DESENVOLVIMENTO DE UMA INTERFACE GRÁFICA PARA SUPORTE AO ENSINO DE FLUXO DE POTÊNCIA – POWER GRID DESIGN

DOI: 10.37702/2175-957X.COBENGE.2025.6416

Autores: *STHEFANO SOARES SCHIAVON,GUSTAVO ALVES,JOSé UBIRAJARA NúñEZ DE NUNES*

Resumo: *A modelagem e análise de sistemas de potência são temas recorrentes na graduação em Engenharia Elétrica. O fluxo de potência, essencial para análise em regime permanente, permite determinar tensões nas barras e fluxos de potência ativa e reativa nas linhas. Seu domínio é crucial em aplicações como planejamento, expansão e estabilidade. Embora existam simuladores disponíveis no mercado, muitos possuem interfaces pouco didáticas para iniciantes. Para suprir essa limitação, este trabalho propõe o Power Grid Design, uma ferramenta didática desenvolvida em Python, gratuita e baseada no método de Newton-Raphson. Com interface gráfica intuitiva e saídas tabulares, facilita a inserção de dados e a interpretação dos resultados. O desenvolvimento do software em sala de aula promove a construção do conhecimento por meio da aplicação prática, alinhando-se à metodologia de aprendizagem baseada em projetos.*

Palavras-chave: *Fluxo de potência,Método de Newton-Raphson,Aprendizagem baseada em projetos*

DESENVOLVIMENTO DE UMA INTERFACE GRÁFICA PARA SUPORTE AO ENSINO DE FLUXO DE POTÊNCIA – *POWER GRID DESIGN*

1 INTRODUÇÃO

A modelagem e análise de sistemas elétricos de potência (SEP) são tópicos recorrentes nas disciplinas de graduação em engenharia elétrica. Dentro desse contexto, destaca-se o fluxo de potência como uma ferramenta fundamental para análise de sistemas elétricos em regime permanente. Através dela, é possível determinar as tensões complexas em todas as barras de um sistema, representando o estado da rede, e a distribuição dos fluxos de potências ativa e reativa em seus circuitos.

O cálculo do fluxo de potência envolve a solução de equações não-lineares por meio de métodos iterativos, dentre os quais, destaca-se o método de Newton-Raphson. Este método, associado a técnicas de decomposição matricial para sistemas de equações algébricas lineares, é amplamente aceito na literatura para a solução do fluxo de potência em sistemas de grande porte (MONTICELLI, 1983).

Dentre os programas computacionais dedicados a análise de sistemas de potência, destaca-se o ANAREDE como um *software* voltado para análises de fluxo de potência e análises de contingência (ELETROBRÁS, 2025). O programa possui versão estudantil, mas a sua entrada de dados manualmente é trabalhosa e a definição dos parâmetros de simulação bem como a conversão dos resultados numéricos em gráficos não são tarefas simples.

Outro *software* amplamente reconhecido pela comunidade científica é o MATPOWER, em código aberto e desenvolvido em MATLAB, é eficiente para simulações e otimizações de sistema de potência. Porém, a sua interface baseada apenas em matrizes dificulta a interação e a interpretação dos resultados (ZIMMERMAN; MURILLO-SANCHEZ; THOMAS, 2011).

Diante das limitações dos simuladores tradicionais no ensino, neste trabalho é apresentado o *Power Grid Design* (PGD), uma ferramenta didática para análise de fluxo de potência baseada no método de Newton-Raphson e desenvolvida em linguagem Python (THURNER *et al*., 2018). Gratuita e com licença GNU GPL[1], a ferramenta possui interface gráfica intuitiva com diagrama unifilar e tabelas interativas. Os dados de entrada são validados e atualizados em tempo real. O *software* foi validado por meio de um estudo comparativo com o ANAREDE, usando o sistema IEEE de 14 barras (UNIVERSITY OF WASHINGTON, 1993), comprovando sua precisão e potencial como ferramenta educacional.

A proposta de desenvolvimento do *software* PGD foi apresentada como um desafio aos alunos do sétimo semestre do curso de Engenharia Elétrica, cursando a disciplina de Sistemas de Energia. Ela coloca o aluno como protagonista, estimulando a aplicação prática de conceitos trabalhados na disciplina através da implementação de um *software* voltado para análise de fluxo de potência, promovendo autonomia e pensamento crítico (OLIVEIRA; SANTOS, 2019). Deste modo, atende-se os requisitos da aprendizagem baseada em projetos, utilizando-se um tempo compatível com a carga horária de um curso generalista de Engenharia Elétrica.

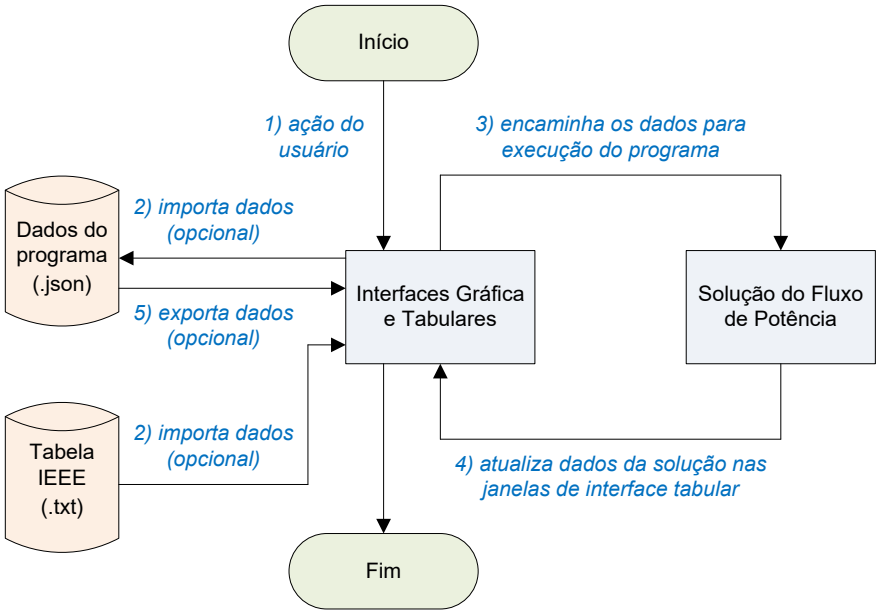
[1] GNU GPL significa GNU *General Public License* (Licença Pública Geral GNU). É uma licença de *software* livre criada pela *Free Software Foundation* (FSF), inicialmente escrita por Richard Stallman para o projeto GNU.

2 O *SOFTWARE* PGD

O *software* PGD foi desenvolvido em Python, com a ajuda da biblioteca gráfica PySide6, para oferecer suporte aos principais sistemas operacionais da atualidade. Assim, qualquer usuário com o ambiente de desenvolvimento configurado consegue fazer alterações e/ou executar o programa. A sua solução baseada no método de Newton-Raphson, anteriormente descrito, pode ser facilmente atualizada ou substituída por outro método de solução.

Na Figura 1 é apresentado um fluxograma que ilustra as principais funcionalidades do *software* PGD. O processo se inicia com a abertura do programa. Em seguida, o usuário pode inserir ou modificar os dados da rede elétrica por meio das **Interfaces Gráfica e Tabulares**, que são os ambientes interativos do *software*. Alternativamente, o usuário pode importar dados a partir de arquivos ".json" contendo dados do programa ou, a partir de arquivos ".txt" com tabelas IEEE, além de exportar dados para arquivos ".json". Essa funcionalidade proporciona uma maneira simples e eficiente de inserir, visualizar, importar e exportar dados, sendo o principal diferencial do programa. Por meio dessas interfaces os dados são encaminhados ao módulo de **Solução do Fluxo de Potência**, que executa o programa retornando à solução na saída. A seguir, os dados de saída (tensões nas barras) são atualizados na janela de interface tabular de barras do sistema e o processo é finalizado.

Figura 1 - Principais funcionalidades disponíveis



Fonte: Os autores.

Embora o projeto seja uma base de código única, há uma abstração entre quem é responsável pelos dados e quem é responsável pela solução matemática. Como pode ser visto na Figura 1, a interface gráfica lida com toda a entrada e saída de dados, seja importação/exportação de arquivos ou interações feitas pelo usuário, através de janelas de interface gráfica e tabulares. Quando o usuário executa o fluxo de potência, as informações atuais do programa são encaminhadas à rotina que é responsável pela solução numérica. Após a conclusão, a interface gráfica é atualizada com os dados da resposta.

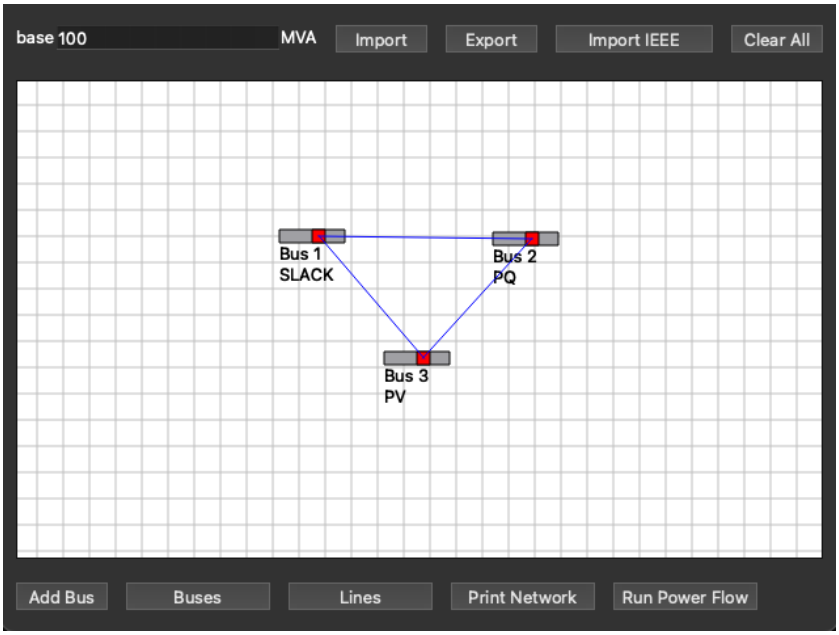
2.1 Interfaces Gráfica e Tabulares

Na Figura 2 é apresentada a interface gráfica do *software* PGD, tomando como exemplo o sistema de três barras apresentado em Saadat (1999). Ela permite que o usuário insira primeiramente as barras do sistema e, em seguida, insira as linhas de transmissão, clicando-se com o mouse em uma barra e arrastando até a outra. Cada vez que é inserido um novo componente (barra ou linha), uma janela correspondente é atualizada com um campo para preenchimento dos dados do componente. As janelas com os dados das barras e com os dados das linhas para o sistema exemplo de três barras são mostradas nas Figuras 4 e 5, respectivamente. Como pode ser visto na Figura 3, a janela tabular das barras possui como especificações o nome, o número, o tipo, a magnitude da tensão, o ângulo da tensão, a potência ativa e a potência reativa de cada barra. Na Figura 4 é mostrada a janela tabular das linhas, que apresenta como especificações a barra de origem, a barra final, a resistência, a reatância indutiva, a condutância e a susceptância *shunt*.

Sempre que for modificado o texto de algum campo de entrada e o usuário clicar fora do mesmo, o programa atualizará os dados do componente. Em seguida, o valor atual é verificado. Se ele for validado, o programa atualiza cada elemento da interface em que o próprio é exibido. Caso contrário, o texto volta para o estado em que estava anteriormente. Mudanças válidas nos campos das janelas atualizam o diagrama unifilar em tempo real. Isso ocorre sempre que um campo de texto é editado e perde o foco do mouse. Essa escolha de *design* torna dispensável o uso de botões para salvar. Conforme descrito anteriormente, dados inválidos são desprezados, fazendo com que o campo volte para seu valor anterior.

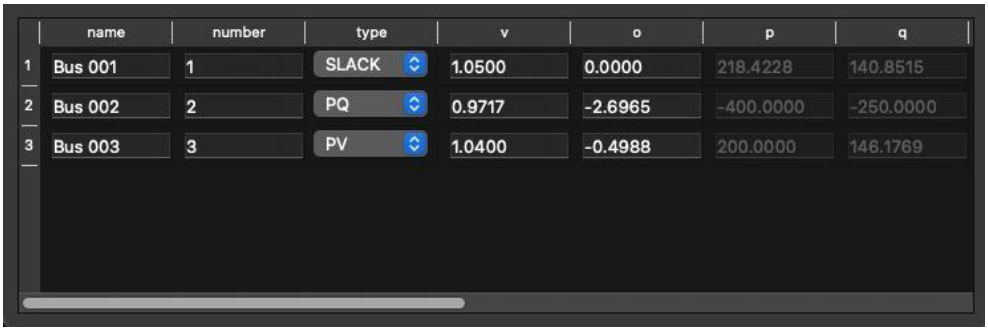
Alterações no diagrama unifilar também refletem nas janelas de dados, embora a única ação possível seja a de criar uma linha de transmissão entre barramentos. Sempre que o usuário arrasta de um barramento para o outro, um segmento de reta (representando a linha de transmissão) é desenhado na interface e uma entrada é inserida na tabela de linhas de transmissão.

Figura 2 - Janela de interface gráfica do programa com o sistema-exemplo de três barras.



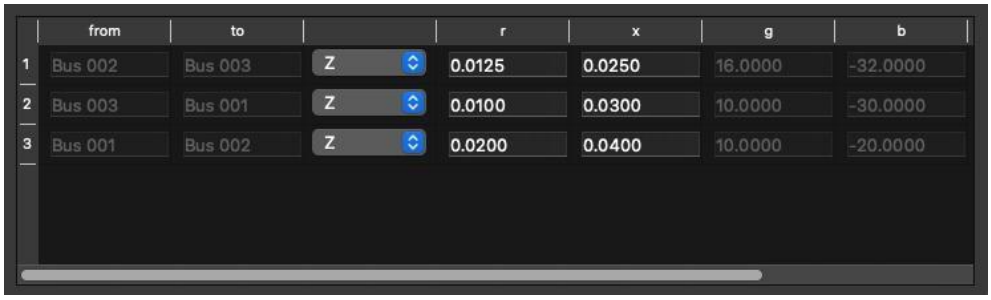
Fonte: Os autores.

Figura 3 - Janela de interface tabular com os dados de barras para o sistema-exemplo de três barras.



Fonte: Os autores.

Figura 4 - Janela de interface tabular com os dados de linhas para o sistema-exemplo de três barras.



Fonte: Os autores.

Em suma, para a modelagem de um sistema e execução do fluxo de potência no *software* PGD deve-se executar as etapas a seguir:

1) adicionar todos os barramentos (botão *Add Bus*);
2) criar as conexões entre os barramentos (botão vermelho no barramento do canvas);
3) modificar dados dos barramentos na tabela (botão *Buses*);
4) modificar dados das linhas de transmissão (botão *Lines*);
   executar o fluxo de potência (botão *Run Power Flow*).

2.2 Arquitetura

O programa foi desenvolvido pensando em facilitar a compreensão do fluxo de potência de uma rede elétrica. Por isso, é fundamental validar os dados de entrada e garantir que as alterações feitas pelo usuário sejam refletidas em tempo real. *Softwares* que reagem às variações do usuário são chamados de reativos. Essa reatividade se dá pelo sistema de observabilidade dos elementos do circuito, modelados a partir de elementos do tipo *Bus e Line* derivados de *NetworkElement*.

Para montar a estrutura do *software*, foi adotado o padrão de projeto *Model-View-Controller* (MVC), que decompõe toda estrutura do *software* em três camadas principais: *Model*, responsável pela lógica de dados; *Views*, encarregada da interface com o usuário; e *Controller*, na qual são definidas as regras de funcionamento do sistema, atuando como intermediário entre *Model* e *View*. Essa divisão de responsabilidades facilita a manutenção e a ampliação de funcionalidades, além de promover maior modularidade e reutilização de código.

2.3 Experiência do usuário

Todos os aspectos da estrutura do *software* foram definidos com o objetivo de garantir fluidez e consistência durante a sua utilização. Nesse sentido, o sistema foi projetado para refletir imediatamente na interface qualquer modificação realizada pelo usuário, reduzindo o número de interações necessárias para produzir um efeito desejado. Esse comportamento está alinhado com a heurística de usabilidade proposta por Nielsen e Molich (1990), que recomenda fornecer *feedback* imediato como forma de manter o usuário informado sobre o estado do sistema. Além disso, a consistência na interação e a previsibilidade das respostas fortalecem a sensação de controle e confiança, conforme apontado pelos autores em sua avaliação heurística de interfaces. O modelo adotado evita que o usuário se sinta inseguro quanto à realização de etapas ou comandos, promovendo uma experiência mais eficiente e confiável.

2.4 Modelagem dos Elementos do Sistema

A classe *NetworkElement* é uma abstração dos elementos básicos do sistema, sendo estendida pelas classes *Bus* e *Line*, que representam, respectivamente, os modelos de barramento e linha de transmissão. Essa classe base define propriedades comuns a todos os elementos, como *id* e *name*. As classes derivadas *Bus* e *Line* herdam essas características e adicionam propriedades específicas de cada tipo de componente. Com isso, aspectos estruturais do funcionamento básico ficam abstraídos na superclasse, o que facilita a modelagem individual de cada elemento do sistema.

2.5 Controlador

Ao serem criados, os componentes são armazenados em um controlador central denominado *SimulatorController*, responsável por garantir a unicidade dos elementos no sistema. Essa unicidade é assegurada por meio do uso de dicionários – estruturas de dados compostas por pares chave – valor, nas quais a chave é o identificador único (*id*) e o valor é o próprio objeto instanciado. Cada elemento é considerado único e imutável, sendo substituído por uma nova instância sempre que houver alguma alteração.

A escolha pela imutabilidade dos elementos visa garantir a segurança e a integridade dos dados. Com esse modelo, qualquer modificação realizada por meio da interface gráfica só pode ser aplicada por intermédio do *SimulatorController*. Dessa forma, todas as alterações ficam centralizadas em um único ponto do sistema, assegurando que tanto os modelos matemáticos quanto os elementos da interface gráfica sejam atualizados pelo mesmo agente e em resposta ao mesmo evento.

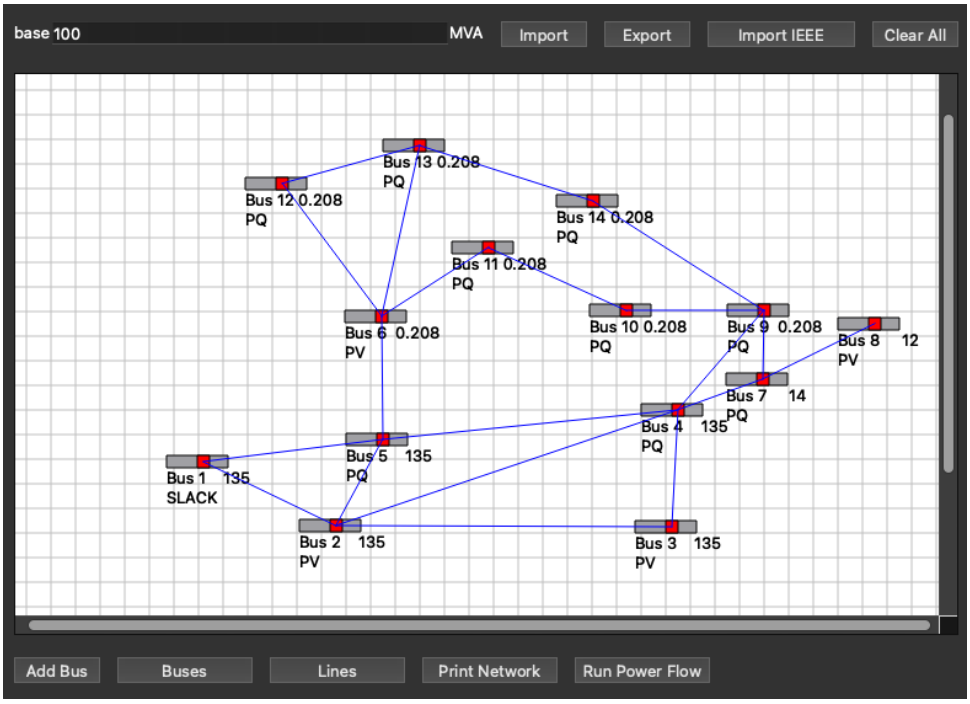
Após a substituição do elemento no dicionário, o controlador percorre a lista de observadores em busca daqueles interessados no elemento alterado. Em seguida, os observadores correspondentes se encarregam de atualizar os modelos internos e os componentes visuais vinculados ao elemento, independentemente da sua localização na interface.

3 ESTUDOS DE CASO E RESULTADOS

A validação do *software* PGD foi realizada mediante um estudo comparativo com o ANAREDE, utilizando o sistema IEEE de 14 barras (UNIVERSITY OF WASHINGTON, 1993). O sistema consiste de 5 geradores, 11 barras de carga, 20 linhas de transmissão, além de transformadores de tap ajustável em algumas seções de linha. A barra 1 é a barra de folga do sistema, enquanto as barras 2, 3, 6 e 8 são barras de geração. As demais barras do sistema são consideradas como barras de carga.

Na Figura 5 é mostrada a janela de interface gráfica do PGD para o sistema IEEE de 14 barras. A partir dela, é possível observar que as barras de carga (*PQ*), as barras de geração (*P*|*V*|) e a barra de folga (|*V*|*θ*) são mostradas na interface, enquanto as suas informações e as informações referentes as linhas de transmissão são ocultadas. Os dados de entrada do sistema podem ser inseridos nas janelas de interface tabulares de barras e de linhas do programa, como descrito anteriormente. O programa permite também a importação desses dados do arquivo padrão IEEE usando a função “Import IEEE”, desenvolvida no programa. Na inicialização do programa é assumido que as magnitudes das tensões são iguais a 1 pu e os ângulos das tensões são iguais a zero para as barras em que essas grandezas sejam as incógnitas.

Figura 5 - Janela de interface de barras para o sistema IEEE de 14 barras.



Fonte: Os autores.

Os resultados obtidos na estimação da magnitude das tensões (em pu) e dos ângulos (em radianos) foram analisados através do erro percentual absoluto médio relativo, descrito pela seguinte equação:

$$Erro = 200 \times \frac{|V_{est} - V_{ref}|}{|V_{est} + V_{ref}|} \ [\%] \qquad (1)$$

sendo

v_{est} o valor estimado usando *software* PGD; e
v_{ref} o valor de referência obtido com o ANAREDE.

Na Tabela 1 são apresentados os valores estimados e de referência para as magnitudes das tensões nas barras de carga e para os ângulos das tensões nas barras de carga e de geração, bem como os valores dos erros correspondentes na estimação de cada grandeza. Os valores v_{est} são extraídos da janela de interface tabular de barras, após a execução do fluxo de potência para o sistema IEEE de 14 barras.

Com base nos resultados obtidos na Tabela 1 é possível observar que a média dos erros de estimação ficam em torno de 0,13% e o seu desvio padrão é de aproximadamente 0,11%, indicando a robustez numérica do *software* PGD.

Tabela 1 - Comparação entre resultado e referência.

Incógnita	v_{ref}	v_{est}	*Erro [%]*
θ_2	-5,0000	-4,9826	0,3486
θ_3	-12,7000	-12,7251	0,1974
θ_4	-10,3000	-10,3129	0,1252
θ_5	-8,8000	-8,7739	0,2970
θ_6	-14,2000	-14,2209	0,1471
θ_7	-13,4000	-13,3596	0,3019
θ_8	-13,4000	-13,3596	0,3019
θ_9	-14,9000	-14,9385	0,2581
θ_{10}	-15,1000	-15,0973	0,0179
θ_{11}	-14,8000	-14,7906	0,0635
θ_{12}	-15,1000	-15,0756	0,1617
θ_{13}	-15,2000	-15,1563	0,2879
θ_{14}	-16,0000	-16,0336	0,2098
$\|V_4\|$	1,0180	1,0177	0,0295
$\|V_5\|$	1,0200	1,0195	0,0490
$\|V_7\|$	1,0620	1,0615	0,0471
$\|V_9\|$	1,0560	1,0559	0,0095
$\|V_{10}\|$	1,0510	1,0510	0,0000
$\|V_{11}\|$	1,0570	1,0569	0,0095
$\|V_{12}\|$	1,0550	1,0552	0,0190
$\|V_{13}\|$	1,0500	1,0504	0,0381
$\|V_{14}\|$	1,0360	1,0355	0,0483

Fonte: Os autores.

4 O *SOFTWARE* PGD COMO AMBIENTE DE APRENDIZAGEM BASEADA EM PROJETOS

A Aprendizagem Baseada em Projetos (ABP) é uma metodologia ativa centrada no estudante, promovendo a aprendizagem por meio da investigação de problemas reais. Ela estimula a autonomia, o pensamento crítico e a integração de saberes, resultando na criação de um produto final (BELL, 2010; LARMER & MERGENDOLLER, 2010; THOMAS, 2000; KRAJCIK & BLUMENFELD, 2006). Nesse contexto, o desenvolvimento do *software* PGD por estudantes de Engenharia Elétrica, cursando a disciplina de Sistemas de Energia do sétimo semestre, atende os seguintes requisitos da ABP, que são (KRAJCIK ; BLUMENFELD, 2006): Questões motivadoras; Problemas e situações que exigem investigação e estudo; Colaboração entre os participantes; Fornecimento de ferramentas para suporte da aprendizagem; e Geração de um produto.

O *software* aqui apresentado poderia ser posicionado como uma ferramenta de suporte a aprendizagem. Contudo, por ainda estar em estágio inicial de desenvolvimento, a sua principal contribuição reside no fato de que os estudantes são desafiados a resolver um problema real (no caso, o fluxo de potência), trabalhando de forma colaborativa para construir

soluções e um produto significativo. Para o desenvolvimento do *software*, é necessário que o aluno estude:

- modelagem de sistemas de potência (geradores, transformadores e linhas de transmissão);
- cálculo de redes de potência (construção de matrizes admitância e/ou impedância de barra);
- fluxo de potência (formulação básica do problema, tipos de barras e técnicas iterativas para solução do problema); e
- aplicações do fluxo de potência (estudos de planejamento do sistema elétrico, de estabilidade transitória de sistemas após contingências, entre outros).

Nesse caso, o professor atua como mediador do processo, promovendo a investigação, a reflexão e a aplicação prática dos conhecimentos. A avaliação ocorre de forma contínua e formativa, valorizando o processo de aprendizagem, a participação, o raciocínio crítico e a criatividade, além do resultado final. No entanto, cabe ressaltar que o desenvolvimento do *software* por si só não garante o sucesso do projeto da perspectiva educacional. É necessário que sejam experimentados e explorados diversos sistemas de teste e, somente como base na interpretação coerente e no entendimento dos resultados obtidos, o projeto pode ser bem sucedido.

O estudo do fluxo de potência ocorre de forma progressiva à medida que o aluno explora e desenvolve diferentes funcionalidades do *software*, em consonância com a concepção piagetiana de aprendizagem. Segundo Piaget (2010), esse processo não se dá por simples integração sucessiva de conceitos, mas por uma diferenciação progressiva, na qual os conceitos vão sendo construídos e refinados com base em um objeto de estudo concreto.

Finalmente, fica claro que a proposição de um desafio, em sala de aula, para a resolução de um problema real da indústria pode resultar em uma ferramenta computacional que tem como propósito não apenas o suporte ao aluno usuário, mas também, o incentivo e o aprendizado dos alunos que trabalham como desenvolvedores.

5 CONCLUSÃO

Neste trabalho, propõe-se o desenvolvimento de uma interface gráfica intuitiva e de fácil uso, voltada ao ensino de fluxo de potência no meio acadêmico. Por ser de código aberto, permite que estudantes e professores analisem, modifiquem e expandam suas funcionalidades, tanto para fins didáticos quanto para desenvolvimento. A interface facilita a compreensão de diagramas unifilares ao permitir a visualização clara de barramentos e linhas de transmissão, com conexões feitas diretamente pelo mouse, dispensando a inserção manual de dados em tabelas.

O tempo de desenvolvimento do *software* foi compatível com a carga horária de um curso generalista de Engenharia Elétrica. Os resultados indicam que a ferramenta oferece um ambiente de aprendizagem baseado em projetos que é funcional, colaborativo e com bom potencial didático, embora ainda precise ser testada com estudantes e docentes. Além disso, o projeto contribui significativamente para o aprendizado dos alunos desenvolvedores, evidenciado por avaliações contínuas em sala de aula.

REFERÊNCIAS

BELL, S. **Project-Based Learning for the 21st Century**. The Clearing House, v. 83, n. 2, p. 39–43, 2010.

ELETROBRÁS Cepel. **ANAREDE - Análise de Redes Elétricas**. Disponível em: <https://www.cepel.br/produtos/anared-2/> Acesso em: 19 mai. 2025.

KRAJCIK, J. S.; BLUMENFELD, P. C. Project-based learning. In: SAWYER, R. Keith (Ed.). **The Cambridge handbook of the learning sciences**. New York: Cambridge University Press, 2006. p. 317–333.

LARMER, J.; MERGENDOLLER, J. R. **Seven Essentials for Project-Based Learning**. Educational Leadership, v. 68, n. 1, p. 34–37, 2010.

MONTICELLI, J. A. **Fluxo de Carga em Redes de Energia Elétrica**. São Paulo: Blucher, 1983.

NIELSEN, Jakob; MOLICH, Rolf. Heuristic evaluation of user interfaces. In: SIGCHI Conference on Human Factors in Computing Systems, 1990, Seattle. ***Proceedings...*** New York: ACM, 1990. p. 249–256.

OLIVEIRA, M. M.; SANTOS, L. M. M. dos. **Aprendizagem baseada em projetos como estratégia ativa na formação de engenheiros**. Revista Educação, Santa Maria, v. 44, n. 3, p. 1-18, 2019.

PIAGET, Jean. **Psicologia e pedagogia**. 10. ed. Rio de Janeiro: Forense Universitária, 2010.

SAADAT, H. **Power System Analysis**. 1. ed. New York: McGraw-Hill, 1999.

THOMAS, J. W. **A Review of Research on Project-Based Learning**. San Rafael, CA: The Autodesk Foundation, 2000.

THURNER, L. et al. Pandapower—An Open-Source Python Tool for Convenient Modeling, Analysis, and Optimization of Electric Power Systems. **IEEE Transactions on Power Systems**, Piscataway, v. 33, n. 6, p. 6510 – 6521, Apr. 2018.

UNIVERSITY OF WASHINGTON. **IEEE 14-Bus System – Power Systems Test Case Archive**. Seattle, WA: Department of Electrical & Computer Engineering, 1993. Disponível em: https://labs.ece.uw.edu/pstca/pf14/pg_tca14bus.htm. Acesso em: 20 maio 2025.

ZIMMERMAN, R. D.; MURILLO-SANCHEZ, C. E.; THOMAS, R. J. MATPOWER: Steady-State Operations, Planning and Analysis Tools for Power Systems Research and Education. **IEEE Transactions on Power Systems**, Piscataway, v. 26, n. 1, p. 12 – 19, Feb. 2011.

DEVELOPMENT OF A GRAPHICAL INTERFACE TO SUPPORT POWER FLOW TEACHING – POWER GRID DESIGN

Abstract: *The modeling and analysis of power systems are recurring topics in undergraduate Electrical Engineering courses. Power flow, essential for steady-state analysis, allows determining voltages at buses and active and reactive power flows in lines. Mastering this knowledge is crucial for applications such as planning, expansion, and stability. Although simulators are available on the market, many have interfaces that are not very educational for beginners.To address this limitation, this paper proposes Power Grid Design, a free educational tool developed in Python and based on the Newton-Raphson method. With an intuitive graphical interface and tabular outputs, it facilitates data entry and result interpretation. The software development in the classroom promotes knowledge development through practical application, aligning with the project-based learning methodology.*

Keywords: *Power flow, Newton-Raphson method,* Project-based learning.