

Solução da Equação de Transporte de Nêutrons via Redes Neurais Informadas por Física: uma abordagem pedagógica

Solution of Neutron Transport Equation by Physics Informed Neural Network: a pedagogical approach

Ronni G.G. Amorim^{*1,2}, Eberth de A. Correa², Leandro X. Cardoso², Lina Chub²

¹Universidade de Brasília, Instituto de Física, Centro Internacional de Física, 70910-900, Brasília, DF, Brasil.

²Universidade de Brasília, Faculdade de Ciência e Tecnologia em Engenharia, 72444-240, Brasília, DF, Brasil.

Recebido em 08 de junho de 2026. Revisado em 31 de julho de 2026. Aceito em 02 de agosto de 2026.

Neste trabalho, trazemos um material pedagógico sobre a equação de transporte de nêutrons aplicada na análise de reatores nucleares. A apresentação é dividida em dois momentos. No primeiro, é realizada uma revisão sobre alguns conceitos de física nuclear, com a finalidade de deduzir a equação de transporte de nêutrons, e assim, na sequência resolver analiticamente tal equação para o reator nuclear crítico com formato cilíndrico. No segundo momento, uma revisão sobre redes neurais informadas por física é apresentada. Posteriormente a equação de transporte de nêutrons para o reator nuclear com formato cilíndrico em funcionamento crítico é resolvida com o uso das PINNs. Por fim, a solução analítica é comparada com a solução obtida por intermédio das PINNs, evidenciando que o método de solução via redes neurais é de fato promissor.

Palavras-chave: Inteligência artificial, aprendizado de máquina, redes neurais informadas por física, PINNs, equação de transporte de nêutrons.

In this work, we present pedagogical text on the neutron transport equation as applied to the analysis of nuclear reactors. The presentation is divided into two main parts. In the first part, we review fundamental concepts of nuclear physics with the aim of deriving the neutron transport equation. We then proceed to solve this equation analytically for critical nuclear reactors with cylindrical geometry. In the second part, we provide a review of Physics-Informed Neural Networks (PINNs). Subsequently, the neutron transport equation for the reactor configuration previously discussed is solved using the PINN approach. Finally, the analytical solution is compared with the solution obtained through the use of PINNs, demonstrating that the neural network-based method is indeed a promising approach.

Keywords: Artificial intelligence, machine learning, Physics-Informed Neural Networks, PINNs, neutrons transport equation.

1. Introdução

As tecnologias nucleares, apesar de serem bastante mitificadas, fazem parte do nosso cotidiano. Seja por meio de um exame de radiodiagnóstico, ou pelo consumo de alimentos conservados com uso de radiação, ou mesmo pela utilização de energia elétrica oriunda de uma usina nuclear, estamos imersos no universo da física nuclear. Dessa forma, compreendê-la nos ajuda a mitigar preconceitos existentes acerca da área nuclear e nos proteger contra o mau uso dessa fonte energética. Contudo, materiais instrucionais sobre energia nuclear escritos em língua portuguesa são escassos, sobretudo os destinados ao público leigo. Em particular, apesar de constar na literatura de língua inglesa bons materiais didáticos que discutem física de reatores nucleares, como por exemplos as referências [1–8], tais bibliografias não encontram paralelo em língua portuguesa,

comprometendo em parte, o aprendizado de alunos de graduação, bem como de interessados no assunto. Nessa perspectiva, este texto traz, num primeiro momento, uma revisão pedagógica acerca da equação de transporte de nêutrons, enfatizando os reatores nucleares críticos. Assim, soluções da equação de transporte nêutrons são apresentadas para o reator em formato cilíndrico. Tais soluções abarcam dois momentos: no primeiro momento são desenvolvidas analiticamente, e posteriormente são solucionadas mediante o uso de redes neurais informadas por física (PINNs).

O uso das PINNs (Physics-Informed Neural Networks) teve início de forma mais estruturada e reconhecida por volta de 2017, com destaque para os trabalhos pioneiros de Maziar Raissi, Paris Perdikaris e George Karniadakis. Esses pesquisadores propuseram formalmente a ideia de incorporar equações diferenciais (como leis físicas) diretamente no treinamento de redes neurais, de modo que a solução obtida respeitasse essas leis – mesmo em regiões sem dados. O trabalho de maior destaque nesse contexto, intitulado *Physics-informed neural networks*:

*Endereço de correspondência: ronniamorim@gmail.com

Editor-Chefe: Marcello Ferreira

A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, foi escrito em 2019 pelo trio consituído por Raissi, Perdikaris e Karniadakis [9]. Apesar de existirem estudos anteriores que combinavam aprendizado de máquina e modelagem física, foi esse trabalho que consolidou o termo PINNs e abriu caminho para uma ampla adoção da técnica em áreas como fluidodinâmica, física quântica, engenharia biomédica, dentre outras [10–12]. Desde então, o campo tem crescido rapidamente, com muitos avanços teóricos e práticos.

Nesse caminho, as Redes Neurais Informadas por Física representam um avanço significativo na interface entre aprendizado de máquina e modelagem científica. Sua importância reside na capacidade de incorporar diretamente leis físicas, expressas por equações diferenciais, dentro da estrutura de redes neurais, garantindo que as soluções obtidas respeitem os princípios fundamentais da física. Essa abordagem tem sido aplicada com sucesso em diversas áreas, sobretudo em física de reatores nucleares [13–15]. As PINNs são especialmente úteis para resolver equações diferenciais parciais e ordinárias em domínios complexos ou com dados escassos, onde métodos numéricos tradicionais enfrentam dificuldades. Ao conciliar dados empíricos e conhecimento físico, elas oferecem uma ferramenta robusta e flexível para simulações mais precisas e eficientes em problemas científicos e de engenharia. Nesse caminho, a nossa meta aqui foi incorporar, de uma forma simples e pedagógica, as PINNs na solução da equação de transporte de nêutrons para reatores nucleares críticos no formato cilíndrico; e no fim, comparar as soluções obtidas via PINNs com as soluções analíticas corriqueiras, também apresentadas neste trabalho. Com isso, almejamos que este trabalho auxilie o leitor, sobretudo o indivíduo lusófono, na compreensão de temas mais complexos tanto do arcabouço relativo à física de reatores nucleares, como no estudo de redes neurais informadas por física. Essa expectativa justifica-se em decorrência do trabalho apresentar uma sólida e cuidadosa revisão sobre a física de nêutrons, apresentando a equação de transporte aplicada no contexto de reatores nucleares, bem como por ter trazido uma discussão muito detalhada sobre PINNs, destinada principalmente a que está sendo introduzido no assunto.

Nessa perspectiva, a apresentação deste trabalho segue a sequência: na seção 2, uma revisão detalhada sobre a equação de transporte de nêutrons é destacada, trazendo ao leitor conceitos importantes como seção de choque e fluxo de nêutrons; na seção 3, aplicamos a equação de transporte de nêutrons na análise de reatores nucleares críticos de formato cilíndrico, obtendo, de forma detalhada, as soluções analíticas; na seção 4, apresentamos uma revisão sobre redes neurais informadas por física; na seção 5, trazemos as soluções para a equação de transporte obtidas com o uso de PINNs; por fim, na seção 6, as considerações finais e perspectivas são elencadas.

2. A Equação de Transporte de Nêutrons

Nesta seção, apresentaremos uma dedução da equação de transporte de nêutrons elencando algumas consequências físicas oriundas de sua interpretação.

A difusão de qualquer substância (neste texto, entenda-se por substância qualquer elemento que flua, seja um gás ou mesmo um grupo de partículas) é um problema já bem delineado pela física. Nesse sentido, o formalismo geral que será apresentado aqui seguirá de perto problemas análogos como a propagação do calor ou de um gás através de um dado meio. Nesse ideário, o leitor já deve ter vivenciado a situação de estar numa sala onde é aberto um recipiente de perfume. Nos instantes iniciais, após a abertura do frasco, apenas as pessoas que estão mais próximas sentirão o cheiro do perfume, mas após algum tempo, pessoas que estejam mais afastadas do recipiente poderão sentir o odor da substância. Isso ocorre porque o odor se difunde pela sala, e sua difusão obedece algumas leis e equações, como a lei de Fick e a equação de transporte, que veremos na sequência. O problema da difusão de nêutron é similar ao do perfume, ou seja, os nêutrons também se difundem por um dado meio, e o comportamento dessas partículas enquanto se difundem pode ser modelado por equações análogas à do perfume. Contudo, no caso dos nêutrons aparecem alguns ingredientes adicionais, pois a colisão de nêutrons com algumas partículas, como núcleos de urânio, podem levar ao surgimento de outros nêutrons, e essa particularidade deve ser incluída nas equações, o que será feito na dedução das equações próprias desse sistema.

2.1. O fluxo de nêutrons

A equação que descreve a difusão de uma dada substância representa a evolução temporal que descreve alguma característica da substância. Por exemplo, a equação de difusão do calor é representada pela evolução da quantidade de calor num dado ponto do espaço num dado instante. No caso da difusão dos nêutrons, a grandeza física que será utilizada como função a ser determinada é o fluxo de nêutrons, que será descrito a seguir.

O fluxo de nêutrons ϕ é definido por

$$\phi = n \cdot v, \quad (1)$$

em que n representa a densidade de nêutrons, isto é, o número de nêutrons por centímetro cúbico numa dada região, e v representa a velocidade média dos nêutrons (consideraremos neste trabalho apenas nêutrons monoenergéticos, isto é, com a mesma velocidade). Perceba que a unidade do fluxo de nêutrons é nêutrons por centímetro quadrado por segundo (nêutrons/cm²s). O fluxo de nêutrons, em geral, é uma grandeza que depende da posição espacial e do tempo.

Em física nuclear, é usual trabalhar com uma grandeza denominada seção de choque, denotada por σ , a qual

fornece a probabilidade de uma partícula colidir com a outra. A seção de choque é geralmente determinada experimentalmente, e depende de alguns parâmetros, como por exemplo da energia cinética (ou velocidade) da partícula incidente (é usual adotar um referencial em que uma das partículas esteja em movimento (a partícula incidente) e a outra esteja parada (partícula alvo). Assim, em materiais de física nuclear, há tabelas que contêm seções de choques para vários pares de partículas com diferentes velocidades. Na essência, a seção de choque (ou seção eficaz, em inglês *cross section*) representa a probabilidade de ocorrência de uma interação entre um nêutron e um núcleo atômico. Essa interação pode ser:

- Dispersão elástica ou inelástica (o nêutron colide e desvia, com ou sem perda de energia),
- Absorção (o nêutron é capturado),
- Fissão (em núcleos fissionáveis como o U-235).

A seção de choque é medida em unidades de área, sendo o barn (*b*) a unidade padrão:

$$1\text{barn} = 10^{-24}\text{cm}^2.$$

A seção de choque total é a soma de todas as seções de choque parciais:

$$\sigma_t = \sigma_s + \sigma_a,$$

em que σ_t representa a seção de choque total, σ_s representa a seção de choque de dispersão ou espalhamento (scattering), e σ_a representa a seção de choque de absorção (inclui fissão, captura, etc.). Vale a pena enfatizar que a seção de choque de fissão é uma subcategoria da seção de choque de absorção, ou seja, a seção de choque de fissão é a soma das seções de choque de fissão, captura radiativa, dentre outras.

A seguir, na Tabela 1, apresentamos alguns valores das seções de choque dos isótopos U-235 (urânio-235) e U-238 (urânio-238) para nêutrons térmicos (ou lentos), isto é, nêutrons com energia cinética de 0,025eV.

Na Tabela 1, como pode ser percebido, a seção de choque de absorção total é a soma seção de choque de fissão com a seção de choque de captura radiativa,

Tabela 1: Valores aproximados das seções de choque dos isótopos U-235 e U-238. Fonte: [8].

Isótopo	Tipo de Interação	σ (barns)
U-235	Fissão	$\approx 582,6$
	Captura Radiativa	$\approx 98,8$
	Absorção Total	$\approx 681,4$
	Espalhamento	≈ 14
	Total	$\approx 695,4$
U-238	Fissão	$\approx 0,002$
	Captura Radiativa	$\approx 2,68$
	Absorção Total	$\approx 2,682$
	Espalhamento	$\approx 8,871$
	Total	$\approx 11,55$

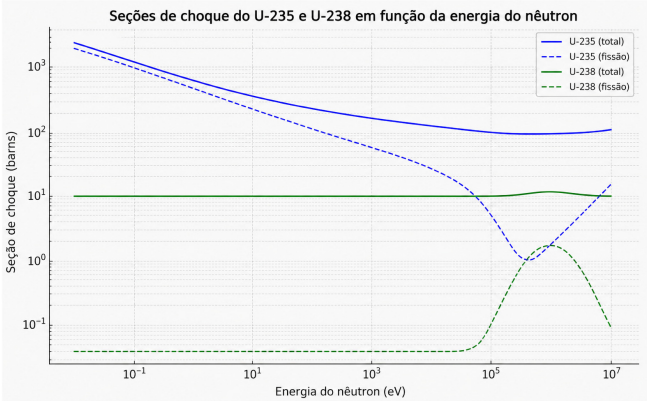


Figura 1: Dependência da seção de choque dos isótopos U-235 e U-238 em função da energia cinética dos nêutrons (Fonte: autoria própria).

enquanto a seção de choque total é igual a soma da seção de choque de absorção total com a seção de choque de espalhamento. O U-235 é fissionável com nêutrons térmicos, por isso tem uma seção de choque de fissão muito alta a baixas energias. Já o U-238 só sofre fissão com nêutrons rápidos, e sua seção de choque de fissão térmica é praticamente desprezível. Esses valores mudam bastante em energias mais altas (região epitérmica ou rápida).

A Figura 1 traz o gráfico das seções de choque dos isótopos U-235 e U-238, mostrando a dependência desse parâmetro com a temperatura ou com a energia cinética dos nêutrons.

No gráfico, as linhas contínuas mostram a seção de choque total, enquanto as linhas tracejadas mostram a seção de choque de fissão. Conforme podemos notar, o U-235 tem uma seção de fissão muito alta em nêutrons térmicos (à esquerda do gráfico), o que o torna ideal para reatores nucleares térmicos. Enquanto isso, o U-238 praticamente não sofre fissão com nêutrons térmicos, mas a seção de fissão aumenta com a energia, tornando-o relevante em reatores rápidos.

Se multiplicarmos a densidade atômica N pela seção de choque σ , obtemos um parâmetro denominado seção de choque macroscópica, denotada por Σ ,

$$\Sigma = N\sigma. \tag{2}$$

Assim como a seção de choque σ é definida para cada tipo de interação, a seção de choque macroscópica Σ também. Por exemplo, a seção de choque de absorção do U-235 e U-238 para nêutrons com energia cinética de 0,025eV são 681,4b e 2,682b, respectivamente. Como as densidades atômicas de U-235 e U-238 no urânio natural são $3,48 \cdot 10^{20}\text{átomos/cm}^3$ e $0,0483 \cdot 10^{24}\text{átomos/cm}^3$, respectivamente; então, a seção de choque macroscópica de absorção do U-235 e U-238 para os nêutrons térmicos são, respectivamente, $0,237\text{cm}^{-1}$ e $0,130\text{cm}^{-1}$. Assim, a seção de choque macroscópica de absorção do urânio natural é dada por $0,367\text{cm}^{-1}$.

Perceba que se conhecemos o fluxo de nêutrons, isto é, o número de nêutrons em cada centímetro quadrado de uma dada região em cada segundo, e também a seção de choque total das partículas-alvo disponíveis nesta região, podemos calcular a quantidade de colisões que ocorrerão por centímetro cúbico por segundo nesta região mediante o produto

$$F = \Sigma_t \phi,$$

em que F é denominado densidade de colisões. Esse é mais um parâmetro relevante na engenharia nuclear. De posse desses parâmetros, podemos discutir o próximo ponto: a lei de Fick.

2.2. A Lei de Fick

A teoria da difusão é baseada na lei de Fick. Tal lei foi originalmente utilizada na difusão química. É de conhecimento que quando, numa solução, a concentração de determinado soluto numa região é maior do que em outra, o soluto se difunde da região de maior concentração para a de menor concentração. Além disso, sabe-se também que a taxa com que o soluto flui é proporcional ao negativo do gradiente da concentração do soluto. Os nêutrons, em certas condições, possuem o comportamento similar ao do soluto numa solução. Isto é, se a densidade ou o fluxo de nêutrons for maior numa região de um reator do que em outra, existirá uma corrente de nêutrons fluindo da região de maior densidade para a de menor densidade. Assim, se considerarmos que o fluxo varia ao longo da direção x , a lei de Fick fica dada por [6, 8]

$$J_x = -D \frac{d\phi}{dx}, \quad (3)$$

em que D é o coeficiente de difusão, o qual depende de parâmetros relacionados ao sistema analisado; no caso considerado, D depende dos parâmetros associados ao meio em que os nêutrons estão fluindo. J_x é denominada densidade de corrente de nêutrons, sendo uma medida do número de nêutrons que passa, por unidade de tempo, através de uma unidade de área perpendicular à direção x . A unidade de J_x é a mesma do fluxo, isto é, nêutrons/cm²s.

Se o fluxo é função das três coordenadas espaciais, definimos

$$\vec{J} = -D \nabla \phi, \quad (4)$$

em que $\vec{J}$ é denominado vetor densidade de corrente de nêutrons. O significado físico de $\vec{J}$ pode ser obtido se considerarmos o produto escalar

$$\vec{J} \cdot \hat{n} = J_n,$$

onde $\hat{n}$ é o vetor normal unitário numa direção arbitrária. J_n obtido é igual à quantidade de nêutrons por segundo

que atravessam uma unidade de área normal à direção definida por $\hat{n}$.

A fim de compreendermos melhor esses conceitos, consideremos que uma fonte de nêutrons pontual que emita Q nêutrons por segundo possua o fluxo de nêutrons dado por

$$\phi(r) = \frac{Qe^{-r/L}}{4\pi Dr},$$

em que r é a distância radial da fonte até algum ponto, D é o coeficiente de difusão, e L é uma constante. A partir deste fluxo, podemos calcular a corrente. Notemos que ϕ depende apenas de r . Assim, se tomarmos o operador gradiente em coordenadas esféricas, considerando apenas a componente radial, temos que $\nabla = \hat{r} \frac{d}{dr}$, em que r é o vetor unitário na direção radial. Assim, a densidade de corrente, que também só terá componente na direção radial, é dada por

$$\vec{J} = -D \nabla \phi,$$

que no caso em tela é dado por

$$\vec{J} = -D \hat{r} \frac{d}{dr} \left(\frac{Qe^{-r/L}}{4\pi Dr} \right),$$

a partir de que obtemos

$$\vec{J} = \hat{r} \frac{Q}{4\pi} \left(\frac{1}{r^2} + \frac{1}{rL} \right) e^{-r/L}.$$

A partir de J , podemos calcular a quantidade de nêutrons por segundo que atravessam uma unidade de área normal à direção definida por $\hat{r}$. Para isso, notemos que no caso considerado $\vec{J}$ é sempre perpendicular à superfície de uma esfera. Assim, $\vec{J} \cdot \hat{r}$ nos fornece a quantidade de nêutrons que atravessam cada unidade de área da esfera. E assim, a quantidade de nêutrons que, a cada segundo, atravessam a esfera inteira é dada por $4\pi r^2 \vec{J} \cdot \hat{r}$, o que nos fornece $Q \left(1 + \frac{r}{L} \right) e^{-r/L}$.

Para finalizar esta subseção, vamos discutir o significado do coeficiente D . Lembrando que $\vec{J}$ e ϕ possuem a mesma unidade e visualizando a Eq. (3), percebemos que D tem que ter dimensão de comprimento. Pode-se mostrar (o que é muito longo para ser feito aqui neste texto introdutório), que o coeficiente de difusão pode ser escrito como [8]

$$D = \frac{1}{3\Sigma_S(1 - \bar{\mu})}, \quad (5)$$

em que $\bar{\mu}$ é o valor médio do cosseno do ângulo que os nêutrons são espalhados após a colisão com as partículas-alvo. Para os núcleos mais usados nos reatores, $\bar{\mu}$ pode ser calculado por meio de

$$\bar{\mu} = \frac{2}{3A},$$

onde A é o número de massa atômica da partícula-alvo. Se definirmos o livre caminho médio de transporte λ_{tr}

como

$$\lambda_{tr} = \frac{1}{\Sigma_S(1 - \bar{\mu})},$$

podemos reescrever D na forma simples

$$D = \frac{1}{3\lambda_{tr}}.$$

2.3. A equação de continuidade

Outro ingrediente fundamental para a obtenção da equação de transporte de nêutrons é a equação de continuidade. Essa equação aparece nas mais variadas áreas da física, de forma que o leitor já possa ter se deparado com ela em outros contextos. A fim de obtê-la para o caso dos nêutrons, consideremos um volume arbitrário V no qual há nêutrons e outras partículas. Com o passar do tempo, a quantidade de nêutrons dentro de V pode mudar por algum dos seguintes processos: se nêutrons escaparem para fora de V , se nêutrons forem absorvidos por alguma partícula presente em V , ou se fontes emissoras de nêutrons estão presentes em V . A equação de continuidade, a qual será obtida nesta subseção, é nada mais que a linguagem matemática para dizer que, ao levarmos em conta os três processos descritos, a quantidade de nêutrons deve se conservar. Resumidamente, podemos escrever que a taxa de variação do número de nêutrons em V é igual à taxa com que os nêutrons são produzidos em V , subtraído da taxa que os nêutrons são absorvidos em V , subtraído também da taxa com que os nêutrons escapam de V . Temos que contabilizar cada termo sugerido neste resumo.

Com a finalidade de contabilizarmos a variação do número de nêutrons com o tempo, consideremos n como a densidade de nêutrons em qualquer ponto em V . Assim, o total de nêutrons em V é dado por

$$\int_V n(\vec{r}, t) dV,$$

onde a integração é realizada em todo o volume V . A taxa de variação do número de nêutrons é dada por

$$\frac{d}{dt} \int_V n dV = \int_V \frac{\partial n}{\partial t} dV.$$

Para computarmos a taxa de produção, consideremos que os nêutrons são emitidos por uma fonte localizada em V que produza s nêutrons por centímetro cúbico a cada segundo. Assim, a taxa de produção é dada por

$$\int_V s dV.$$

Com essa integral obtemos o número de nêutrons produzidos no volume V a cada segundo.

A taxa de absorção por centímetro cúbico por segundo é dada por $\Sigma_a \phi$, em que Σ_a é a seção de choque macroscópica da partícula-alvo (essa partícula-alvo, que

no reator nuclear pode ser o núcleo de U-235, poderá absorver o nêutron após a colisão). Assim, a taxa de absorção é dada por

$$\int_V \Sigma_a \phi dV.$$

Após o cálculo dessa integral, obtemos o número de nêutrons absorvidos dentro de V a cada segundo.

A taxa de escape de nêutrons é encontrada por meio da densidade de corrente de nêutrons. Para isso, considere $\vec{J}$ como o vetor densidade de corrente na superfície de V e $\hat{n}$ é o vetor normal unitário direcionado para fora da superfície. Assim, sabemos que $\vec{J} \cdot \hat{n}$ nos fornece o número de nêutrons que atravessam a superfície A que delimita V - de dentro para fora - por cm^2/s . Com isso, a taxa de vazamento é dada por

$$\int_A \vec{J} \cdot \hat{n} dA.$$

Como A é uma superfície fechada, podemos usar o teorema da divergência e reescrever a taxa de vazamento como

$$\int_V \nabla \cdot \vec{J} dV.$$

Nesse caminho, a taxa com que a quantidade de nêutrons dentro de V varia com o tempo é dada por

$$\int_V \frac{\partial n}{\partial t} dV = \int_V s dV - \int_V \Sigma_a \phi dV - \int_V \nabla \cdot \vec{J} dV.$$

Como todas as integrais estão relacionadas ao mesmo volume, podemos escrever

$$\frac{\partial n}{\partial t} = s - \Sigma_a \phi - \nabla \cdot \vec{J}, \quad (6)$$

que é a forma geral da equação de continuidade. Nos casos em que a densidade de nêutrons não é função do tempo, obtemos

$$\nabla \cdot \vec{J} + \Sigma_a \phi - s = 0, \quad (7)$$

que é equação de continuidade do estado estacionário.

2.4. A equação de transporte

Se substituirmos a Eq. (4) na Eq. (6), obtemos

$$D \nabla^2 \phi - \Sigma_a \phi + s = \frac{\partial n}{\partial t}, \quad (8)$$

que é a equação de transporte de nêutrons.

Para o caso estacionário, obtemos

$$D \nabla^2 \phi - \Sigma_a \phi = -s, \quad (9)$$

a qual pode ser escrita na forma

$$\nabla^2 \phi - \frac{1}{L^2} \phi = -\frac{s}{D}, \quad (10)$$

em que $L = \sqrt{D/\Sigma_a}$ é denominado comprimento de difusão, termo que aparece com frequência na engenharia nuclear.

A Eq. (10) é a equação de difusão de nêutrons estacionária mais genérica, ainda não adaptada para a análise de um retator nuclear. Em breve faremos essa adaptação. Antes disso, vamos aplicá-la a uma situação prática.

2.4.1. Uma placa infinita irradiando nêutrons

Consideremos uma fonte de nêutrons que seja um plano infinito que emite Q_0 nêutrons por cm^2/s . Se considerarmos que o plano é o $y-z$, localizado em $x=0$, podemos admitir que o fluxo é uma função apenas da coordenada x , que é a distância de onde o fluxo é medido até o plano. Assim, podemos considerar que não há fontes em nenhum outro ponto além de $x=0$, ou seja, utilizaremos a fonte no plano como uma condição de contorno. Dessa forma, a Eq. (10) pode ser escrita para este caso como

$$\nabla^2 \phi - \frac{1}{L^2} \phi = 0, \quad x \neq 0. \quad (11)$$

A solução geral da Eq. (11) é dada por

$$\phi(x) = Ae^{-x/L} + Be^{x/L},$$

em que A e B são constantes a serem determinadas com o uso das condições de contorno. Por ora, consideraremos apenas $x > 0$. Assim, para que o fluxo seja finito, o coeficiente B deve ser nulo. Dessa forma, o fluxo de nêutrons fica dado por

$$\phi(x) = Ae^{-x/L}.$$

O coeficiente A é determinado com o uso do fato de que na origem, em $x=0$, há uma fonte que emite Q_0 nêutrons por centímetro quadrado por segundo. Como o plano emite nos dois sentidos, admitamos, por simetria, que no sentido $x > 0$ a emissão seja de $Q_0/2$ nêutrons por centímetro quadrado por segundo. Dessa forma, a condição de contorno nos diz que

$$\lim_{x \rightarrow 0} J(x) = \frac{Q_0}{2}.$$

A partir da lei de Fick, podemos escrever

$$J(x) = -D \frac{d\phi}{dx} = \frac{DA}{L} e^{-x/L}.$$

Logo, usando a condição de contorno, temos que

$$A = \frac{Q_0 L}{2D}.$$

E assim, o fluxo para $x > 0$ é dado por

$$\phi(x) = \frac{Q_0 L}{2D} e^{-x/L}.$$

Se quisermos uma solução que valha também para $x < 0$, por simetria, basta que escrevamos

$$\phi(x) = \frac{Q_0 L}{2D} e^{-|x|/L}. \quad (12)$$

Os resultados deste exemplo serão utilizados num caso mais complexo que trataremos a seguir.

2.4.2. Uma placa infinita irradiando nêutrons numa região fechada

No próximo exemplo, consideraremos o plano infinito do exemplo anterior, irradiando nêutrons a uma taxa Q_0 nêutrons por centímetro quadrado por segundo, envolto pelos dois lados por paredes infinitas. Consideremos que o plano que irradia esteja localizado na origem do sistema de coordenadas, em $x=0$, e a uma distância d de ambas as paredes. Ou seja, a parede da esquerda está localizada em $x=-d$, enquanto a da direita está em $x=d$. A equação que devemos resolver é a mesma do exemplo anterior, Eq. (11) pois consideraremos a placa irradiando como uma condição de contorno, isto é,

$$\lim_{x \rightarrow 0} J(x) = \frac{Q_0}{2}.$$

A solução é dada por $\phi(x) = Ae^{-x/L} + Be^{x/L}$, e aqui não podemos assumir nenhuma das duas constantes como nula, já que o sistema não inclui o infinito, pois é limitado pelas paredes.

Agora, além dessa condição, temos também que o fluxo de nêutrons nas paredes deve ser nulo, ou seja,

$$\phi(d) = \phi(-d) = 0.$$

Assim, temos então

$$\phi(d) = Ae^{-d/L} + Be^{d/L} = 0,$$

e

$$\phi(-d) = Ae^{d/L} + Be^{-d/L} = 0,$$

Com isso, podemos escrever $B = -Ae^{-2d/L}$, e a solução será

$$\phi(x) = A[e^{-x/L} - e^{(x-2d)/L}].$$

A constante A pode ser determinada, como no exemplo anterior, com a condição da fonte Q_0 . Usando a lei de Fick obtemos:

$$J(x) = \frac{DA}{L} [e^{-x/L} + e^{(x-2d)/L}].$$

E assim,

$$A = \frac{Q_0 L}{2D} [1 + e^{-2d/L}]^{-1}.$$

Logo, para $x > 0$, obtemos

$$\phi(x) = \frac{Q_0 L}{2D} \left[\frac{e^{-x/L} - e^{(x-2d)/L}}{1 + e^{-2d/L}} \right].$$

Por simetria, a solução válida para qualquer valor de x é dada por

$$\phi(x) = \frac{Q_0 L}{2D} \left[\frac{e^{-|x|/L} - e^{(|x|-2d)/L}}{1 + e^{-2d/L}} \right]. \quad (13)$$

Multiplicando o numerador e o denominador da Eq. (13) por $e^{d/L}$, chegamos a

$$\phi(x) = \frac{Q_0 L}{2D} \left[\frac{e^{-(|x|-d)/L} - e^{(|x|-d)/L}}{e^{d/L} + e^{-d/L}} \right]. \quad (14)$$

Agora, usando as funções trigonométricas hiperbólicas, isto é $2 \cosh w = e^w + e^{-w}$ e $2 \sinh w = e^w - e^{-w}$, a Eq. (14) passa a ser escrita na forma

$$\phi(x) = \frac{Q_0 L}{2D} \left[\frac{\sinh[(d - |x|)/L]}{\cosh(d/L)} \right]. \quad (15)$$

Neste ponto, é importante comentarmos que a condição de contorno $\phi(d) = \phi(-d) = 0$ não é uma condição física, pois o fluxo de nêutrons não é realmente nulo nas paredes; caso fossem, não haveria corrente de nêutrons através das paredes, e nenhum nêutron seria encontrado além delas. Assim, na verdade $\phi \approx 0$ nas paredes. Sob um ponto de vista mais formal, a diferença reside na condição de fronteira que adotamos, ou seja, utilizamos a condição de Dirichlet, na qual o nó se encontra rigorosamente na fronteira. A abordagem fica fisicamente mais aceitável caso utilizássemos a condição de Marshak [16], na qual o nó encontra-se fora do domínio físico, isto é, numa fronteira extrapolada. De forma simplificada, na condição de Dirichlet, o fluxo cai abruptamente para zero na parede, enquanto na de Marshak, o fluxo atinge zero um pouco depois da parede. Em síntese, na condição de Marshak, a extrapolação da fronteira pode ser escrita como $\phi(d \pm \delta) = 0$, em que $\delta \approx 2D$. O valor mais preciso da teoria de transporte é $\delta = 0,7104 \cdot (3D) \approx 2,13D$. Esse resultado advém da condição

$$\phi + 2D \frac{d\phi}{dx} = 0,$$

o que implica também $J = \phi/2$. Optamos por não deduzir aqui os detalhes do último resultado mostrado. Na prática, continuaremos usando a condição de Dirichlet, mas compreendendo que, fisicamente, há fluxo de nêutron na parede, o que possibilita a corrente de nêutrons através dela, e assim, o vazamento de nêutrons nos casos reais. Assim, com o intuito de simplificar os cálculos, continuaremos a utilizar a condição de Dirichlet.

Com tal explicação concluída, podemos retomar a discussão prática sobre os planos infinitos. A Eq. (15) nos fornece bastante informação acerca do sistema considerado. Por exemplo, suponha que queiramos determinar o número de nêutrons que escapa do sistema por segundo de cada cm^2 através das duas paredes. Para isso, é suficiente calcularmos a corrente de nêutrons na parede,

ou seja, em $x = d$ e em $x = -d$ e adicionarmos os dois resultados.

Fazendo isso, a partir da Eq. (15) podemos deduzir que

$$J(x) = -\frac{Q_0 \cosh[(d - |x|)/L]}{2 \cosh(d/L)}.$$

Logo,

$$J(d) = -\frac{Q_0}{2 \cosh(d/L)}.$$

Da mesma forma,

$$J(-d) = \frac{Q_0}{2 \cosh(d/L)}.$$

E assim, a quantidade de nêutrons que escapa do sistema por segundo de cada cm^2 através das duas paredes é igual a $J(d) + J(-d)$, ou seja,

$$J(d) - J(-d) = \frac{Q}{\cosh(d/L)}.$$

A partir deste resultado, podemos determinar a probabilidade com que os nêutrons emitidos possam vazar pelas paredes. Se a fonte emite Q nêutrons por centímetro quadrado por segundo e se escapam $\frac{Q_0}{\cosh(d/L)}$, essa probabilidade é dada por

$$\frac{J(d) + J(-d)}{Q} = \frac{1}{\cosh(d/L)}.$$

Esses cálculos, apesar de serem relativamente simples, são muito importantes na elaboração de projetos de blindagem, quando se quer reduzir a quantidade de nêutrons que escapam de determinado sistema.

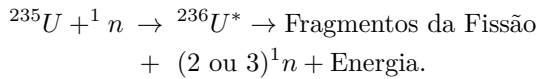
2.5. A equação de transporte de nêutrons no contexto do reator nuclear

Nesta subseção, deduziremos a equação de transporte de nêutrons apropriada para se analisar o fluxo de nêutrons num reator nuclear. Consideraremos o modelo mais simples que simula um reator nuclear: um reator rápido e crítico contendo uma mistura de combustível e refrigerante, consistindo de apenas uma região. Este reator é denominado reator nú e a dinâmica dos nêutrons em seu interior podem ser descritos pela Eq. (10), isto é,

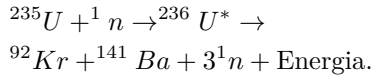
$$D\nabla^2 \phi - \Sigma_a \phi = -s,$$

No reator crítico, o que estamos considerando aqui, são os nêutrons emitidos na fissão. Aqui vale recordar que quando um nêutron atinge um núcleo de urânio-235, ele pode ser absorvido, formando um núcleo instável de urânio-236, que se divide em dois fragmentos menores (núcleos fi-lhos) e libera mais nêutrons, além de uma grande quantidade de energia.

A reação mais geral pode ser escrita como



Um exemplo específico de fissão é



A energia liberada é em torno de 200MeV por fissão. Dessa forma, a fonte de nêutrons s pode ser dada por

$$s = \eta \Sigma_{aF} \phi,$$

em que η é o número médio de nêutrons emitidos por fissão, e Σ_{aF} é a seção macroscópica de choque total do combustível.

Notemos que o termo de fonte pode ser reescrito como

$$s = \eta f \Sigma_a \phi,$$

em que a razão $f = \frac{\Sigma_{aF}}{\Sigma_a}$ é denominado fator de utilização e Σ_a é a seção macroscópica de choque da mistura combustível e refrigerante. Assim, f é igual a fração de nêutrons absorvidos no reator e que de fato são absorvidos no combustível. Ou seja, f fornece a fração de nêutrons que de fato podem provocar a fissão no urânio-235.

O produto ηf é denominado fator de multiplicação de um meio infinito, representado por $k_\infty = \eta f$. Um meio infinito é definido como um meio em que não há perda de nêutrons por fuga.

De forma geral, o fator de multiplicação, geralmente representado por

$$k = \frac{\text{número de nêutrons na geração atual}}{\text{número de nêutrons na geração anterior}}.$$

k é um parâmetro fundamental na física de reatores nucleares. Ele indica a razão entre o número de nêutrons na geração atual e o número de nêutrons na geração anterior. Esse parâmetro pode ser classificado em dois tipos, o fator de multiplicação efetivo, k_{eff} , que leva em conta todas as perdas de nêutrons no reator, como nêutrons que escapam do reator e nêutrons absorvidos sem causar fissão; e o fator de multiplicação infinito (já mencionado anteriormente), k_∞ .

O valor de k pode ser interpretado com os seguintes critérios:

- $k < 1$ – Subcrítico: A reação de fissão está diminuindo; o reator "morre" com o tempo.
- $k = 1$ – Crítico: A reação está estável. Cada geração de nêutrons mantém a próxima. É o estado desejado para operação contínua de um reator.
- $k > 1$ – Supercrítico: A reação está se intensificando. O número de nêutrons (e a potência) cresce com o tempo.

Com isso, o termo de fonte pode finalmente ser escrito como

$$s = k_\infty \Sigma_a \phi.$$

Substituindo na Eq. (10), obtemos

$$\nabla^2 \phi + B^2 \phi = 0, \quad (16)$$

em que

$$B^2 = \frac{k_\infty - 1}{L^2}. \quad (17)$$

Nas equações (16) e (17) temos o termo B que é referido em inglês pela palavra *Buckling* medida em cm^{-2} . Essas equações serão aplicadas na próxima seção para o estudo do reator nuclear cilíndrico.

3. Solução Analítica da Equação de Transporte de Nêutrons aplicadas a reatores nucleares

Nesta seção, o nosso objetivo é aplicar a Eq. (16) para analisar reatores nucleares. Consideraremos nesta abordagem analítica o reator no formato cilíndrico, que é o caso mais comum em usinas nucleares, sobretudo nas usinas PWR [5]. Esperamos que esta seção forneça ao leitor subsídios que o auxilie na compreensão da física de reatores nucleares, sobretudo dos conceitos envolvidos no projeto de reatores.

Nesse caminho, vamos considerar o reator cilíndrico de raio a e altura H . Devido à simetria do problema, optamos por utilizar coordenadas cilíndricas (r, θ, z) . Em nossa análise, consideraremos o caso em que o fluxo de nêutrons não dependa da coordenada θ . Assim, o laplaciano apenas com a dependência em r e z é dado por

$$\nabla^2 = \frac{1}{r} \frac{\partial}{\partial r} \left(r \frac{\partial}{\partial r} \right) + \frac{\partial^2}{\partial z^2}.$$

Com isso, a Eq. (16) fica escrita na forma

$$\frac{1}{r} \frac{\partial}{\partial r} \left(r \frac{\partial \phi}{\partial r} \right) + \frac{\partial^2 \phi}{\partial z^2} + B^2 \phi = 0. \quad (18)$$

Assim, como foi feito na seção precedente, vamos supor que a solução seja dada por $\phi(r, z) = R(r)Z(z)$. Assim, obtemos as seguintes equações diferenciais ordinárias

$$\frac{1}{r} \frac{d}{dr} \left(r \frac{dR}{dr} \right) + B_r^2 R = 0, \quad (19)$$

$$\frac{d^2 Z}{dz^2} = -B_z^2 Z, \quad (20)$$

onde B_r^2 e B_z^2 são as constantes de separação, e satisfazem

$$B^2 = B_r^2 + B_z^2. \quad (21)$$

A solução da Eq. (20) é dada por

$$Z(z) = A_1 \cos(B_z z) + C_1 \sin(B_z z).$$

Utilizando a condição de contorno que o fluxo deve ser nulo nas tampas do reator (bases do cilindro), $z = H/2$ e $z = -H/2$, encontramos que a solução deve ser uma função par que satisfaça $Z(H/2) = 0$.

Isso nos fornece para o reator crítico

$$\cos\left(B_z \frac{H}{2}\right) = 0,$$

ou seja,

$$B_z = \frac{\pi}{H}. \quad (22)$$

E assim, a solução da parte em z fica dada por

$$Z(z) = A_1 \cos\left(\frac{\pi}{H} z\right). \quad (23)$$

Agora retomamos à solução da parte radial. Após manipulação, a Eq. (19) pode ser reescrita na forma

$$r^2 \frac{d^2 R}{dr^2} + r \frac{dR}{dr} + B_r^2 r^2 R = 0 \quad (24)$$

Essa equação é um caso particular da equação diferencial de Bessel, que é dada dada por [17]

$$r^2 \frac{d^2 R}{dr^2} + r \frac{dR}{dr} + (B_r^2 r^2 - p^2) R = 0. \quad (25)$$

Ou seja, a Eq. (24) é a equação diferencial de Bessel, Eq. (25), com $p = 0$. Sua solução é bem conhecida na literatura, pois é dada em termos de [17]

$$R(r) = A_2 J_0(B_r r) + C_2 Y_0(B_r r), \quad (26)$$

em que A e C são constantes, enquanto $J_0(B_r r)$ e $Y_0(B_r r)$ são, respectivamente, a função de Bessel e a função de Neumann de ordem zero. Todas as identidades e propriedades sobre as funções de Bessel que utilizamos podem ser encontradas na referência. Enquanto a função de Bessel $J_0(B_r r)$ é bem comportada para qualquer valor da variável, a função de Neumann $N_0(B_r r)$ diverge na origem, ou seja, $\lim_{r \rightarrow 0^+} Y_0(B_r r) = \infty$. Como o nosso problema envolve a origem, ou seja, o núcleo do reator cilíndrico, localizado em $r = 0$, faz parte do sistema analisado, o coeficiente C_2 da Eq. (26) deve ser igual a zero. Por isso, a solução da parte radial fica dada por

$$R(r) = A_2 J_0(B_r r). \quad (27)$$

Usando a condição de contorno que o fluxo deve ser nulo na lateral do cilindro, em $r = a$, temos

$$J_0(B_r a) = 0.$$

Como estamos interessados no reator crítico, apenas a primeira raiz da função de Bessel de ordem zero é relevante. Essa raiz é igual a 2,405 [17]. Assim,

$$B_r = \frac{2,405}{a} \quad (28)$$

E a solução da parte radial toma a forma

$$R(r) = A_2 J_0\left(\frac{2,405r}{a}\right). \quad (29)$$

Com isso, a solução do problema fica a princípio dada por

$$\phi(r, z) = A J_0\left(\frac{2,405r}{a}\right) \cos\left(\frac{\pi z}{H}\right), \quad (30)$$

em que $A = A_1 A_2$ é uma constante que determinaremos a seguir.

Assim, como comentamos no caso dos planos infinitos, a condição de Dirichlet, $\phi = 0$ na fronteira da região considerada (tampas e laterais do cilindro), é apenas uma condição matemática. Na prática, o fluxo de nêutrons não é nulo na fronteira, pois se assim fosse, nêutrons não vazariam do reator. Na verdade, temos que $\phi \approx 0$, o que advém da condição de Marshak [16], isto é,

$$\phi(R, z) + 2D \frac{\partial \phi(R, z)}{\partial r} = 0,$$

nas laterais do cilindro ($r = R$), e

$$\phi(r, \pm H/2) + 2D \frac{\partial \phi(r, \pm H/2)}{\partial z} = 0,$$

nas tampas do cilindro ($z = \pm H/2$). Na prática, trocamos $R \rightarrow R + \delta$ e $H = H + 2\delta$, em que $\delta = 2D$, nas equações oriundas da solução do reator, inclusive na Eq. (17). Como este trabalho tem cunho meramente pedagógico, continuaremos a admitir a condição de Dirichlet (entendendo-a apenas como formalidade matemática), mas sabendo que na realidade o fluxo de nêutrons não se anula nas fronteiras da região analisada.

A potência, grandeza física que se conhece (a priori) ao se projetar um reator, pode ser calculada por meio da integral [8]

$$P = E_R \Sigma_f \int_V \phi(r, z) dV,$$

o que no caso considerado é igual a

$$P = E_R \Sigma_f \int_0^a \int_{-H/2}^{H/2} A J_0\left(\frac{2,405r}{a}\right) \times \cos\left(\frac{\pi z}{H}\right) 2\pi r dz dr, \quad (31)$$

Lembremos que o jacobiano em coordenadas cilíndricas é igual a $r dr d\theta dz$.

A integral em z é facilmente calculada e resulta em $\frac{2H}{\pi}$. Com isso, ficamos com

$$P = 4AHE_r\Sigma_f \int_0^a J_0\left(\frac{2,405r}{a}\right) r dr. \quad (32)$$

Para calcular a integral em r devemos lembrar que

$$J_0(B_r r) = \sum_{n=0}^{\infty} \frac{(-1)^n}{(n!)^2} \left(\frac{B_r r}{2}\right)^{2n}.$$

Com isso, a integral em r é dada por

$$\int_0^a \sum_{n=0}^{\infty} \frac{(-1)^n}{(n!)^2} \left(\frac{B_r r}{2}\right)^{2n} r dr.$$

Ou ainda

$$\sum_{n=0}^{\infty} \frac{(-1)^n}{(n!)^2} \left(\frac{B_r}{2}\right)^{2n} \int_0^a r^{2n+1} dr.$$

O que nos fornece

$$\sum_{n=0}^{\infty} \frac{(-1)^n}{(n!)^2} \left(\frac{B_r}{2}\right)^{2n} \frac{a^{2n+2}}{2n+2}.$$

Podemos escrever a expressão anterior na seguinte forma

$$2\frac{a}{B_r} \sum_{n=0}^{\infty} \frac{(-1)^n}{2n!(n+1)!} \left(\frac{B_r a}{2}\right)^{2n+1}.$$

Essa última expressão pode ser escrita em termos da função de Bessel de ordem 1, $J_1(B_r a)$, ou seja

$$\frac{a}{B_r} J_1(B_r a).$$

Lembrando que $B_r = 2,405/a$, a Eq. (32) fica dada por

$$P = 1,6632a^2 AHE_r\Sigma_f J_1(2,405).$$

Assim,

$$A = \frac{P}{1,6632a^2 H E_r \Sigma_f J_1(2,405)}.$$

E assim, a solução final do problema é

$$\phi(r, z) = \frac{P}{1,6632a^2 H E_r \Sigma_f J_1(2,405)} \times J_0\left(\frac{2,405r}{a}\right) \cos\left(\frac{\pi z}{H}\right). \quad (33)$$

Agora, vamos calcular a quantidade de nêutrons que escapam do reator a cada segundo. Para isso, precisamos calcular a densidade de corrente. O operador gradiente nem coordenadas cilíndricas é dado por $\nabla = \hat{r} \frac{\partial}{\partial r} + \hat{\theta} \frac{1}{r} \frac{\partial}{\partial \theta} + \hat{z} \frac{\partial}{\partial z}$. Como para o caso considerado o fluxo de

nêutrons não depende de θ , temos que os componentes não-nulos da densidade de corrente de nêutrons são

$$J_r = -D \frac{P}{1,6632a^2 H E_r \Sigma_f J_1(2,405)} \times \left(\frac{\partial}{\partial r} J_0\left(\frac{2,405r}{a}\right)\right) \cos\left(\frac{\pi z}{H}\right). \quad (34)$$

e

$$J_z = -D \frac{P}{1,6632a^2 H E_r \Sigma_f J_1(2,405)} \times J_0\left(\frac{2,405r}{a}\right) \left(\frac{\partial}{\partial z} \cos\left(\frac{\pi z}{H}\right)\right). \quad (35)$$

Usando a relação $\frac{dJ_0(x)}{dx} = -J_1(x)$, temos

$$J_r = \frac{2,405DP}{1,6632a^3 H E_r \Sigma_f J_1(2,405)} \times J_1\left(\frac{2,405r}{a}\right) \cos\left(\frac{\pi z}{H}\right). \quad (36)$$

e ainda,

$$J_z = \frac{\pi DP}{1,6632a^2 H^2 E_r \Sigma_f J_1(2,405)} \times J_0\left(\frac{2,405r}{a}\right) \sin\left(\frac{\pi z}{H}\right). \quad (37)$$

Calculando o fluxo através das paredes do reator, $\int_A \vec{J} \cdot \hat{n} dA$, obtemos

$$\int_0^{2\pi} \int_{-H/2}^{H/2} J_r(r=a) a d\theta dz,$$

e

$$2 \int_0^{2\pi} \int_0^a J_z(z=H/2) r d\theta dr.$$

No cálculo do fluxo através da base e tampa do reator foi usada a relação $\int x J_0(x) dx = x J_1(x)$. Com isso, a quantidade de nêutrons que escapa por segundo através das paredes do reator é dada por

$$\frac{DP}{2a^2 H^2 E_r \Sigma_f} (11,57H^2 + \pi^2 a^2) \text{ nêutrons/s.}$$

Os resultados estabelecidos nesta seção serão comparados aos obtidos a partir das PINNs.

4. O Que é Uma Rede Neural Informada Por Física (PINN)?

Recentemente tivemos grandes avanços com as redes neurais para se fazer previsões baseadas e dados de pré-treinamento. Aplicações em *deep learning* desde reconhecimento de imagens, identificação de padrões em grandes volumes de dados, análise de tendências com aplicações no mercado financeiro e, principalmente, o

desenvolvimento de *Large Language Models* (LLMs) para facilitar a interface máquina-humano. Nos últimos anos, tem sido proposto o treinamento de redes neurais para a resolução de equações diferenciais parciais (EDP), através das redes neurais informadas pela física (PINNs em inglês), tanto para a resolução na forma direta quanto na forma indireta. A resolução de uma EDP na forma direta refere-se ao fato de, dadas as condições de contorno, as soluções são previstas através da equação diferencial proposta com os coeficientes em suas formas funcionais conhecidos. O problema inverso refere-se a descobrir os coeficientes a partir de dados conhecidos.

Em ambos os casos, podemos destacar as vantagens em usarmos PINNs ao invés de outros métodos com um custo computacional maior como o método de elementos finitos (MEF):

1. Dispensam discretização espacial:
 - É necessária uma malha computacional para o MEF, o que pode ser complexo em geometrias irregulares ou em altas dimensões.
 - As PINNs aprendem a solução diretamente a partir das coordenadas espaciais (ou temporais), sem necessidade de discretização.
2. São mais indicadas para dados incompletos ou ruidosos:
 - Os dados experimentais podem ser incorporados diretamente pelas PINNs, mesmo que esparsos ou ruidosos, ajustando a solução de forma híbrida (física + dados).
 - O MEF depende fortemente de condições de contorno bem definidas.
3. Tratamento Natural de Problemas Inversos:
 - O MEF resolve, na grande maioria dos casos, problemas diretos, enquanto as PINNs podem inferir parâmetros desconhecidos da equação (problemas inversos) de forma mais direta.
4. Comprovadamente, tem menor custo computacional em problemas de alta dimensionalidade:
 - O MEF sofre com a “maldição da dimensionalidade” (ex.: problemas 3D+tempo exigem malhas extremamente finas).
 - As PINNs aproximam a solução com uma rede neural, que pode ser mais eficiente em domínios complexos.
5. Suavidade da solução:
 - As PINNs fornecem soluções infinitamente diferenciáveis (útil para calcular derivadas da solução).
 - O MEF geralmente usa funções de bases contínuas por partes, limitando a suavidade.
6. Facilidade em problemas multifísicos e acoplados:
 - As PINNs podem incorporar múltiplas leis físicas simultaneamente na função de perda.
 - No MEF, modelos multifísicos exigem acoplamento complexo entre diferentes formulações.

Contudo, devemos destacar algumas desvantagens das PINNs. O treinamento pode ser demorado, pois

redes neurais profundas exigem ajuste fino e podem ter convergência lenta. Uma outra desvantagem que merece destaque é a dificuldade em capturar soluções com descontinuidades, pois o MEF lida melhor com choques ou descontinuidades (ex.: escoamentos compressíveis). Porém, seu uso vem se ampliando e os modelos vêm sendo cada vez mais validados, constituindo uma nova área de desenvolvimento para possíveis aplicações Físicas e industriais.

4.1. Redes Neurais informadas pela Física (PINNs)

O termo “informadas pela Física” advém do fato de definirmos a taxa de perda (*loss rate*) a partir da EDP em questão. Em nosso caso, queremos resolver uma equação diferencial dada na forma:

$$\begin{cases} \nabla^2 u(\mathbf{r}) + k^2 u(\mathbf{r}) = 0, \\ u(\mathbf{r}) = 0 \text{ if } \mathbf{r} \in \partial\Omega, \end{cases} \quad (38)$$

que é uma equação diferencial parcial do tipo Helmholtz, no qual $B^2 = (k_\infty - 1)/L^2$ como já foi explicado anteriormente.

Agora, queremos aproximar a solução com uma rede neural. Para tanto, considere a rede neural diagramada pela Figura 2. A questão aqui é que queremos aproximar a solução tal que:

$$u_\psi(\mathbf{r}) \approx u(\mathbf{r}), \quad (39)$$

onde $u_\psi : \mathbb{R}^3 \rightarrow \mathbb{R}$ denota uma função com parâmetros ψ a ser treinada pela rede neural. Esta deve ser organizada em $C + 1$ camadas de neurônios. A primeira camada $\mathcal{N}^{(0)}$ é a chamada camada de entrada que deve conter as variáveis, ou seja,

$$\mathcal{N}^{(0)}(\mathbf{r}) = \mathbf{r} = (x, y, z). \quad (40)$$

Cada camada c subsequente será então parametrizada por um peso $\mathbf{P}^{(c)} \in \mathbb{R}^{s_{c-1} \times s_c}$ e por um *bias* $\mathbf{b}^{(c)} \in \mathbb{R}^{s_c}$, onde s_c é definido como tamanho da saída da camada c . As camadas $c \in [1, C - 1]$ são chamadas de camadas escondidas densas cuja saída de cada uma é definida recursivamente como sendo:

$$\mathcal{N}^{(c)}(\mathbf{r}) = \xi \left(\mathbf{P}^{(c)} \mathcal{N}^{(c-1)}(\mathbf{r}) + \mathbf{b}^{(c)} \right), \quad (41)$$

onde ξ é chamada de função de ativação não-linear que especificaremos mais adiante. Ao final do processo, na camada de saída que terá apenas um valor será então dada por:

$$\mathcal{N}^{(C)}(\mathbf{r}) = \mathbf{P}^{(C)} \mathcal{N}^{(C-1)}(\mathbf{r}) + \mathbf{b}^{(C)}, \quad (42)$$

onde, finalmente, $u_\psi(\mathbf{r}) = \mathcal{N}^{(C)}(\mathbf{r})$.

Diagramaticamente, na Figura 2, temos a camada de entrada $\mathcal{N}^{(0)}$ representada por 3 neurônios para abrigar os valores de x, y e z . Seguem então mais 3 camadas

Estratégia de treinamento de uma vanilla-PINN com condições de contorno de Dirichlet

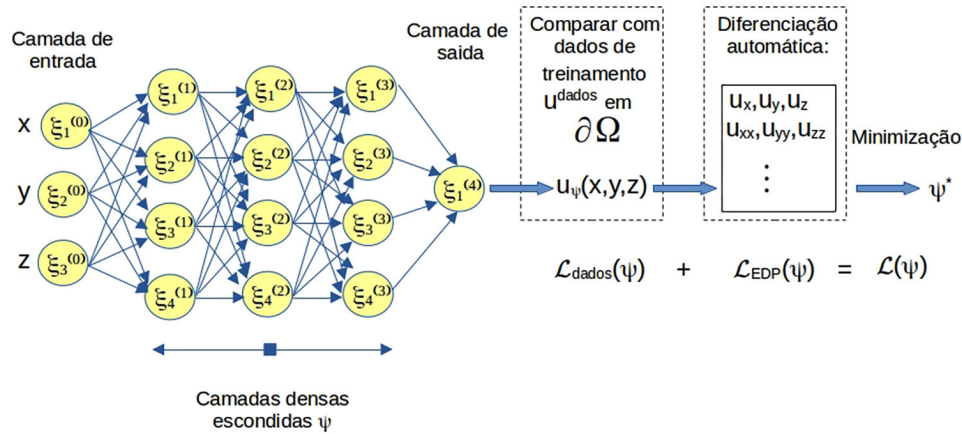


Figura 2: Rede neural com 4 neurônios por camada escondida densa.

densas escondidas de 4 neurônios cada, finalizando na camada de saída $\mathcal{N}^{(C)}(\mathbf{r})$. Neste caso, temos que $C = 4$ e a chave é definirmos quem são as respectivas taxas de perda (*loss rate*), o que faremos logo abaixo. O número de parâmetros a serem treinados seguem a fórmula $p_e * E_1 + E_1 * E_2 + E_2 * E_3 + E_3 * p_s + E_1 + E_2 + E_3 + p_s$. No caso da Figura 2 temos que $p_e = 3$ (número de parâmetros de entrada), $E_1 = E_2 = E_3 = 4$ (número de neurônios por camada densa escondida) e $p_s = 1$ (número de parâmetros de saída), totalizando 61 parâmetros a serem treinados pela rede neural. Este é apenas um exemplo esquemático mais simples para explicarmos o que faremos a seguir.

Na Figura 3 temos o dimensionamento real de parâmetros que usaremos em nossa PINN que serão $p_e = 3$, $E_1 = E_2 = E_3 = E_4 = E_5 = E_6 = E_7 = 64$ e $p_s = 1$, totalizando 25.281 parâmetros de treinamento da rede neural. A fim de capturarmos as particularidades da PDE em todos os pontos avaliados no cilindro nós utilizaremos 7 camadas densas de neurônios escondidos. Para tanto, é necessário utilizarmos uma topologia de rede neural um pouco mais sofisticada para evitarmos um explosão de gradientes. Isso é alcançado criando-se “atalhos” através de *add layers* entre as camadas densas escondidas. Isso é feito automaticamente pelo *tensorflow* através da *flag* “PINN_ResNet” ao construirmos o modelo, como veremos a seguir no programa. O *tensorflow* mostra inicialmente a camada de entrada se conectando à primeira camada densa escondida como na Figura 2 e, em seguida, a conexão com as demais. Dessa forma, o *tensorflow* adicionará os *add layers* a cada duas camadas densas escondidas, criando os mencionados “atalhos” para os gradientes de aprendizado ao longo da rede. Vale ressaltar que esses *add layers* não contribuem com novos parâmetros de treinamento e são apenas vias de fluxo para os gradientes gerados entre cada duas camadas. Essa é uma característica fundamental distinta dessas

redes, o que não ocorre com as redes sequenciais usuais como a da Figura 2.

A técnica que empregaremos é a aproximação por aprendizado supervisionado usando dados de treinamento que, em nosso caso, serão os valores conhecidos da função em $\partial\Omega$ e no ponto central. Seja então, $\mathcal{D}$ o conjunto com N_{dados} desses dados, $(\mathbf{r}, u)$, para treinamento:

$$\mathcal{D} = \{(\mathbf{r}_i^{\text{dados}}, u_i^{\text{dados}})\}_{i=1}^{N_{\text{dados}}} \quad (43)$$

Assim, $u_\psi(\mathbf{r})$ será considerada uma boa aproximação se as previsões de u estiverem próximas de u_i^{dados} para cada $i \in [1, N_{\text{dados}}]$. Ou seja, queremos o conjunto de parâmetros otimizados tais que:

$$\psi^* = \min_{\psi} \mathcal{L}_{\text{dados}}(\psi) \quad (44)$$

com

$$\begin{aligned} \mathcal{L}_{\text{dados}}(\psi) &= \frac{1}{N_{\text{dados}}} \sum_{i=1}^{N_{\text{dados}}} |u_\psi(\mathbf{r}_i) - u_i^{\text{dados}}|^2 \\ &= \mathcal{L}_{\text{centro}} + \mathcal{L}_{\text{faces}} \end{aligned} \quad (45)$$

que é o erro quadrático médio que define a taxa de perda com relação aos dados de treinamento.

Ao resolvermos a Eq. (44) estaremos otimizando os parâmetros ψ com relação aos dados de treinamento que, em nosso caso, serão os pontos $(\mathbf{r}_i^{\text{dados}}, u_i^{\text{dados}})$ em $\partial\Omega$ e no ponto central, como mencionamos logo acima. Na verdade, para resolvermos esta equação de otimização utilizamos o algoritmo de gradiente descendente ou *backpropagation*. Esse algoritmo depende de uma diferenciação automática que é realizada sem problemas pelo *Tensorflow* sobre $\mathcal{L}_{\text{dados}}(\psi)$. Esta tarefa é realizada dentro de uma tolerância desejada através de:

$$\psi^{j+1} = \psi^j - \alpha \nabla_{\psi} \mathcal{L}(\psi^j) \quad (46)$$

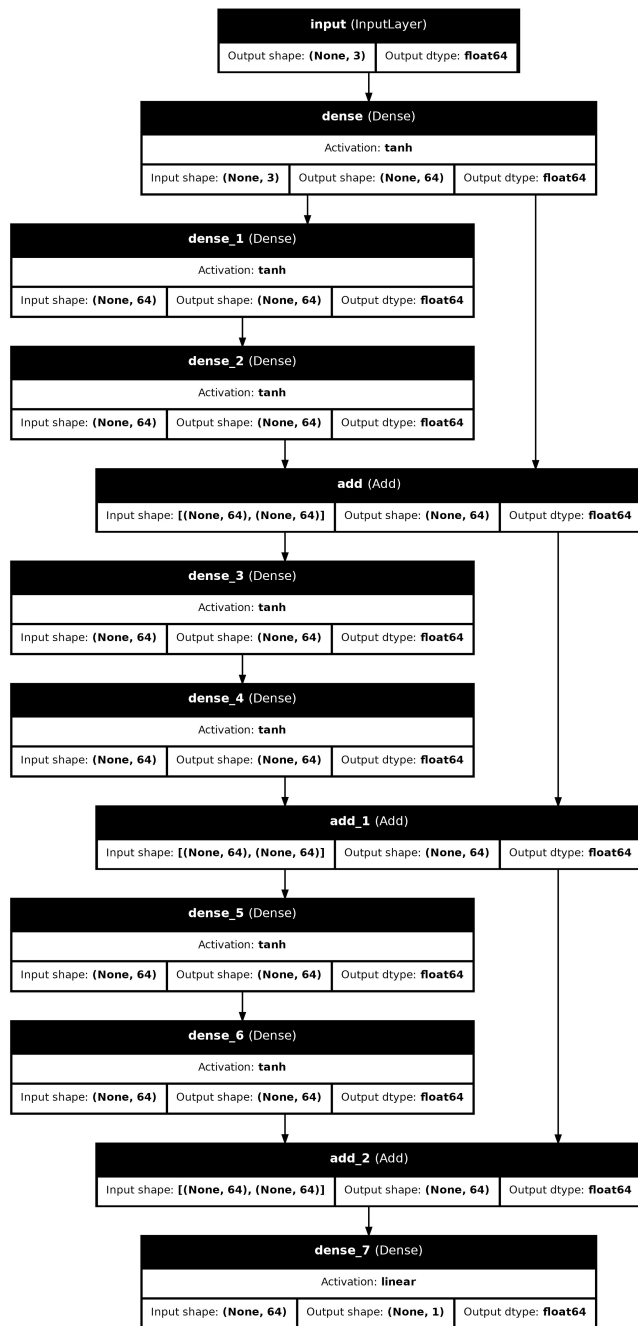


Figura 3: Arquitetura utilizada para treinar a PINN do problema cilíndrico.

com $\mathcal{L} = \mathcal{L}_{\text{dados}}$ para a j -ésima iteração que chamamos de época. α é a taxa de aprendizado e deve ser otimizado para algumas rodadas iniciais. Se for muito pequeno demorará muito a convergir e se for grande não otimizará corretamente os parâmetros. Então, rodamos para um número pequeno de épocas e ajustamos o seu valor que geralmente fica em torno de 10^{-3} a 10^{-2} , inicialmente. Vamos utilizar como otimizador o chamado *adaptive moment estimation* (ADAM) que ajusta a taxa de aprendizado dos parâmetros durante o treino, a partir de um valor inicial que estipulamos acima do valor

final desejado. Essa técnica de resolução é chamada de Vanilla-PINN[18].

5. Soluções da Equação de Transporte de Nêutrons obtidas a partir das PINNs

Nesta seção, apresentamos a solução do reator cilíndrico via PINN; e para esse fim será usado o conteúdo discutido nas duas seções precedentes. Assim, queremos resolver a Eq. (18) por meio do treinamento de uma PINN que satisfaça certas condições de contorno para um cilindro de altura H e raio a . Tal como fizemos anteriormente, definiremos um sistema de coordenadas e um fluxo de nêutrons normalizados. Ao final dos cálculos teremos constantes multiplicativas que fazem o sistema retornar às dimensões físicas reais, facilitando o treinamento das PINNs. Dessa forma, faremos as seguintes normalizações;

$$\begin{aligned} r &\rightarrow \sqrt{x^2 + y^2} \rightarrow \frac{r}{a} \\ z &\rightarrow \frac{z}{H} \\ u(r, z) &= \frac{\phi(r, z)}{\zeta} \end{aligned} \quad (47)$$

com

$$\zeta = \frac{P}{1,632a^2 H E_R \Sigma_f J_1(2,405)}. \quad (48)$$

Dessa forma, não é difícil ver que a Eq. (18) se torna uma equação similar à equação de Helmholtz Eq. (38)

$$\frac{1}{r} \frac{\partial}{\partial r} \left(r \frac{\partial u}{\partial r} \right) + r_{aH}^2 \frac{\partial^2 u}{\partial z^2} + k^2 u = 0, \quad (49)$$

com $k = aB$ e $r_{aH} = a/H$. Temos agora uma equação diferencial normalizada e parametrizada pela razão entre o raio a e a altura H e o parâmetro $k^2 = (aB)^2 = k_r^2 + k_z^2$, com $k_r \approx 2,405$ e $k_z = \pi r_{aH}$.

No entanto, esta equação, tal como está, não é conveniente para o treino de uma PINN devido à singularidade em $r = 0$. Isso traz instabilidades ao modelo que não permitem o treinamento da PINN de forma correta. Para contornarmos isso, vamos reescrever o operador diferencial desta equação em coordenadas cartesianas para obtermos

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + r_{aH}^2 \frac{\partial^2 u}{\partial z^2} + k^2 u = 0. \quad (50)$$

Assim, a Eq. (33), que representa a solução analítica será dada por:

$$u(r, z) = u(x, y, z) = J_0(k_r r) \cos(\pi z) \quad (51)$$

com $r = \sqrt{x^2 + y^2}$ e as condições de contorno dadas pela Eq. (38).

No listing 1, podemos ver as bibliotecas que serão necessárias para resolvermos a Eq. (38) utilizando as

técnicas de aprendizado de máquina. A primeira biblioteca é a *tensorflow* que contém todas as ferramentas necessárias para o treinamento de modelos de redes neurais que serão utilizadas. A segunda é a biblioteca padrão para manipulações matemáticas e técnicas de vetorização chamada *numpy* que se integra perfeitamente com a biblioteca do *tensorflow*. A biblioteca *scipy.special* fornece as funções de Bessel utilizadas no cálculo. As demais bibliotecas *matplotlib.pyplot* e

mpl_toolkits.mplot3d são bibliotecas de visualização dos resultados já bem estabelecidas do *Python*. Note que o parâmetro k^2 , definido acima, entra como *ksq* e r_{aH} como *r_aH* e, além disso, definimos a precisão do modelo no *tensorflow* em “float64”. Temos também os pesos para a taxa de perda com relação à EDP em questão, um ponto central onde será colocada a fonte e às condições de contorno, respectivamente com *wpde*, *wcenter* e *wbc*.

Listing 1: Bibliotecas a serem chamadas.

```
1 import tensorflow as tf
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from mpl_toolkits.mplot3d import Axes3D
5 from scipy.special import jv, j0, yv
6 # Parametros fisicos
7 r_aH = 0.5 # razao a/H do raio pela altura do cilindro
8 k_r = 2.404825557695773 # localiza o zero da funcao de Bessel j0
9 ksq = (np.pi * r_aH)**2 + k_r**2 # k^2 = (aB)^2 = (aBz)^2 + (aBr)^2
10 tf.keras.backend.set_floatx("float64")
11 wpde, wcenter, wbc = 1., 10., 10.
12 # Solucao exata
13 def exact_solution(r, theta, z):
14     """Solucao exata que satisfaz u=0 em r=1. e z=+/-0.5"""
15     # Constantes para satisfazer condicoes de contorno
16     k_r = 2.404825557695773 # Primeiro zero de J0
17     k_z = np.pi # Para satisfazer u=0 em z=+/-0.5
18     return j0(k_r * r) * np.cos(k_z * z)
```

No listing 2 mostramos a função que gera os pontos do *mesh* para treinamento. Adotamos as coordenadas cilíndricas (r, θ, z) e também mostramos a função que

converte as coordenadas cilíndricas para cartesianas, pois utilizaremos com frequência essa transformação. Na Figura 4 mostramos os pontos do *mesh* gerados.

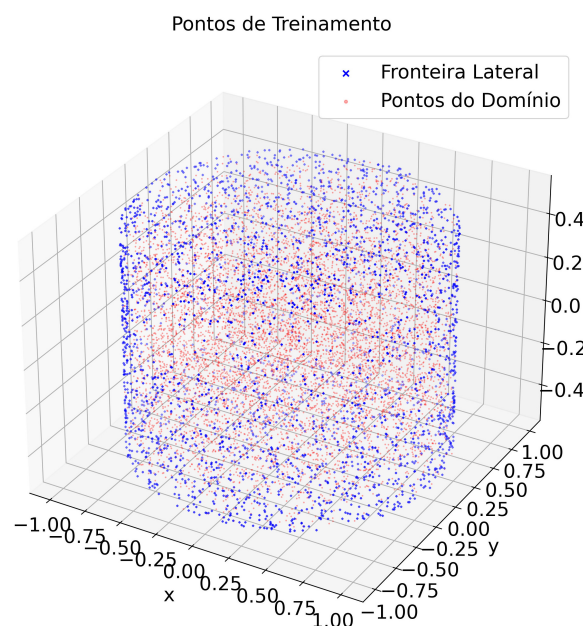


Figura 4: *mesh* para o treinamento da PINN.

Por padrão, a não ser que se queira aumentar o número de pontos do *mesh*, teremos: $n_{\text{boundary}}=3000$, $n_{\text{domain}}=8000$, o que significam 3000 pontos nas faces e 8000 pontos na região interna do cilindro. A lista *boundary_values* contém os valores da função u nas

fronteiras, ou seja, são os valores da condição de contorno sobre u em todos os *boundary_points*. *domain_points* é uma lista com todos os pontos no interior do reator cilíndrico normalizado com $r \in [0, 1]$ e $z \in [-0.5, 0.5]$.

Listing 2: Gerador do *mesh* para treinamento.

```

1 def cartesian(r, theta, z):
2     return r * np.cos(theta), r * np.sin(theta), z
3 def generate_data(n_boundary=3000, n_domain=8000):
4     n_lateral = n_boundary // 2
5     r_lat = np.full((n_lateral, 1), 1.0)
6     theta_lat = np.random.uniform(-np.pi, np.pi, (n_lateral, 1))
7     z_lat = np.random.uniform(-0.5, 0.5, (n_lateral, 1))
8     lateral_points = np.hstack([r_lat, theta_lat, z_lat])
9
10    n_faces = n_boundary - n_lateral
11    n_face_each = n_faces // 2
12
13    def generate_disk_cylindrical(n_points, z_val):
14        r = np.sqrt(np.random.uniform(1e-5, 1.0, (n_points, 1)))
15        theta = np.random.uniform(-np.pi, np.pi, (n_points, 1))
16        z = np.full((n_points, 1), z_val)
17        return np.hstack([r, theta, z])
18
19    face1_points = generate_disk_cylindrical(n_face_each, -0.5)
20    face2_points = generate_disk_cylindrical(n_face_each, 0.5)
21
22    boundary_points = np.vstack([lateral_points, face1_points, face2_points])
23    boundary_values = np.zeros((boundary_points.shape[0], 1))
24
25    # Dominio amostragem proporcional ao volume
26    r_vol = np.sqrt(np.random.uniform(1e-5, 1.0, (n_domain, 1)))
27    theta_vol = np.random.uniform(-np.pi, np.pi, (n_domain, 1))
28    z_vol = np.random.uniform(-0.5, 0.5, (n_domain, 1))
29    domain_points = np.hstack([r_vol, theta_vol, z_vol])
30
31    # Visualizacao dos pontos nas faces e no dominio do cilindro
32    font_size = 18
33    fig = plt.figure(figsize=(12, 10))
34    ax = fig.add_subplot(111, projection='3d')
35
36    # Converter para coordenadas cartesianas para visualizacao
37    x_bod, y_bod, z_bod = cartesian(boundary_points[:,0], boundary_points[:,1],
38                                   boundary_points[:,2])
39    x_dom, y_dom, z_dom = cartesian(domain_points[:,0], domain_points[:,1],
40                                   domain_points[:,2])
41
42    # Plot dos pontos de amostragem
43    ax.scatter(x_bod, y_bod, z_bod, s=2, marker='x', c='blue',
44              label='Fronteira Lateral')
45    ax.scatter(x_dom, y_dom, z_dom, s=0.5, marker='.', c='red', alpha=0.3,
46              label='Pontos do Dominio')
47
48    # Configuracao dos eixos e titulo com fonte 18
49    ax.set_xlabel('x', fontsize=font_size, labelpad=12)
50    ax.set_ylabel('y', fontsize=font_size, labelpad=12)
51    ax.set_zlabel('z', fontsize=font_size, labelpad=12)
52    ax.set_title('Pontos de Treinamento', fontsize=font_size, pad=15)
53    ax.tick_params(axis='both', which='major', labelsize=font_size)
54    leg = ax.legend(fontsize=font_size, loc='upper right')
55    for handle in leg.legend_handles:
56        handle._sizes = [30]
57    plt.savefig("mesh_cilindrico.png", dpi=300, bbox_inches='tight')
58    plt.show()
59    return boundary_points, boundary_values, domain_points

```

No listing 3 definimos a arquitetura de nossa rede neural e cada camada densa escondida é nomeada. Como vimos na Figura 3, utilizaremos 7 camadas com 64 neurônios cada. Essa parte é determinada por tentativa e erro. Devido ao ponto central, essa arquitetura se mostrou mais eficiente na convergência. Outras com menos *hidden layers* podem ser testadas aumentando-se o número de neurônios. Além disso, utilizaremos o ativador como sendo a função não linear “tanh”, com o *kernel* inicializador *glorot_normal*, o que garante uma convergência mais rápida e estável, pois evita que os pesos iniciais impliquem uma explosão nos gradientes. Esse ativador é essencial para os cálculos uma vez que possui segunda derivada não nula, o que não acontece com ativadores lineares como “relu”. Isso se deve ao fato de estarmos lidando com operadores diferenciais de segunda ordem. Esta arquitetura está representada na Figura 3. Note que o *InputLayer* tem dimensão 3, embora estajamos interessados apenas nas coordenadas r e z . Na verdade, a solução da Eq. (18) utilizou da simetria com relação à variável angular θ . Dessa forma, teremos sempre o conjunto de variáveis (r, θ, z) que serão importantes na hora de convertermos para coordenadas cartesianas (x, y, z) quando necessário, como veremos adiante. Essa é uma boa estratégia que ficará mais clara nos próximos *listings*. Por fim, note a *flag* “PINN_ResNet” na definição do modelo ao fim desta rotina. Como comentamos anteriormente, ela será responsável pela inclusão de “atalhos” para os gradientes, evitando que haja uma explosão destes.

Listing 3: Arquitetura da rede neural.

```

1 def build_model():
2     inputs = tf.keras.layers.Input(shape=(3,), dtype=tf.float64, name='input')
3     x = tf.keras.layers.Dense(64, activation='tanh',
4                               kernel_initializer='glorot_normal')(inputs)
5     # Blocos Residuais
6     for i in range(3):
7         shortcut = x
8         h = tf.keras.layers.Dense(64, activation='tanh',
9                                   kernel_initializer='glorot_normal')(x)
10        h = tf.keras.layers.Dense(64, activation='tanh',
11                                  kernel_initializer='glorot_normal')(h)
12        x = tf.keras.layers.add([shortcut, h])
13    outputs = tf.keras.layers.Dense(1, activation=None,
14                                    kernel_initializer='glorot_normal')(x)
15    return tf.keras.Model(inputs=inputs, outputs=outputs, name="PINN_ResNet")

```

No listing 4 definimos a a EDP a ser resolvida e a solução exata para comparação com o modelo. A implementação é feita através do decorador @, utilizando a função *tensorflow.GradientTape*. O modo “persistent” é utilizado para que numa mesma operação de memória sejam calculadas as derivadas segundas da função. Aqui utilizamos dois *GradientTape* para calcularmos as derivadas segundas. Isso foi necessário devido ao fato de termos que transformar para coordenadas cartesianas para evitarmos problemas com pontos mal definidos devido ao ângulo θ , levando ao treinamento errático com pesos estagnados num ponto trivial. Como os textos explicativos no *listing* afirmam, o *tensorflow* calcula as derivadas em coordenadas cilíndricas internamente e reatorna os valores já convertidos. Dessa forma, o modelo é treinado em coordenadas cartesianas e a equação diferencial será resolvida integralmente em coordenadas cilíndricas.

Listing 4: Programação da EDP no *tensorflow*.

```

1 @tf.function
2 def helmholtz_residual(model, x, y, z):
3     """Calcula o residuo da EDP em coordenadas cartesianas para
4        evitar a singularidade em r=0"""
5     x = tf.reshape(x, [-1, 1])
6     y = tf.reshape(y, [-1, 1])
7     z = tf.reshape(z, [-1, 1])
8     with tf.GradientTape(persistent=True) as tape2:
9         tape2.watch([x, y, z])
10        with tf.GradientTape(persistent=True) as tape1:
11            tape1.watch([x, y, z])
12            u = model(tf.concat([x, y, z], axis=1))
13            u_x = tape1.gradient(u, x)
14            u_y = tape1.gradient(u, y)
15            u_z = tape1.gradient(u, z)
16        u_xx = tape2.gradient(u_x, x)
17        u_yy = tape2.gradient(u_y, y)
18        u_zz = tape2.gradient(u_z, z)
19    del tape1

```

```

20 del tape2
21 # Equacao de Helmholtz em Cartesianas ajustada pela escala adimensional (r_aH)
22 # d^2u/dx^2 + d^2u/dy^2 + (r_aH)^2 * d^2u/dz^2 + ksq * u = 0
23 residual = u_xx + u_yy + (r_aH**2) * u_zz + ksq * u
24 return residual
25 # Solucao exata
26 def exact_solution(r, theta, z):
27     k_z = np.pi
28     return j0(k_r * r) * np.cos(k_z * z)

```

No listing 5 temos a definição da taxa de perda total. Introduzimos a fonte de nêutrons no ponto central no qual o valor de $u(r=0, \theta=0, z=0) = 1$, que coincide em coordenadas cartesianas em $(x=0, y=0, z=0)$. A função *compute_loss* calcula a taxa total de perda do modelo e os retorna com os respectivos pesos multiplicados. Essa é a função que será utilizada para otimizar o algoritmo e sua minimização é adotada como critério de convergência utilizando-se a Eq. (46). Um valor a partir de 10^{-5} é considerado um bom critério para convergência e, como veremos, refletirá na reprodução dos resultados analíticos até uma certa acurácia.

Listing 5: Funções de perda e cálculos de erros e resíduos.

```

1 # Funcao de perda
2 def compute_loss(model, boundary_points, boundary_values, domain_points):
3     # Conversao de Fronteira
4     r_b, theta_b, z_b = tf.split(boundary_points, 3, axis=1)
5     x_b, y_b, z_b = r_b * tf.cos(theta_b), r_b * tf.sin(theta_b), z_b
6     u_pred_b = model(tf.concat([x_b, y_b, z_b], axis=1))
7     loss_bc = tf.reduce_mean(tf.square(u_pred_b -
8                                     tf.reshape(boundary_values, (-1, 1))))
9     # Perda no Centro
10    center_point = tf.constant([[0.0, 0.0, 0.0]], dtype=tf.float64)
11    center_pred = model(center_point)
12    center_exact = tf.constant([[exact_solution(0., 0., 0.)]], dtype=tf.float64)
13    loss_center = tf.reduce_mean(tf.square(center_pred - center_exact))
14    # Residuo no Dominio (Convertido para Cartesianas)
15    r, theta, z = tf.split(domain_points, 3, axis=1)
16    x = r * tf.cos(theta)
17    y = r * tf.sin(theta)
18    residual = helmholtz_residual(model, x, y, z)
19    loss_pde = tf.reduce_mean(tf.square(residual))
20    return (wbc * loss_bc, wpde * loss_pde, wcenter * loss_center)

```

Diferentemente do treinamento das redes neurais tradicionais em que temos um conjunto de dados para treinamento e, após fazê-lo, utilizamos o modelo para fazer previsões, as PINNs utilizam a equação diferencial (ou equação física) para treinar ao mesmo tempo que faz previsões[9]. Isso só é possível porque a equação diferencial física fornece a lei que governa a EDP em todos os pontos do espaço em que queremos treinar a rede neural. Recentemente, houve uma explosão de trabalhos empregando PINNs para resolver EDPs em 2D na literatura[18]. Nota-se que ao empregar as EDPs normalizadas em módulo 1, o número de pontos de treinamento não precisa ser tão grande e a convergência ocorre relativamente de forma rápida. Em 3D, as coisas são diferentes. O fato de adicionarmos mais uma dimensão faz com que devamos adicionar muitos pontos para atingirmos certa acurácia. Contudo, realizar o treinamento ao mesmo tempo que as condições de contorno sejam fixadas nem sempre garante uma convergência computacional rápida e estável. Para tanto, devemos fazer um pré-treinamento apenas com as condições de contorno para que os parâmetros de treino reconheçam desde o início os valores da função explicitados pelas condições de contorno. Isso garante que a PINN convergirá em tempo hábil, como veremos a seguir. No listing 6 vemos a forma das funções de pré-treinamento e de treinamento de nossa rede neural. note a inclusão da função “tf.clip_by_global_norm” dentro do calculo do gradiente, o que garante a prevenção de “surto” no gradiente. Além disso, veja que a perda total na otimização de pré-treino é definida como a perda nas faces com peso 100 mais a perda no ponto central com peso 10. Isso garante que os parâmetros iniciais para esses pontos já terão parâmetros pré-inicializados para que o modelo a ser treinado não comece tudo a partir do zero.

Listing 6: Funções de pré-treino e treinamento das PINNs.

```

1 def pretrain(model, boundary_points, boundary_values, epochs=1000):
2     """Pre-treino ajustado com conversao para coordenadas Cartesianas"""
3     r_b, theta_b, z_b = tf.split(tf.convert_to_tensor(boundary_points,
4                                                         dtype=tf.float64), 3, axis=1)

```

```

5  x_b, y_b, z_b = r_b * tf.cos(theta_b), r_b * tf.sin(theta_b), z_b
6  cart_boundary = tf.concat([x_b, y_b, z_b], axis=1)
7  boundary_values = tf.convert_to_tensor(boundary_values, dtype=tf.float64)
8  center_point = tf.constant([[0.0, 0.0, 0.0]], dtype=tf.float64)
9  center_exact = tf.constant([[1.0]], dtype=tf.float64)
10 optimizer = tf.keras.optimizers.Adam(learning_rate=1e-3)
11 for epoch in range(epochs):
12     with tf.GradientTape() as tape:
13         pred_boundary = model(cart_boundary)
14         loss_boundary = tf.reduce_mean(tf.square(pred_boundary -
15                                                     boundary_values))
16         pred_center = model(center_point)
17         loss_center = tf.reduce_mean(tf.square(pred_center - center_exact))
18         loss = 100.0 * loss_boundary + 10.0 * loss_center
19         grads = tape.gradient(loss, model.trainable_variables)
20         grads, _ = tf.clip_by_global_norm(grads, 1.0)
21         optimizer.apply_gradients(zip(grads, model.trainable_variables))
22         if epoch % epochs == 0:
23             print(f"Fim do pre-treino | Perda Total: {loss:.4e}")
24     return model
25
26 def train_model(model, boundary_points, boundary_values, domain_points,
27                 epochs=30000, lr=2e-3):
28     lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
29         initial_learning_rate=lr, decay_steps=1500, decay_rate=0.92)
30     optimizer = tf.keras.optimizers.Adam(learning_rate=lr_schedule)
31     loss_history = []
32     for epoch in range(1, epochs + 1):
33         with tf.GradientTape() as tape:
34             loss_vec = compute_loss(model, boundary_points, boundary_values,
35                                     domain_points)
36             loss = tf.add_n(loss_vec)
37             grads = tape.gradient(loss, model.trainable_variables)
38             grads, _ = tf.clip_by_global_norm(grads, 1.0)
39             optimizer.apply_gradients(zip(grads, model.trainable_variables))
40             loss_history.append(loss.numpy().item())
41             if epoch % 1000 == 0:
42                 print(f"Epoca {epoch:5d} | Perda Total: {loss_history[-1]:.4e}")
43     return loss_history

```

Agora, vamos construir o modelo com os pontos nos locais certos, gerando a Figura 4. Para tanto, basta digitar os comandos do listing 7 logo abaixo. Temos como saída também um esquemático com as redes neurais geradas através da instância “tf.keras.utils.plot_model”, o que gera a Figura 3. Note o comando %matplotlib ipynb que é um comando “mágico”, típico do uso do *jupyter notebook*, para efeitos de visualizações interativas. Aconselhamos o uso do *jupyter* para as simulações pois facilitam na identificação de erros.

Listing 7: Para gerar o modelo inicial com os pontos distribuídos e gerar a figura 4.

```

1 %matplotlib ipynb
2 tf.keras.backend.clear_session()
3 # Construir modelo
4 model = build_model()
5 boundary_points, boundary_values, domain_points = generate_data()
6 model.summary()
7 tf.keras.utils.plot_model(model, to_file='model_plot.png', show_shapes=True,
8                             show_layer_names=True, show_dtype=True,
9                             show_layer_activations=True)

```

Vamos então preparar a parte de visualização dos resultados. Temos no listing 8 a função “plot_results” na qual passamos o modelo treinado e o “loss_history” para visualização. As saídas estão em “resultados.png”, “resultados_2D.png” e “loss_rate.png”.

Listing 8: Visualização dos resultados após o treinamento.

```

1 def plot_results(model, loss_history):
2     font_size = 1

```

```

3 plt.style.use('seaborn-v0_8')
4 plt.rcParams.update({'font.size': font_size})
5 # ===== Configuracao dos dados =====
6 n = 50
7 r = np.linspace(0., 1., n)
8 theta = np.linspace(-np.pi, np.pi, n)
9 z = np.zeros_like(r) # z=0 para visualizacao 2D
10 # Criar malha 2D
11 R_mesh, Theta_mesh = np.meshgrid(r, theta)
12 Z_mesh = np.zeros_like(R_mesh)
13 # Converter para coordenadas cartesianas para visualizacao
14 X_mesh, Y_mesh, _ = cartesian(R_mesh, Theta_mesh, Z_mesh)
15 points_cartesian = np.column_stack([X_mesh.ravel(), Y_mesh.ravel(),
16                                     Z_mesh.ravel()])
17 # ===== Calcular solucoes =====
18 U_pred = model(points_cartesian).numpy().reshape(R_mesh.shape)
19 # Solucao exata
20 U_exact = exact_solution(R_mesh.ravel(), Theta_mesh.ravel(), Z_mesh.ravel())
21 U_exact = U_exact.reshape(R_mesh.shape)
22 # Diferenca
23 diff = U_pred - U_exact
24 # ===== Analise Quantitativa =====
25 max_diff = np.max(np.abs(diff))
26 mse = np.mean(diff**2)
27 print(f"\nMetricas de Erro:")
28 print(f"Erro Maximo Absoluto: {max_diff:.4e}")
29 print(f"Erro Medio Quadratico (MSE): {mse:.4e}")
30 # ===== Visualizacao 3D =====
31 # Aumentamos a figura (24x8) para dar espaco aos rotulos maiores
32 fig = plt.figure(figsize=(24, 8))
33 # 1. Solucao PINN
34 ax1 = fig.add_subplot(131, projection='3d')
35 surf1 = ax1.plot_surface(X_mesh, Y_mesh, U_pred, cmap='viridis', rstride=1,
36                          cstride=1, alpha=0.8)
37 ax1.set_title('Solucao PINN', fontsize=font_size, pad=12)
38 ax1.set_xlabel('x', fontsize=font_size, labelpad=12)
39 ax1.set_ylabel('y', fontsize=font_size, labelpad=12)
40 ax1.set_zlabel('u(x,y,0)', fontsize=font_size, labelpad=12)
41 ax1.tick_params(axis='both', which='major', labelsize=font_size)
42 cbar1 = fig.colorbar(surf1, ax=ax1, shrink=0.5, aspect=5, pad=0.1)
43 cbar1.ax.tick_params(labelsize=font_size)
44 # 2. Solucao Exata
45 ax2 = fig.add_subplot(132, projection='3d')
46 surf2 = ax2.plot_surface(X_mesh, Y_mesh, U_exact, cmap='viridis', rstride=1,
47                          cstride=1, alpha=0.8)
48 ax2.set_title('Solucao Exata', fontsize=font_size, pad=12)
49 ax2.set_xlabel('x', fontsize=font_size, labelpad=12)
50 ax2.set_ylabel('y', fontsize=font_size, labelpad=12)
51 ax2.set_zlabel('u(x,y,0)', fontsize=font_size, labelpad=12)
52 ax2.tick_params(axis='both', which='major', labelsize=font_size)
53 cbar2 = fig.colorbar(surf2, ax=ax2, shrink=0.5, aspect=5, pad=0.1)
54 cbar2.ax.tick_params(labelsize=font_size)
55 # 3. Diferenca
56 ax3 = fig.add_subplot(133, projection='3d')
57 surf3 = ax3.plot_surface(X_mesh, Y_mesh, diff, cmap='coolwarm', rstride=1,
58                          cstride=1, alpha=0.8)
59 ax3.set_title(f'Diferenca (MAX: {max_diff:.2e})', fontsize=font_size, pad=12)
60 ax3.set_xlabel('x', fontsize=font_size, labelpad=12)
61 ax3.set_ylabel('y', fontsize=font_size, labelpad=12)
62 # REMOVER zlabel e zticks para limpar a visualizacao
63 ax3.set_zlabel('')
64 ax3.set_zticklabels([])
65 ax3.tick_params(axis='x', labelsize=font_size)
66 ax3.tick_params(axis='y', labelsize=font_size)
67 cbar3 = fig.colorbar(surf3, ax=ax3, shrink=0.5, aspect=5, pad=0.1)
68 cbar3.ax.tick_params(labelsize=font_size)

```

```

69 plt.tight_layout()
70 plt.savefig("resultados.png", dpi=300, bbox_inches='tight')
71 # ===== Visualizacao 2D =====
72 fig2 = plot_2d_results_imshow(model, 1., exact_solutio
73 # ===== Curva de Convergencia =====
74 fig3 = plt.figure(figsize=(12, 7))
75 plt.semilogy(loss_history, linewidth=2)
76 plt.title('Curva de Convergencia', fontsize=font_size, pad=12)
77 plt.xlabel('Epoca', fontsize=font_size, labelpad=10)
78 plt.ylabel('Perda (log)', fontsize=font_size, labelpad=10)
79 plt.tick_params(axis='both', which='major', labelsize=font_size)
80 plt.grid(True, which="both", ls="--")
81 plt.savefig("loss_rate.png", dpi=300, bbox_inches='tight')
82 plt.close('all')
83
84 def plot_2d_results_imshow(model, r, exact_solution):
85     font_size = 18
86     fig, axes = plt.subplots(1, 3, figsize=(22, 7))
87     for ax in axes:
88         ax.grid(True, linestyle='--', linewidth=0.3, alpha=0.3, color='gray')
89         ax.tick_params(axis='both', which='major', labelsize=font_size)
90     # Grade cartesiana
91     n = 200
92     x = np.linspace(-r, r, n)
93     y = np.linspace(-r, r, n)
94     X, Y = np.meshgrid(x, y)
95     # Converter para coordenadas cilindricas
96     R = np.sqrt(X**2 + Y**2)
97     Theta = np.arctan2(Y, X)
98     Z = np.zeros_like(R)
99     # Mascara para dominio circular
100    mask = R <= r
101    # Preparar arrays de resultados
102    U_pred = np.zeros_like(R)
103    U_exact = np.zeros_like(R)
104    # Calcular apenas dentro do dominio
105    if np.any(mask):
106        # Pontos validos
107        R_valid = R[mask].flatten()
108        Theta_valid = Theta[mask].flatten()
109        Z_valid = Z[mask].flatten()
110        X_valid, Y_valid, Z_valid = cartesian(R_valid, Theta_valid, Z_valid)
111        # Preparar para modelo PINN
112        points_valid = np.column_stack([X_valid, Y_valid, Z_valid])
113        points_tensor = tf.convert_to_tensor(points_valid, dtype=tf.float64)
114        # Calcular solucoes
115        U_pred_valid = model(points_tensor).numpy().flatten()
116        U_exact_valid = exact_solution(R_valid, Theta_valid, Z_valid)
117        # Inserir nos arrays
118        U_pred[mask] = U_pred_valid
119        U_exact[mask] = U_exact_valid
120    # Diferenca
121    diff = U_pred - U_exact
122    # Mascara fora do dominio
123    U_pred_masked = np.ma.masked_where(~mask, U_pred)
124    U_exact_masked = np.ma.masked_where(~mask, U_exact)
125    diff_masked = np.ma.masked_where(~mask, diff)
126    # Plotar
127    extent = [-r, r, -r, r]
128    # --- GRAFICO 1 ---
129    im1 = axes[0].imshow(U_pred_masked, extent=extent, origin='lower',
130                        cmap='viridis', aspect='equal', interpolation='bicubic')
131    axes[0].set_title('Solucao PINN', fontsize=font_size, pad=12)
132    axes[0].set_xlabel('x', fontsize=font_size)
133    axes[0].set_ylabel('y', fontsize=font_size)
134    cbar1 = plt.colorbar(im1, ax=axes[0])

```

```

135     cbar1.ax.tick_params(labels=font_size) # Fonte da legenda da barra de cores
136     # --- GRAFICO 2 ---
137     im2 = axes[1].imshow(U_exact_masked, extent=extent, origin='lower',
138                          cmap='viridis', aspect='equal', interpolation='bicubic')
139     axes[1].set_title('Solucao Exata', fontsize=font_size, pad=12)
140     axes[1].set_xlabel('x', fontsize=font_size)
141     axes[1].set_ylabel('y', fontsize=font_size)
142     cbar2 = plt.colorbar(im2, ax=axes[1])
143     cbar2.ax.tick_params(labels=font_size)
144     # --- GRAFICO 3 ---
145     im3 = axes[2].imshow(diff_masked, extent=extent, origin='lower',
146                          cmap='coolwarm', aspect='equal', interpolation='bicubic')
147     axes[2].set_title(f'Diferenca (max: {np.max(np.abs(diff)):.2e})',
148                      fontsize=font_size, pad=12)
149     axes[2].set_xlabel('x', fontsize=font_size)
150     axes[2].set_ylabel('y', fontsize=font_size)
151     cbar3 = plt.colorbar(im3, ax=axes[2])
152     cbar3.ax.tick_params(labels=font_size)
153     plt.tight_layout()
154     plt.savefig("resultados_2d.png", dpi=300, bbox_inches='tight')
155     return fig

```

Finalmente, no listing 9 fazemos o pré-treinamento e o treinamento da PINN para gerar os resultados e visualizá-los nos gráficos.

Listing 9: Pré-treinamento e treinamento da PINN e a geração das figuras.

```

1 # Pre-treinar
2 model = pretrain(model, boundary_points, boundary_values, epochs=1200)
3
4 # Treinar o modelo
5 loss_history = train_model(model, boundary_points, boundary_values,
6                             domain_points, epochs=60000, lr=2e-3)
7
8 # Avaliacao final
9 final_relerror, final_absererror = compute_validation_error(model)
10 print(f"\nErro relativo final: {final_relerror:.6f}")
11 print(f"\nErro absoluto final: {final_absererror:.6f}")
12
13 # Visualizacao
14 plot_results(model, loss_history)

```

Na Figura 5, mostramos a taxa de perda (“loss_rate”) do modelo treinado. Este é calculado na função “compute_loss” no listing 5 onde temos “loss_bc” para *boundary conditions*, “loss_center” para o ponto central e “loss_pde” para a equação diferencial parcial em questão. Essa função é chamada no listing 5 para ser otimizada pelo *tensorflow*.

Na Figura 6, mostramos os resultados convergidos da PINN treinada para o reator cilíndrico para o plano definido por $z = 0$. Note a delimitação das bordas no plano em $z = 0$, bem como o formato da superfície em comparação com o resultado exato. Sem o pré-treinamento das condições de contorno, mesmo colocando-se um número muito grande de Épocas para treino, haveria deformações nestes locais. A boa convergência pode ser visualizada através do painel que mostra a diferença de valores para um conjunto de pontos pré-determinados. Note que a diferença máxima ficou em torno de 1.18×10^{-3} , o que indica uma convergência razoável. Para este ensaio utilizamos cerca de 30000

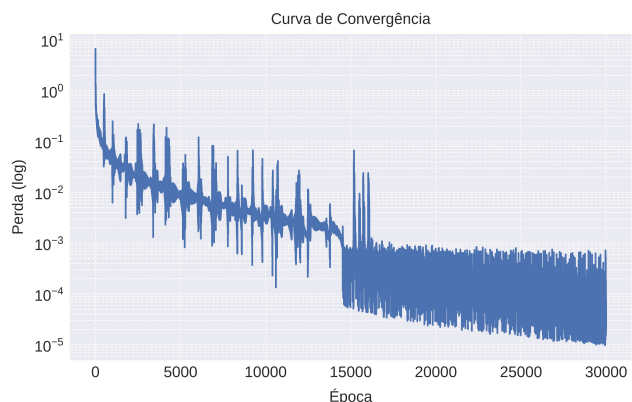


Figura 5: Taxa de perda (“loss_rate”) para o ensaio do reator cilíndrico com 30000 Épocas.

Épocas. Mostramos também nesta figura a projeção no plano XY num mapa de superfície para vermos manchas quentes e frias de acordo com as diferenças com relação

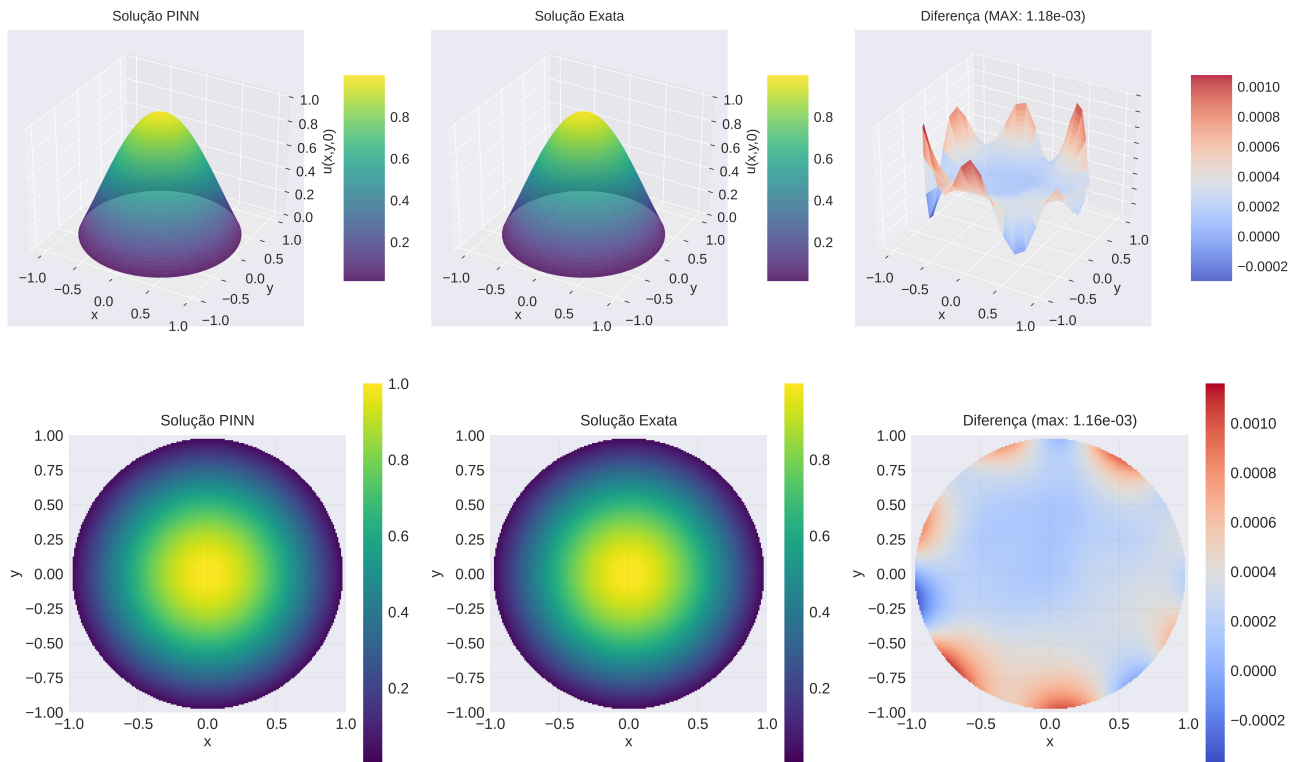


Figura 6: Resultados após a convergência do modelo para o plano definido por $z = 0$.

à solução exata. Vemos que o modelo parece mostrar boa concordância com os valores exatos próximo à região central e um erro um pouco maior nas regiões um pouco mais afastadas do centro, mas com boa concordância nas fronteiras.

Quando normalizamos a equação diferencial do reator cilíndrico vimos que $k^2 = (aB)^2 = k_r^2 + k_z^2$, com $k_r \approx 2,405$ e $k_z = \pi r_{aH}$. Dessa forma, poderemos calcular o raio do reator cilíndrico para um $r_{aH} = 0.5$, o que implica um cilindro com $H = 2a$. Para tanto, se considerarmos um combustível típico de um reator comercial, a base de Urânio, a literatura nos fornece para uma usina de água leve – como um PWR/BWR [5], que é o caso típico de usinas comerciais – o valor de referência para o comprimento de difusão dos nêutrons térmicos na água é de cerca de 2,5 cm. Essa ordem de grandeza para o comprimento de difusão é corroborada em livros-texto de física de reatores nucleares que listam para água leve $L \approx 2,54$ cm[19, 20]. Por outro lado, o fator de multiplicação infinito assume valores no intervalo $k_\infty = 1,02 - 1,05$, com valor típico de 1,03[19, 20]. Considerando esses valores e as Eqs.(21), (22) e (28), teremos para o reator cilíndrico em que $H = 2a$ o raio $a = 26$ cm e a altura $H = 52$ cm. Há algumas limitações no modelo teórico aqui considerado se compararmos com as dimensões dos reatores reais, pois não incluímos em nossa análise ingredientes obrigatórios nos reatores reais tais como substâncias moderadoras (água pesada ou grafite); não levamos em consideração a heterogeneidade entre as células de combustível; no caso real, o

fluxo costuma anular numa fronteira extrapolada, não exatamente na superfície geométrica como aparece na hipótese do nosso modelo. Essas nuances modificariam bastante os resultados obtidos para o tamanho do reator. Por esse motivo, essa situação idealizada é chamada de reator *nu* uniforme, mas possui elevada importância nas simulações reais para o entendimento do reator como um todo. A simplificação na análise se deve ao caráter pedagógico deste trabalho, ou seja, consideramos até o momento a equação de difusão de nêutrons aplicada a reatores nucleares na sua forma mais simples; assim, uma alternativa seria, além de reconsiderar as limitações mencionadas, utilizarmos, por exemplo, a equação de difusão de nêutrons multigrupos.

6. Considerações Finais e Perspectivas

Neste trabalho foram apresentados de forma pedagógica dois assuntos que despertam bastante interesse da comunidade acadêmica, quais sejam: reatores nucleares e inteligência artificial. Nesse processo, é esperado que o leitor, mesmo o menos familiarizado com tais conteúdos, compreenda a essência da física de nêutrons aplicada ao reator nuclear, bem como os princípios básicos das redes neurais informadas por física. O material se destaca pela forma pedagógica que é apresentado, isto é, tanto as passagens matemáticas concernentes à equação de difusão de nêutrons, como a discussão acerca das PINNs foram construídas com bastante detalhe, com o objetivo de ser suficientemente inteligível. Nesse bojo, o problema

do reator nuclear foi discutido analiticamente para o formato cilíndrico, o qual foi também solucionado com o uso da PINN, e os resultados obtidos por meio de ambos os métodos foram compatíveis. As limitações do modelo utilizado ficaram evidentes quando o tamanho estimado foi comparado com o tamanho real do reator; tendo tais diferenças origem nas hipóteses do arabcouço simplificado. Assim, como perspectiva, apontamos o aprofundamento, com o uso de PINNs, na equação de nêutrons multigrupo, assim seria possível tratarmos situações mais realísticas. Nessa perspectiva, o material apresentado apresenta potencial de ser utilizado por estudantes de graduação em física e engenharia que estejam interessados a iniciar estudos em física de reatores nucleares.

Agradecimento

Os autores agradecem ao Deputado Federal José Airtton Félix Cirilo que através da emenda parlamentar Nº 23310003 tornou possível a realização deste trabalho.

Disponibilidade de dados

Todo o conjunto de dados que dá suporte ao presente texto foi publicado no corpo do artigo

Referências

- [1] G.I. Bell e S. Glasstone, *Nuclear reactor theory* (Van Nostrand Reinhold, New York, 1970).
- [2] J.J. Duderstadt e L.J. Hamilton, *Nuclear reactor analysis* (John Wiley & Sons, New York, 1976).
- [3] A.F. Henry, *Nuclear-reactor analysis* (MIT Press, Cambridge, 1975).
- [4] E.E. Lewis, *Fundamentals of Nuclear Reactor Physics* (Academic Press, San Diego, 2008).
- [5] J.C. Lee, em: *Wiley Encyclopedia of Electrical and Electronics Engineering* (Wiley, Hoboken, 2016), 3 ed.
- [6] L. Vlahos, H. Isliker, Y. Kominis e K. Hizanidis, arXiv:0805.0419 (2008).
- [7] W.M. Stacey, *Nuclear Reactor Physics* (Wiley, New York, 2001).
- [8] J.R. Lamarsh e A.J. Baratta, *Introduction to Nuclear Engineering* (Prentice Hall, Upper Saddle River, 2001), 3 ed.
- [9] M. Raissi, P. Perdikaris e G.E. Karniadakis, *J. Comp. Phys.* **378**, 686 (2019).
- [10] X. Jin, S. Cai, H. Li e G. Karniadakis, *Journal of Computational Physics* **426**, 109951 (2021).
- [11] H. Jin, M. Mattheakis e P. Protopapas, em: *2022 International Joint Conference on Neural Networks – IJCNN* (Piscataway, 2022).
- [12] G. Kissas, Y. Yang, E. Hwang, W.R. Witschey, J.A. Detre e P. Perdikaris, *Computer Methods in Applied Mechanics and Engineering* **358**, 112623 (2020).
- [13] E. Schiassi, M. Florio, B.D. Ganapol, P. Picca e R. Furfaro, *Annals of Nuclear Energy* **167**, 108833 (2022).
- [14] J. Wang, X. Peng, Z. Chen, B. Zhou, Y. Zhou e N. Zhou, *Annals of Nuclear Energy* **176**, 109234 (2022).
- [15] Y. Yang, H. Gong, S. Zhang, Q. Yang, Z. Chen, Q. He e Q. Li, *Annals of Nuclear Energy* **183**, 109656 (2023).
- [16] R.E. Marshak, *Journal of Applied Physics* **18**, 185 (1947).
- [17] G.B. Arfken e H.J. Weber, *Mathematical Methods for Physicists* (Elsevier, Amsterdam/London, 2005), 6 ed.
- [18] H. Baty, arXiv:2403.00599 (2024).
- [19] J.A. DeJuren e H. Rosenwasser, *Journal of Research of the National Bureau of Standards* **51**, 203 (1953).
- [20] M.M. Bretscher, *Proceedings of the Indiana Academy of Science* **72**, 249 (1962).