mgr,Mlr)))) * alphagt(Mga,Mgt,Mlt,Mgb,Mgr,Mlr)

def Wwhl(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return Kw * np.sqrt(max(0,rol * (Ptt(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) -
Prb(Mga,Mgt,Mlt,Mgb,Mgr,Mlr)))) * (1 -
alphagt(Mga,Mgt,Mlt,Mgb,Mgr,Mlr))

def Wg(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return Cg * (Peb(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) -
Prb(Mga,Mgt,Mlt,Mgb,Mgr,Mlr))

def Wgout(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return alphagr(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) * Cout * z *
np.sqrt(max(0,rol * (Prt(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) - Ps)))

def Wlout(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return alphalr(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) * Cout * z *
np.sqrt(max(0,rol * (Prt(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) - Ps)))
```

Figura D.0.4: Equações de vazões dos fluidos para o modelo *FOWM.*

```
# Propriedades dos Fluidos
def alphagt(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return Mgt/(Mgt + Mlt)

def alphagr(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return Mgr/(Mgr + Mlr)

def Pai(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return (R*T/(Va*M) + g*La/Va)*Mga

def Prb(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return Prt(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) + (Mlr + Mlstill) * g *
math.sin(teta)/Ass

def Prt(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return Mgr * R * T/(M * (wu * Vss - (Mlr + Mlstill)/rol))

def Peb(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return Mgb * R * T/(M * Veb)

def romt(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return (Mgt + Mlt)/Vt

def Vgt(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return Vt - Mlt/rol

def rogt(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return Mgt/Vgt(Mga,Mgt,Mlt,Mgb,Mgr,Mlr)

def roai(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return M * Pai(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) / (R * T)

def Ptt(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return rogt(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) * R * T/M

def alphalr(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return 1 - alphagr(Mga,Mgt,Mlt,Mgb,Mgr,Mlr)

def Ptb(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return Ptt(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) +
romt(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) * g * Hvgl

def Pbh(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return Ppdg(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) + romres * g * (Ht - Hpdg)

def Ppdg(Mga,Mgt,Mlt,Mgb,Mgr,Mlr):
    return Ptb(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) + romres * g * (Hpdg - Hvgl)
```

Figura D.0.5: Equações de propriedade dos fluidos para o modelo *FOWM.*

```
# Balanço de Massa
def Dm(m,t):

    Dm = np.zeros(6)

    Mga = m[0]
    Mgt = m[1]
    Mlt = m[2]
    Mgb = m[3]
    Mgr = m[4]
    Mlr = m[5]

    Dmga = Wgc - Wiv(Mga,Mgt,Mlt,Mgb,Mgr,Mlr)
    Dmgt = Wr(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) * alphagw +
Wiv(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) - Wwhg(Mga,Mgt,Mlt,Mgb,Mgr,Mlr)
    Dmlt = Wr(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) * (1 - alphagw) -
Wwhl(Mga,Mgt,Mlt,Mgb,Mgr,Mlr)
    Dmgb = (1 - E) * Wwhg(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) -
Wg(Mga,Mgt,Mlt,Mgb,Mgr,Mlr)
    Dmgr = E * Wwhg(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) +
Wg(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) - Wgout(Mga,Mgt,Mlt,Mgb,Mgr,Mlr)
    Dmlr = Wwhl(Mga,Mgt,Mlt,Mgb,Mgr,Mlr) -
Wlout(Mga,Mgt,Mlt,Mgb,Mgr,Mlr)

    Dm[0] = Dmga
    Dm[1] = Dmgt
    Dm[2] = Dmlt
    Dm[3] = Dmgb
    Dm[4] = Dmgr
    Dm[5] = Dmlr

    return Dm

# Condições iniciais:
m = np.zeros(6)
m[0] = 3372
m[1] = 1390
m[2] = 14674
m[3] = 6489
m[4] = 2620
m[5] = 27000
```

Figura D.0.6: Balanço de massa e definição das condições iniciais do modelo *FOWM*.

```
    # Resolução Balanço de Massa
    tfinal = 36000 #tempo longo para estabilização do sistema pré-
    otimização
    t = np.linspace(0, tfinal, num=tfinal+1)
    sol = odeint(Dm,m,t)

    # Definindo novos valores de massa para início da otimização
    m[0] = sol[tfinal,0]
    m[1] = sol[tfinal,1]
    m[2] = sol[tfinal,2]
    m[3] = sol[tfinal,3]
    m[4] = sol[tfinal,4]
    m[5] = sol[tfinal,5]
```

Figura D.0.7: Resolução do modelo *FOWM* previamente à aplicação da otimização.

```
#---------------Differential Evolution----------------

# Definição de ações e valor inicial
acoes = 60
delta = np.zeros(acoes)

# Definindo os limites de cada ação
b = (-0.17,0.17)
bnds = []
i = 0
while i < acoes:
    bnds.append(b)
    i+=1
```

Figura D.0.8: Definição do vetor de ações e limites para o método DE.

```
# Definição da função a ser minimizada
def objective(delta):
    global z
    global m

    Wloutrewardtotal = []

    descontoc = 0.04 # Constante de supressão

    # Aplicação das ações
    i=0
    while i < acoes:
        z = z + delta[i]
        if z >= 0.98:
            z = 0.98
        elif z <= 0.02:
            z = 0.02

        # Resolvendo o modelo para cada ação
        steptfinal = 60
        stept = np.linspace(0, steptfinal, num=steptfinal+1)
        stepsol = odeint(Dm,m,stept)

        m[0] = stepsol[steptfinal,0]
        m[1] = stepsol[steptfinal,1]
        m[2] = stepsol[steptfinal,2]
        m[3] = stepsol[steptfinal,3]
        m[4] = stepsol[steptfinal,4]
        m[5] = stepsol[steptfinal,5]

        # Cálculo da recompensa para cada ação
        k=0
        Wloutreward = np.zeros(steptfinal+1)
        while k <= steptfinal:
            Wloutreward[k]=Wlout(stepsol[k,0], stepsol[k,1],
stepsol[k,2], stepsol[k,3], stepsol[k,4], stepsol[k,5])
            k+=1
        Wloutrewardtotal.append(np.mean(Wloutreward)*steptfinal)
        i+=1
```

Figura D.0.9: Definição da função a ser minimizada e cálculo da recompensa para o período de ações do método DE.

```
# Resolvendo o período pós ações
    steptfinal = 32400
    stept = np.linspace(0, steptfinal, num=steptfinal+1)
    stepsol = odeint(Dm,m,stept)

    m[0] = stepsol[steptfinal,0]
    m[1] = stepsol[steptfinal,1]
    m[2] = stepsol[steptfinal,2]
    m[3] = stepsol[steptfinal,3]
    m[4] = stepsol[steptfinal,4]
    m[5] = stepsol[steptfinal,5]

    # Cálculo da amplitude de pressão no ponto PDG
    # 10800s = Início do período de avaliação
    Ppdgamp = (max(Ppdg(stepsol[10800:,0], stepsol[10800:,1],
stepsol[10800:,2], stepsol[10800:,3], stepsol[10800:,4],
stepsol[10800:,5])) - min(Ppdg(stepsol[10800:,0], stepsol[10800:,1],
stepsol[10800:,2], stepsol[10800:,3], stepsol[10800:,4],
stepsol[10800:,5])))

    # Cálculo da recompensa para o período de avaliação
    ee=10800
    k=0
    Wloutreward = np.zeros(steptfinal+1-ee)
    while k <= (steptfinal - ee):
        Wloutreward[k] = Wlout(stepsol[k,0], stepsol[k,1],
stepsol[k,2], stepsol[k,3], stepsol[k,4], stepsol[k,5])
        k+=1
    Wloutrewardtotal.append(np.mean(Wloutreward)*(steptfinal-ee))
```

Figura D.0.10: Resolução e cálculo da recompensa do período pós ações para o método DE.

```
# Cálculo da recompensa total
    recompensatotal = 0
    for i in Wloutrewardtotal:
        recompensatotal += i

    # Cálculo do somatório das variações aboslutas
    descontoz = 0
    for x in delta:
        descontoz = descontoz + abs(x)

    # Ajuste da recompensa total
    recompensatotal = (recompensatotal - Ppdgamp) * (1 - descontoz *
descontoc)

    # Para acompanhamento
    print(recompensatotal)

    # Reinicialização do sistema
    z = zinicial
    m[0] = sol[tfinal,0]
    m[1] = sol[tfinal,1]
    m[2] = sol[tfinal,2]
    m[3] = sol[tfinal,3]
    m[4] = sol[tfinal,4]
    m[5] = sol[tfinal,5]

    return -recompensatotal
```

Figura D.0.11: Cálculo da recompensa total e retorno da função objetivo do método DE.

```
# Minimização da função objetivo
sol = differential_evolution(objective, bounds=bnds, popsize = 2,
maxiter = 200, seed = 123, polish=False, init = 'random', disp=True)

# Visualização da resposta
print(sol)
```

Figura D.0.12: Chamada do método DE e visualização da solução.

```
# Configuração dos vetores para geração de gráficos a partir da
solução
graficoacoes = np.linspace(0, acoes, num=acoes)
graficosol = []
graficoppdg = []
graficowlout = []
z = zinicial
for i in sol.x:
    z += i
    if z >= 0.98:
            z = 0.98
    elif z <= 0.02:
            z = 0.02
    steptfinal = 60
    stept = np.linspace(0, steptfinal, num=steptfinal+1)
    stepsol = odeint(Dm,m,stept)
    m[0] = stepsol[steptfinal,0]
    m[1] = stepsol[steptfinal,1]
    m[2] = stepsol[steptfinal,2]
    m[3] = stepsol[steptfinal,3]
    m[4] = stepsol[steptfinal,4]
    m[5] = stepsol[steptfinal,5]
    graficoppdg.append(Ppdg(stepsol[steptfinal,0],
ol[steptfinal,1], stepsol[steptfinal,2], stepsol[steptfinal,3],
ol[steptfinal,4], stepsol[steptfinal,5]))
    graficowlout.append(Wlout(stepsol[steptfinal,0],
ol[steptfinal,1], stepsol[steptfinal,2], stepsol[steptfinal,3],
ol[steptfinal,4], stepsol[steptfinal,5]))
    graficosol.append(z)
steptfinal = 36000
stept = np.linspace(0, steptfinal, num=steptfinal+1)
stepsol = odeint(Dm,m,stept)
m[0] = stepsol[steptfinal,0]
m[1] = stepsol[steptfinal,1]
m[2] = stepsol[steptfinal,2]
m[3] = stepsol[steptfinal,3]
m[4] = stepsol[steptfinal,4]
m[5] = stepsol[steptfinal,5]

graficoppdgpos = Ppdg(stepsol[:,0], stepsol[:,1], stepsol[:,2],
stepsol[:,3], stepsol[:,4], stepsol[:,5])
graficowloutpos = []
i=0
while i <= steptfinal:
    graficowloutpos.append(Wlout(stepsol[i,0], stepsol[i,1],
stepsol[i,2], stepsol[i,3], stepsol[i,4], stepsol[i,5]))
    i+=1
graficopos = np.linspace(0, steptfinal, num=steptfinal+1)
```

Figura D.0.13: Configuração dos vetores para geração dos gráficos com base na solução do método DE.

```
# Geração dos gráficos
plt.figure(1)
plt.plot(graficoacoes,graficosol,'r',label='Abertura Choke(%)')
plt.legend(loc='best')
plt.xlabel('Ação')
plt.grid()
plt.show()

plt.figure(2)
plt.plot(graficoacoes,graficowlout,'g',label='Produção Ação(kg/s)')
plt.legend(loc='best')
plt.xlabel('Ação')
plt.grid()
plt.show()

plt.figure(3)
plt.plot(graficopos,graficowloutpos,'g',label='Produção Pós-
Ação(kg/s)')
plt.legend(loc='best')
plt.xlabel('Tempo(s)')
plt.grid()
plt.show()

plt.figure(4)
plt.plot(graficoacoes,graficoppdg,'b',label='Ppdg Ação(Pa)')
plt.legend(loc='best')
plt.xlabel('Ação')
plt.grid()
plt.show()

plt.figure(5)
plt.plot(graficopos,graficoppdgpos,'b',label='Ppdg Pós-Ação(Pa)')
plt.legend(loc='best')
plt.xlabel('Tempo(s)')
plt.grid()
plt.show()

# Impressão do tempo de execução
print("--- %s seconds ---" % (time.time() - start_time))
```

Figura D.0.14: Geração dos gráficos e visualização do tempo de execução para o método DE.