FCT – Faculdade de Ciências e Tecnologia

DMC – Departamento de Matemática e Computação

Bacharelado em Ciência da Computação

Gilmar Francisco de Oliveira Santos

Aplicação de Redes Neurais Artificiais na Solução Numérica de Equações Diferenciais

Revisão Bibliográfica

Orientador: Prof. Dr. Cassio Machiaveli Oishi

2020

Aplicação de Redes Neurais Artificiais na Solução Numérica de Equações Diferenciais

Autor: Gilmar Francisco de Oliveira Santos

Orientador: Prof. Dr. Cassio Machiaveli Oishi

Responsável: Prof. Dr. Danilo Medeiros Eler

Curso: Bacharelado em Ciência da Computação

Instituição: Universidade Estadual Paulista

Faculdade de Ciências e Tecnologia

Departamento de Matemática e Computação

Presidente Prudente, 2020.

Gilmar Francisco de Oliveira Santos (aluno)

Prof. Dr. Cassio Machiaveli Oishi (orientador)

Prof. Dr. Danilo Medeiros Eler (responsável)

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

A solução de (EDs) Equações Diferenciais é relevante a muitos segmentos dada a grande aplicação na representação dos mais diferentes fenômenos, Métodos Numéricos são largamente utilizados para solução dessas equações, com destaque ao Método das Diferenças Finitas que é um dos mais simples e é usado como base para a construção de diversos outros métodos mais específicos e robustos para o problema tratado.

A aplicação de RNAs (Redes Neurais Artificiais) para a aproximação de solução de EDs foi Lee e Kang em 1989, porém, o tópico ressurgiu recentemente, dado que só atualmente dispomos dos recursos computacionais necessários para implementar eficientemente tais técnicas. RNAs podem ser utilizadas como Aproximadores Universais para funções como demonstrado por Hornik (1989), essa característica permite a sua aplicação na solução de EDs.

Uma dos objetivos relevantes quando analisamos os problemas que envolvem esse tipo de equação é a necessidade de resultados precisos e em grande velocidade, logo o desenvolvimento novas técnicas para atingir esses objetivos se faz necessário, com o advento das RNAs e colaborações interessantes podem ser realizadas com os métodos clássicos várias possibilidades para atingir esses objetivos são abertas.

O presente trabalho visa estudar diferentes (MNCs) Métodos Numéricos Clássicos para a solução de EDs e desenvolver técnicas para aproximar a solução dessas equações ou colaborar com MNCs, usando de RNAs.

As RNAs em muitos dos casos não são adequadas para resolver diretamente o problema, mas podem ser usadas em casos específicos onde os métodos numéricos apresentem dificuldades, como é o caso onde se precisa resolver EDs em domínios complexos, como mostrado em [10] e [34], sendo a que a RNA obtida pelo treinamento mantém propriedades

interessantes como diferenciabilidade, continuidade e independência de malha.[33]

Outro caso que mostra diversas possibilidades é aplicar RNAs para fornecer parâmetros para ajudar, seja na estabilidade ou velocidade de convergência dos MNCs, cujas propriedades são matematicamente demonstradas, o que os garante maior confiabilidade. Podemos aplicar RNAs para gerar "chutes"iniciais, como estimadores de parâmetros ou mesmo para gerar malhas computacionais [56].

1.1 Objetivos do Trabalho

Esse projeto tem como objetivo principal estudar e aplicar MNCs para a solução de EDs aliados a técnicas de RNAs, focando principalmente na solução de EDPs (Equações Diferenciais Parciais).

Para tanto, serão estudadas e comparadas diferentes técnicas para solução dessas equações apresentadas na literatura [33, 34, 37, 5, 35, 14], com o conhecimento adquirido, desenvolver uma técnica que utiliza de RNAs para solucionar esses problemas ou que colabore diretamente na estimação de parâmetros que acelerem a convergência ou estabilização dos MNCs, como por exemplo é mostrado em [20] onde Dehghanpour et al. propõe o uso de uma RNA para calcular os coeficientes ótimos para o método Runge-Kutta de terceira ordem.

1.1.1 Objetivos Específicos

- Estudar os conceitos básicos para a solução numérica de EDOs e EDPs, desde de sua modelagem computacional, simulação eficiente e visualização dos dados.
- Estudar conceitos teóricos básicos sobre *Machine Learning* e *Deep Learning*
- Implementar e comparar diferentes técnicas usando de Redes Neurais para solucionar EDs e desenvolver uma abordagem onde a RNA colabore na convergência ou estabilização de um MNC.

1.2 Organização deste Trabalho

Esta revisão bibliográfica será organizada em 3 capítulos e o restante do texto segue da seguintes forma:

- O Capítulo 2 apresenta a fundamentação teórica acerca de RNAs, EDs e técnicas numéricas para resolução de problemas relacionados, o que é necessário para uma melhor compreensão do que será abordado nos capítulos seguintes;
- O Capítulo 3 apresenta uma revisão bibliográfica das principais aplicações de RNAs na solução de EDs;
- No Capítulo 4 traremos uma conclusão do que fora estudado para escrever este texto.

Capítulo 2

Fundamentação Teórica

2.1 Redes Neurais Artificiais

Simular o cérebro como mostra [4] foi um dos objetivos principais do campo da Inteligência Artificial, desde seis primórdios, as RNAs são a consequência direta e de maior sucesso do mesmo, o cérebro humano era visto como uma RNA composta por diversos nós, os chamados neurônios, como mostra a Figura 2.1, e estes conectam-se entre si formando sinapses, a informação, ou seja, sinais elétricos, em um neurônio biológico entra por seus dentritos, chega ao corpo do neurônio onde os sinais vindos de diferentes neurônios são combinados, o sinal gerado é então transmitido para os outros neurônio a ele conectados por meio de seu axônio, gerando então uma resposta de ativação ou inibição dos neurônios seguintes [50, 39].

Segundo Tatibana e Kaetsu (2012) o cérebro humano possui cerca de 10 bilhões de neurônios, sendo que estes são fatores determinantes sobre como um ser humano age e raciocina.

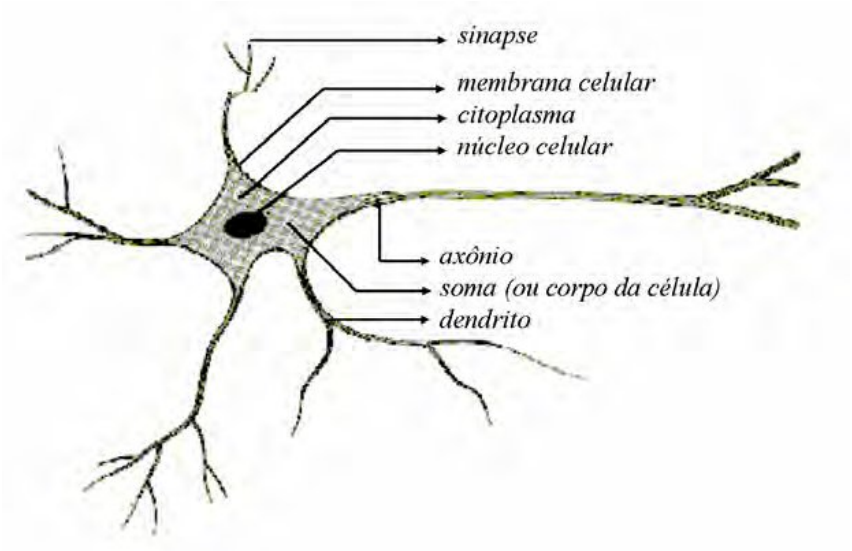


Figura 2.1: Neurônio Biológico;
Fonte: Tatibana e Kaetsu (2012)

O modelo de RNAs *Perceptron* foi desenvolvido por Rosenblatt (1958), possui um conceito de neurônio artificial ainda utilizado atualmente, nele é possível aprender apenas funções lineares. Cada neurônio computa uma soma ponderada de suas entradas, e repassa a soma em uma função de ativação não-linear com limiarização (as chamadas funções de ativação). [38, 39].

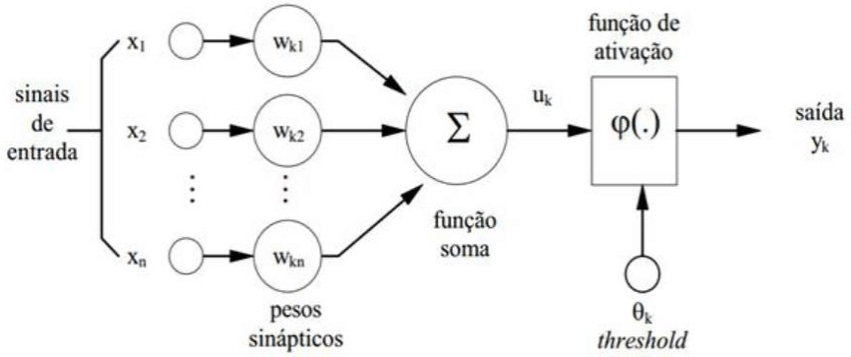


Figura 2.2: Neurônio Artifical;
Fonte: Haykin (2001)

Onde n é o número de entradas do neurônio, w_i é o peso associado à entrada xi, e θ é o limiar (*threshold*) do neurônio".

Para gerarmos a saída de um *Perceptron* calculamos:

$$y = \sigma(\sum_{i=1}^{n} W_i * X_i)$$

Para treinar um *Perceptron* utilizamos a regra delta de Hebb, que consiste em [39]:

$$W^{t+1} = W^{t+1} - \alpha * (d - y) * X$$

onde W representa o peso das conexões entre neurônios (A notação W será utilizada para referenciar os pesos de uma RNA, porém também utilizaremos θ para representar o conjunto dos pesos e bias e uma RNA), α é um parâmetro que indica quanta mudança será feita nos pesos, d é o valor desejado para a saída da rede, y é o valor real da saída e X representa os valores de entrada.

A associação de vários *Perceptron* forma o *Multilayer Perceptron*, o qual tem a capacidade para aprender funções não-lineares [4]:

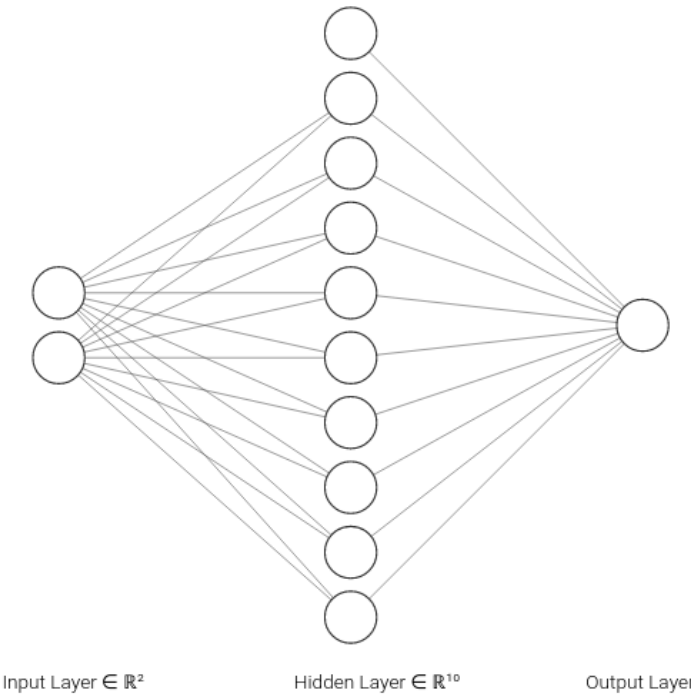


Figura 2.3: *Multilayer Perceptron* com apenas uma camada oculta

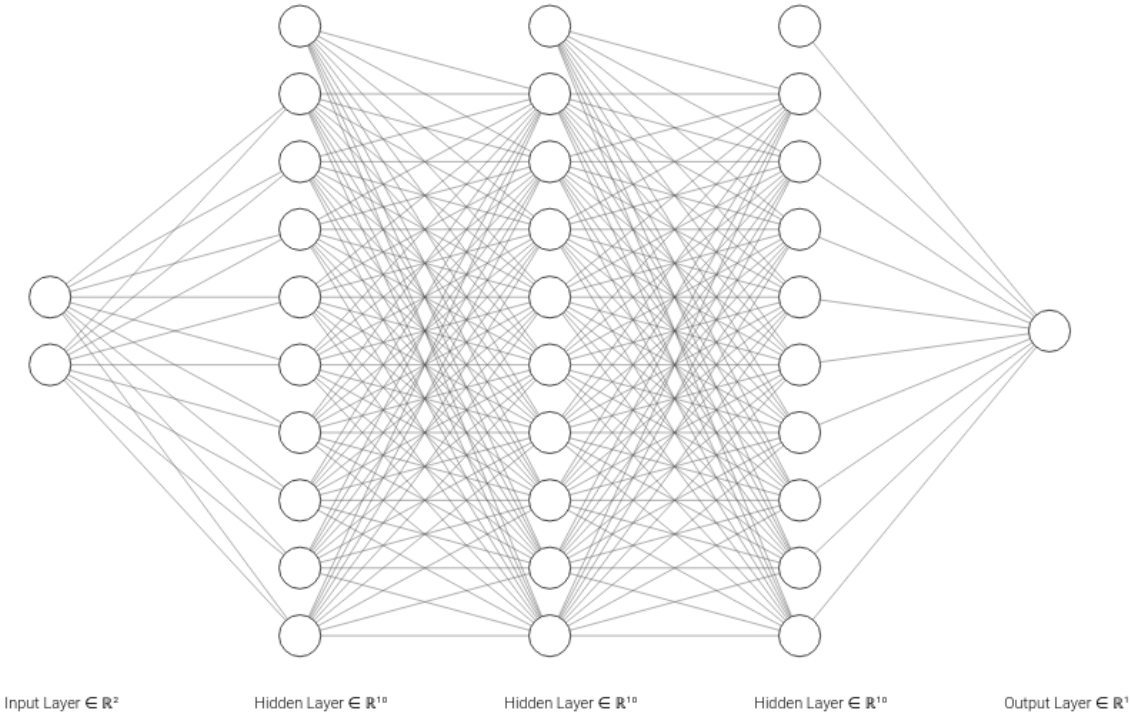


Figura 2.4: *Multilayer Perceptron* com 3 camadas ocultas

Como é mostrado em Figura 2.3 e na Figura 2.4, quando trabalhamos com RNAs com apenas uma camada oculta denominamos *Machine Learning*, quando termos mais camadas ocultas *Deep Learning* [25].

2.1.1 Funções de Ativação

As funções de ativação são utilizadas para processar a multiplicação de matrizes vistas anteriormente, que correspondem ao valor z, e a saída dessa função será o valor fornecido aos neurônios da próxima camada, elas são geralmente de natureza não-lineares, pois assim permitem que a RNA aprenda uma grande variedade de funções [25, 39]. As principais funções de ativação são e os seus formatos podem ser vistos na Figura 2.5:

- Threshold (função de ativação do *Perceptron* original)

$$\sigma(z) = \begin{cases} 1 & \text{se} \quad z > 0 \\ -1 & \text{c.c} \end{cases}$$

- Sigmoid

$$\sigma(z) = \frac{1}{1 + e^{(-z)}}$$

- Tanh

$$\sigma(z) = \frac{e^{(z)} - e^{(-z)}}{e^{(z)} - e^{(-z)}}$$

- ReLu

$$\sigma(z) = max(z, 0)$$

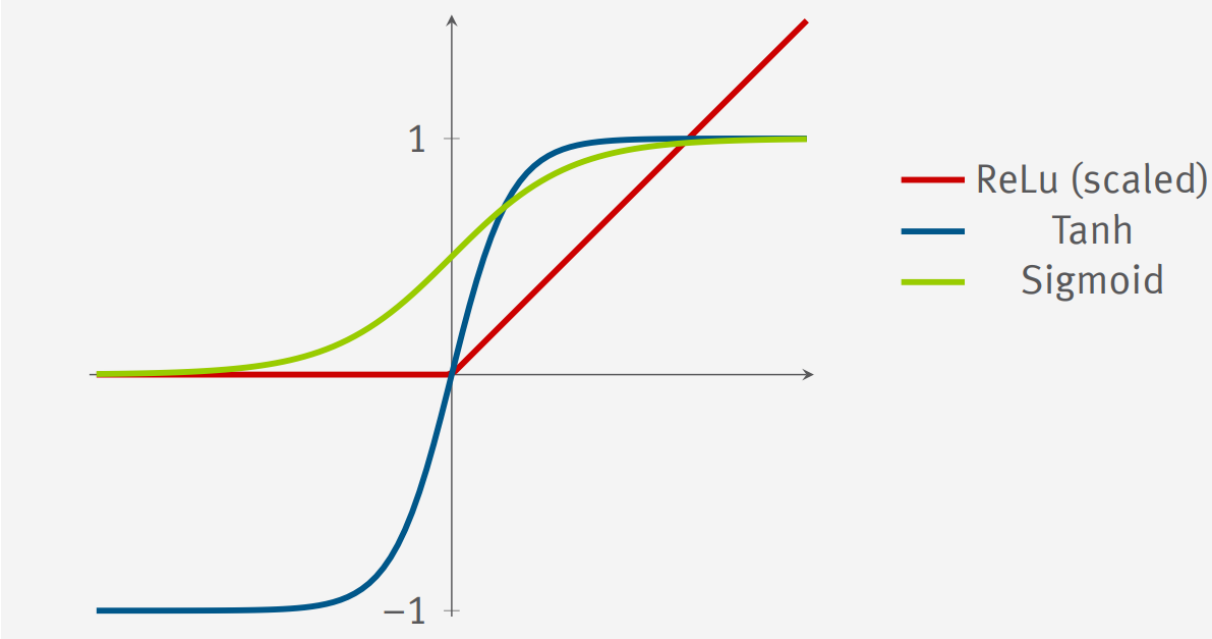


Figura 2.5: Funções de Ativação Típicas

2.1.2 Arquitetura

A arquitetura de uma RNA, diz respeito a como é feita a disposição dos neurônios dentro da RNA e a conexão feita entre estes.

As RNAs são organizadas em camadas de neurônios, onde cada camada apresenta um determinado número de neurônios, sendo que qualquer RNA apresenta no mínimo duas camadas, uma chamada de entrada, onde valores são fornecidos, para que sejam processados e outros valores são gerados e são retornados por uma camada de saída. Porém para aplicações realmente úteis em termos práticos, necessitamos de uma RNA com pelo menos uma camada oculta, dado que menos que isso não forneceria capacidade a RNA de trabalhar com problemas linearmente separáveis. As camadas entre a de entrada e de saída são chamadas de ocultas e nelas é que ocorrem processamentos utilizando funções de ativação [4, 39].

Cada camada se relaciona com a próxima por meio de conexões, essas conexões fornecem a saída dos neurônios de uma dada camada, como valores de entrada para os

neurônios da próxima, logo os dados fluem da camada de entrada até chegar a camada de saída, esse tipo de relacionamento é chamado de feedforward e será o utilizado por todas as seguinte sessões, também existe o tipo feedback, no qual neurônios podem formar ciclos em suas conexões e o fluxo de dados não é unidirecional, cuja utilizada está em recuperação de sinais [54] . Para o presente trabalho apenas a RNA completamente conectada será utilizada, seguindo as indicações presente na literatura [23, 33, 34], nesse tipo de RNA, todos os neurônios em uma camada estão ligados aos neurônios na camada seguinte, e uma das facilidades de usar essa arquitetura é que podemos simplesmente calcular a saída de uma RNA como uma multiplicação de matrizes.

2.1.3 *Backpropagation*

Mitchell (1997) define que um programa de computador que que é dito ter aprendido a partir de uma experiência E, com respeito a uma classe de tarefas T e tem uma performance P, se a performance com relação as tarefas em T, medida por P, melhora com a experiência E. No caso das RNAs na solução de EDs, apresentaremos medidas de performance baseadas nas equações e condições fornecidas para o problema, chamaremos essa medida de erro [33].

Idealmente para descobrimos a magnitude do erro temos que calcular a diferença entre a ativação de um nó de saída e o valor real desejado, para o dado nó. O Erro total é a soma dos erros para cada nó de saída, dado que não gostaríamos que valores negativos e positivos se cancelem, de forma que tenhamos uma falsa ideia de que encontramos valores baixos para os erros, primeiro elevamos os valores ao quadrado, somamos e então dividimos pela quantidade de elementos, chamando assim de EQM (Erro Quadrático Médio) (2.1), outras variações como o EQT (Erro Quadrático Total) (2.2) também é muito utilizada, o as direferenças são elevadas ao quadrado, somadas e então dividido por 2 [4].

$$EQM = \frac{1}{n}\sum_{i=1}^{n}(d_j - s_j)^2 \tag{2.1}$$

$$EQT = \frac{1}{2}\sum_{i=1}^{n}(d_j - s_j)^2 \tag{2.2}$$

$$\tag{2.3}$$

Onde d_j é o valor desejado para algum dos nós da camada saída e s_j a saída real

daquele nó.

A aprendizagem no *Perceptron* era feita simplesmente calculando O EQM na camada de saída, porém em redes do tipo *Multilayer Perceptron*, que são justamente as utilizadas neste trabalho dadas as limitações presentes no *Perceptron* original como é explorado em Minsky e Papert (1969), não saberíamos como usar o erro para atualizar os pesos nas camadas ocultas [4, 25, 39], para superar essa dificuldade surgiu então a Regra Delta Generalizada, que nos dá o indicativo de como o erro da camada de saída deve ser retro-propagado pela rede, daí o nome do algoritmo "*Backpropagation*". Na Seção 2.1.4 e Seção 2.3 veremos que o *Backpropagation* foi novamente generalizado se tornando assim um dos casos presentes na Diferenciação Automática.

Como explica Goodfellow et al. (2016) o treinamento de uma RNA consiste em fornecer determinado exemplo para a mesma, realizando assim o processo de feedforward, que gera um custo/erro, dado que sabemos o resultado esperado ou então temos uma informação de quão distantes estamos do resultado desejado, o custo é representado por $J(\theta)$, onde θ são os pesos da RNA, o algoritmo de *Backpropagation* permite que o erro seja retro-propagado da camada de saída até a primeira camada oculta e assim por meio do gradiente desse custo, podemos calcular quanto cada peso da RNA deve ser ajustado. Como necessitamos calcular derivadas em relação a diversos pesos, muitas vezes temos centenas ou até milhares deles, em que precisamos calcular derivas, computar de forma analítica pode computacionalmente custoso, então a Diferenciação Automática apresentada na Seção 2.3 é utilizada.

2.1.4 Otimizadores

Gradiente Descendente

Gradiente descendente (Haskell, 1944) é um algoritmo de otimização iterativo, que visa a minimização de uma função dada com relação aos parâmetros aos quais ela depende, na (2.4) o sinal de negativo é utilizado, pois o gradiente calculado nos dá a direção em que a função de custo cresce, como queremos minimizar, tomamos o oposto dessa direção [18, 25].

$$W^{t+1} = W^t - \alpha \frac{\partial L(W, X)}{\partial W} \tag{2.4}$$

Na (2.4) o $L(W, X)$ é a função de *loss*, ou seja, uma função que mede o erro da saída

atual da RNA, quando aplicada para as entradas de treinamento X. Podemos encontrar casos em que o membro $\frac{\partial L(W)}{\partial W}$ é representado por $\frac{\partial W}{\partial J(W)}$ cuja ideia é a mesma, queremos simplesmente encontrar a direção do gradiente para que possamos atualizar os pesos e minimizar a função de custo utilizada [39, 25].

A derivada do custo (função de *loss* $L(W)$) é calculada com relação a todos os pesos da RNA. A derivada é um conceito de Cálculo e refere-se à inclinação da função em um determinado ponto [26]. Precisamos conhecer a inclinação para que possamos conhecer a direção (sinal) para mover os valores dos coeficientes para obter um custo menor na próxima iteração [18, 6, 25].

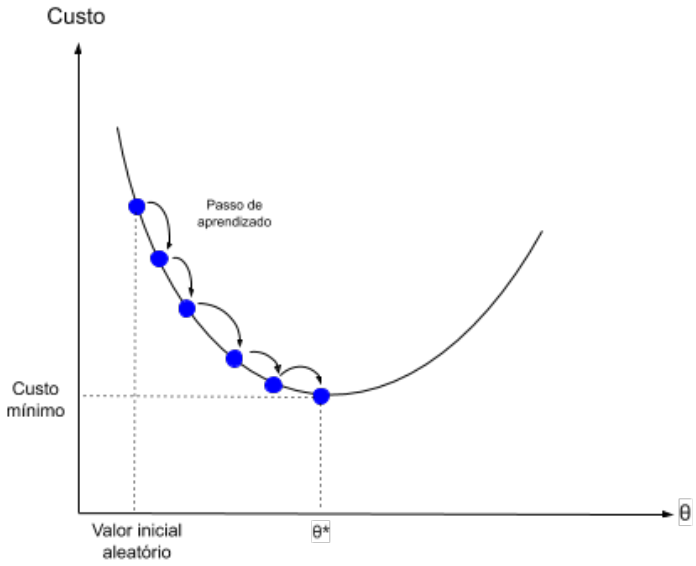


Figura 2.6: Descida do Gradiente em Direção ao mínimo
Fonte: https://sigmoidal.ai

Sabendo da derivada em que direção está em declive como mostrado na Fig. 2.6, podemos atualizar os valores dos coeficientes. Um parâmetro de taxa de aprendizagem (α) deve ser especificado e controla o quanto os coeficientes podem mudar em cada atualização. O valor de α deve ser escolhido com cuidado para a convergência não seja muito lenta, no caso de usarmos um valor muito pequeno, ou que não consigamos alcançar o mínimo, dado que um valor muito grande poderia levar o algoritmo a ficar oscilando no entorno do ponto de mínimo [18, 6].

Este processo é repetido até que o custo dos coeficientes (função de *loss*) seja 0.0 ou próximo o suficiente de zero, indicando que as saídas da RNA estão cada vez mais próximas dos valores reais (saídas desejadas) [4].

O Gradiente Descendente Estocástico realiza o mesmo processo que o Gradiente Descendente normal, com a modificação que avalia cada exemplo de treinamento individual-

mente, atualizando os pesos não mais apenas depois de ter avaliado o custo para todos os exemplos de treinamento, essa abordagem tem a vantagem de que não se demora tanto para atualizar os pesos da RNA, porém a convergência é mais ruidosa e de modo geral nunca se atinge o ponto de mínimo, mas se fica oscilando próximo dele, já que a cada exemplo os pesos são atualizados, também se perde as vantagens relacionadas a paralelização de processamentos. [25, 22]

Na prática a abordagem mais eficaz para a maioria dos problemas demonstra ser utilizar um meio termo entre o Gradiente Estocástico e o Gradiente Descendente, o chamado Gradiente Mini-Batch, pois no primeiro caso perdemos as vantagens de vetorização (paralelização dos cálculos) do processamento e no segundo a atualização dos pesos é muito demorada, portanto definimos uma quantidade entre um e o número total de exemplos para treinamento [25, 41, 22].

Adam (*Adaptive Moment Estimation*)

É uma versão melhorada do Gradiente Descendente e um otimizador muito popular como cita Goodfellow et al. (2016), pois é um dos poucos que empiricamente foi mostrado se adaptar bem a uma ampla variedade de problemas no campo das RNAs. Adam foi apresentado por Diederik Kingma da OpenAI e Jimmi Ba da Universidade de Toronto em seu artigo de 2015 [32]. Como cita Andrew Ng em [41] de modo geral na comunidade de *Deep Learning* e Machile Learning diversos algoritmos para otimização foram propostos e funcionam muito bem para problemas específicos, porém nenhum deles mostrou obter sucesso em uma variedade de problemas tão ampla quanto o Adam, de tal modo ele é utilizado como a escolha padrão para o início da maioria dos projetos envolvendo RNAs.

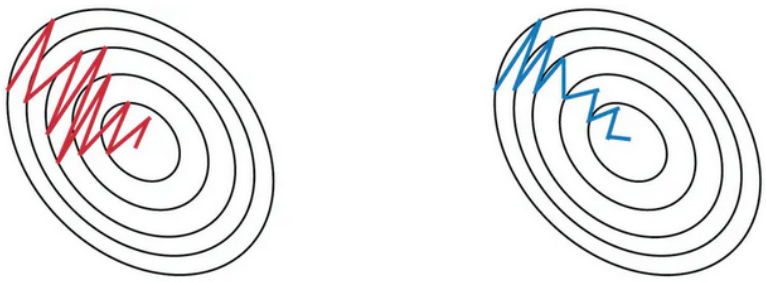


Figura 2.7: Encontrando o ponto de mínimo (centro da elipse), em vermelho Gradiente Descendente sem momento, em azul com momento. Fonte: https://www.hackster.io

O otimizador Adam aplica duas evoluções ao algoritmo do Gradiente Descendente: O momento [43]: cuja intuição é que indica a direção de "caída" dos pesos da RNA de modo

que a cada iteração os pesos são atualizados sem que percamos muito dessa direção o que evita que a otimização divirja e convirja mais rapidamente pois reduz a oscilações dos pesos tendo assim uma convergência mais direta para o ponto de mínimo, como podemos observar na Figura 2.7, para tanto realizamos o cálculo da média móvel exponencial dos pesos da RNA, no artigo original os valores $\beta_1 = 0.9$ e $\beta_2 = 0.999$ são recomendados, ou seja estamos utilizando a média de 10 gradientes anteriores como é mostrado em (2.5); A atualização dos pesos é feita conforme segue em (2.6):

$$\frac{\partial W_{medio}^{t+1}}{\partial J(W)} = \beta_1 \frac{\partial W_{medio}^{t}}{\partial J(W)} + (1 - \beta_1) \frac{\partial W^{t}}{\partial J(W)} \tag{2.5}$$

$$W^{t+1} = W^{t+1} + \alpha \frac{\partial W_{medio}^{t+1}}{\partial J(W)} \tag{2.6}$$

A outra modificação é a aplicação do algoritmo RMSProp [49], que escala e adapta a taxa de aprendizado para cada peso da rede, essa escala é proporcional a raiz quadrada da media móvel exponencial utilizando o quadrado do gradiente, também pode ser chamado de "segundo momento", ela permite que o algoritmo reduza ou aumente a taxa de aprendizado quando necessário, então quando encontramos a "direção"para o ponto de mínimo, o algoritmo pode fazer por exemplo atualizações de pesos mais bruscas, resultado de ter aumentado a taxa de aprendizado [25, 32]. O algoritmo para atualização dos pesos fica então:

$$\frac{\partial W_{seg}^{t+1}}{\partial J(W)} = \beta_2 \frac{\partial W_{seg}^{t}}{\partial J(W)} + (1 - \beta_2) \left(\frac{\partial W^{t}}{\partial J(W)}^2 \right) \tag{2.7}$$

$$W^{t+1} = W^{t+1} + \alpha \left(\frac{\partial W_{medio}^{t+1}}{\partial J(W)} \right) / \left(\sqrt{\frac{\partial W_{seg}^{t+1}}{\partial J(W)}} + \epsilon \right) \tag{2.8}$$

Onde ϵ por padrão é 10^{-8} e é empregado para prover maior estabilidade ao algorítimo, dado que com isso sempre evitaremos divisão por zero. A equação (2.8) nos apresenta então uma versão generalizada do *Backpropagation*, e que será usada no presente trabalho.

2.2 Teorema da Aproximação Universal

O Teorema da Aproximação Universal proposto por Cybenko (1988) declara que uma RNA com apenas uma camada oculta, e com um neurônio na camada de saída com ativação linear, pode representar qualquer função contínua e para qualquer ϵ desejado em que $|N(\theta, x) - f(x)| < \epsilon$, ou seja, para qualquer grau de erro desejado.

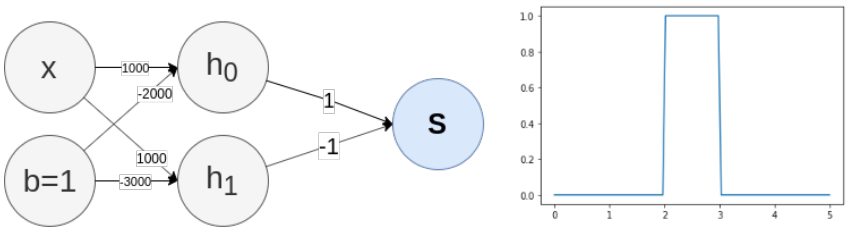


Figura 2.8: Composição de sigmoids para gerar um retângulo de saída

.

Utilizando como base para demonstrar graficamente a função de ativação sigmoid para as camadas ocultas e linear na camada de saída, podemos gerar valores para pesos em uma RNA com apenas uma camada oculta de modo que a saída de duas sigmoids gere uma função do tipo sinal, como mostra a figura [19].

Tomando como exemplo a aproximação da função $y = sen(x)$ em $[0, 2\pi]$, seguimos uma ideia semelhante a indução vista nas disciplinas de Cálculo, onde o valor da integral é a soma das áreas de retângulos em baixo de uma curva, quando a largura dos retângulos individuais tende a zero, encontramos o valor da integral. Para o caso da aproximação das funções realizamos a composição de funções, com a característica de gerar uma saída do tipo sinal, compondo duas a duas as funções sigmoid, geramos um retângulo como saída (Figura 2.8) e que pode ser deslocado no eixo simplesmente alterando o valor do bias (w/b indica em cada saída de h_i onde será o ponto em que a o valor passa de zero para um), cuja altura aproxima o valor da função no intervalo referente a largura do mesmo, quando essa largura tente a zero, ou seja o número de composições tende ao infinito temos a própria função objetivo.

De tal forma podemos concluir que uma RNA com apenas uma camada oculta pode aproximar uma função qualquer para uma precisão dada, bastando que se tenha neurônios suficientes na camada oculta para que a RNA tenha capacidade de representar a função, por essa ideia uma RNA teórica com infinitos neurônios na camada oculta poderia representar qualquer função. A mesma ideia pode ser aplicada quando passamos para mais dimensões. [19, 29].

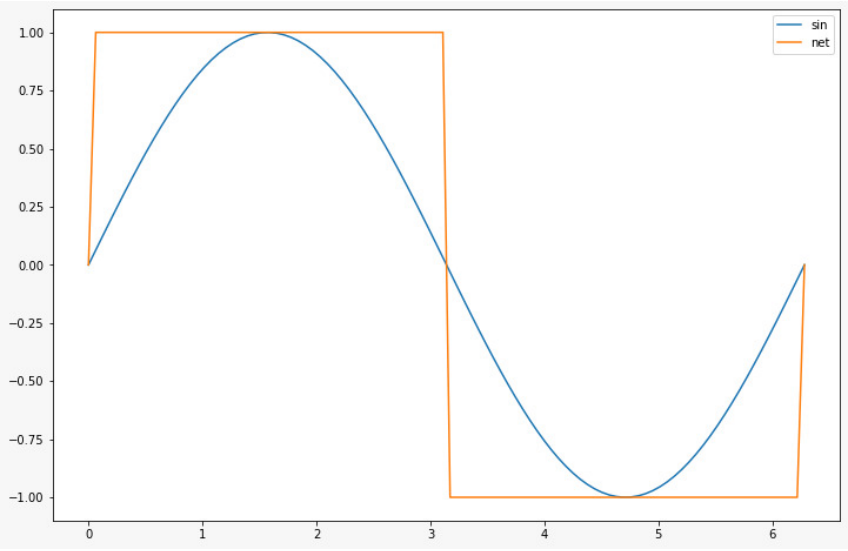


Figura 2.9: Composição de 4 sigmoids para aproximar $sen(x)$ em $[0, 2\pi]$

.

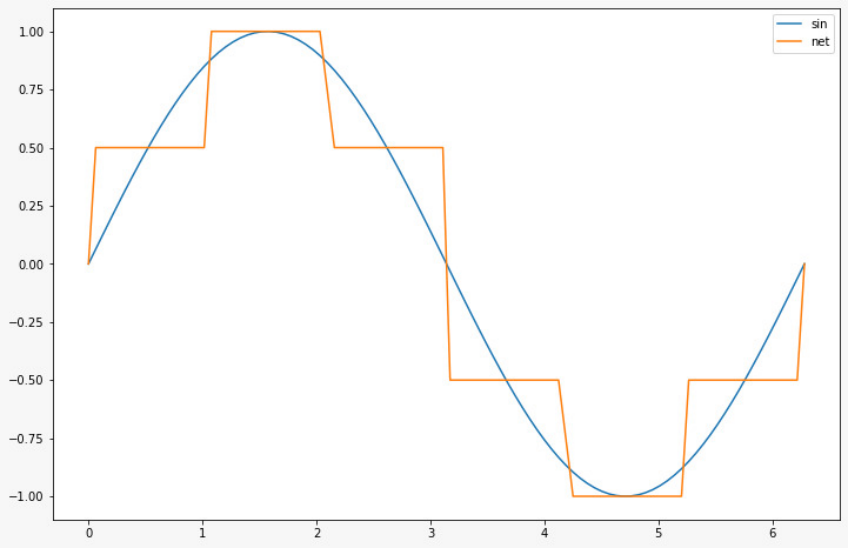


Figura 2.10: Composição de 12 sigmoids para aproximar $sen(x)$ em $[0, 2\pi]$

.

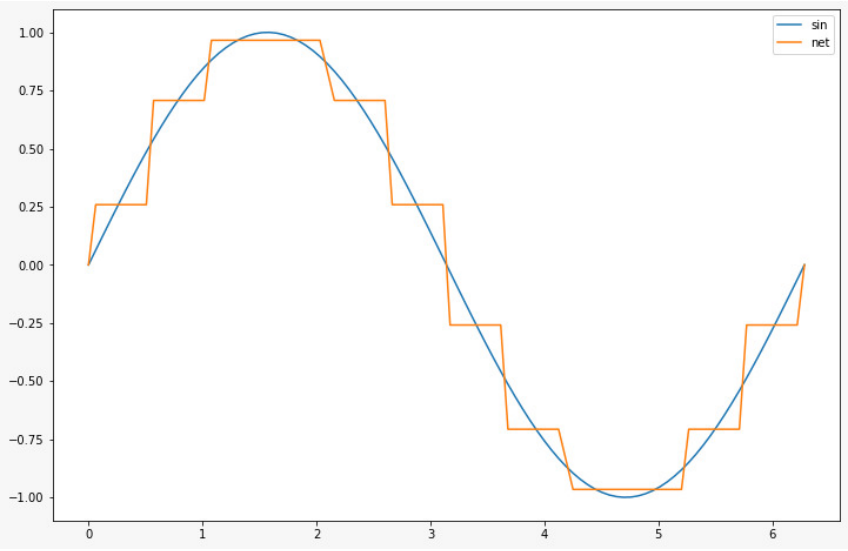


Figura 2.11: Composição de 12 sigmoids para aproximar $sen(x)$ em $[0, 2\pi]$

.

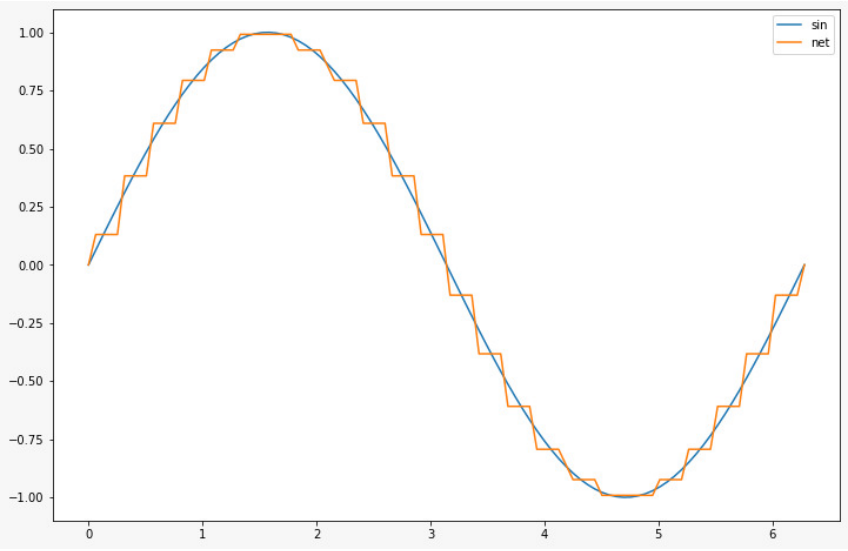


Figura 2.12: Composição de 24 sigmoids para aproximar $sen(x)$ em $[0, 2\pi]$

.

Em Figura 2.9, Figura 2.10, Figura 2.11 e Figura 2.12 utilizamos a arquitetura de RNA mostrada em Figura 2.8, porém alterando o número de neurônios na camada oculta, cada composição de sigmoid representa um neurônio adicionado a camada oculta.

Na prática buscamos encontrar o menor número de neurônios que forneça capacidade a RNA de aproximar a função dada obedecendo as restrições de precisão, porém esse processo como tanto outros no campo das RNAs é empírico [25], outro fator a se destacar é que não se faz necessário usar de pesos e bias com valores absolutos tão elevados a ponto de gerar uma função sinal, em RNA geral se indica que estes sejam próximos a zero [33, 25, 41], pois a maioria das funções apresentam regiões mais suaves que podem ser

aproximadas pela composição de funções sigmoids usando de pesos menores e que cobrem uma maior região do domínio, o que muitas vezes permite um número mais reduzido de neurônios. O processo de aproximar uma função utilizando uma RNA é então a tarefa de encontrar os pesos para realizar a composição adequada de funções de ativação de modo a encontrar a função desejada.

Hornik (1989) clarificou as ideias presentes em Cybenko (1988), mostrando que a capacidade de aproximação da RNA vem da característica de feedforward da mesma e não da função de ativação, o que é natural de ser observado se seguirmos as ideias mostradas nas figuras, tendo neurônios suficientes na camada oculta, nos basta utilizar os pesos adequados nas conexões com a camada de saída, para que possamos formar a altura, ou seja, o valor de y da função em dado ponto, e então realizar a composição dessas funções [29].

O Teorema da Aproximação Universal é a propriedade que garante a possibilidade da aplicação de RNAs para resolver ED's, dado que a solução que queremos aproximar é uma função, logo a RNA, desde que com os parâmetros definidos corretamente, é capaz de aprender o comportamento da solução do problema, de um modo geral temos informações do problema que nos ajudam a saber o quanto estamos "errando", como por exemplo um resíduo [33] ou dados experimentais [45].

2.3 Diferenciação Automática

Computacionalmente temos três grandes abordagens para gerar derivadas de uma função, são elas a Diferenciação Simbólica, Numérica e a Automática. A Diferenciação Simbólica visa derivar exatamente a função informada, por meio da aplicação das regras do Cálculo, como a regra da cadeia ou a regra do tombo, ou seja, realiza uma manipulação de expressões matemáticas para gerar as derivadas, porém em quesitos práticos leva a códigos ineficientes, onde diversos cálculos são desnecessariamente refeitos e expressões muitos longas mesmo para funções simples são encontradas. A Diferenciação Numérica é o centro dos MNC para resolução de EDs e resulta em uma aproximação da derivada desejada, muitas das vezes utiliza o Método das Diferenças Finitas [12], neste caso erros de arredondamento no processo de discretização e cancelamento podem ser introduzidos, além de dificuldade de lidar com malhas computacionais, como foi será tratado nas seções seguintes (Baydin et al, 2017).

A outra abordagem é a DA (Diferenciação Automática) desenvolvida por Wengert

(1964), nela conseguimos obter derivadas exatas e ainda aproveitar cálculos realizados para não desperdiçar processamento, a DA representa um conjunto de abstrações que nos permitem escrever a função que desejamos derivar de uma forma em que fique fácil de aplicar a regra da cadeia do Cálculo. A forma com que a função é escrita é por meio de composições de operações elementares que um computador consegue realizar, estando entre elas a soma, subtração, multiplicação, divisão, senos, cossenos, logaritmos, exponenciais e entre outros. Diferentemente da Diferenciação Simbólica onde a função a ser derivada é informada sem alterações, a DA usa a função transformada em uma composição de funções elementares, que pode ter seus membros derivados exatamente, já que são operações básicas. A função é então construída usando um grafo computacional, como mostra a Figura 2.13, onde valores intermediários são salvos de modo a não precisarem ser recalculados, o que acelera consideravelmente o processamento [9, 25, 41].

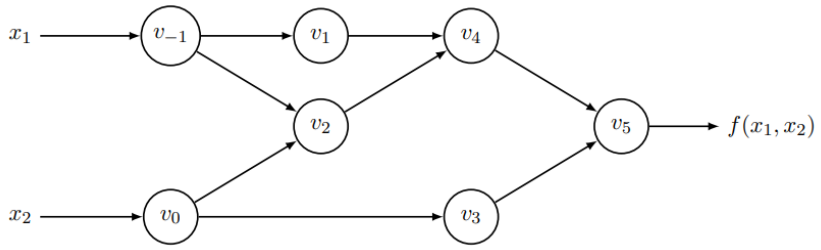


Figura 2.13: Exemplo de grafo computacional gerado para o cálculo da função processada $f(x_1, x_2) = ln(x_1) + x_1x_2 - sen(x_2)$;
Fonte: Baydin et al. (2017)

O grafo computacional mostrado na Figura 2.13 é utilizado para calcular o valor de f, os valores intermediários v_i são mantidos pelo grafo de modo a poderem serem reaproveitados quando necessário. Para calcular a derivada, por exemplo de f em relação a x_1, primeiro definimos que as derivadas de v_i são calculadas por meio de $\frac{\partial v_i}{\partial x_1}$, v_{-1} terá 1 como derivada com relação a x_1, enquanto que v_0 a derivada em relação a x_1 é zero, continuamos então pelo grafo computacional aplicando a regra da cadeia até obter o valor de saída que será a derivada da função com relação a variável desejada, essa forma de calcular é chamada de "Foward mode". (Baydin et al., 2017)

Baydin et al. (2017) evidencia que a DA é usada de modo geral por aplicações de RNAs que usam de aprendizado baseado em Gradientes, onde precisamos calcular de forma eficiente as derivadas do custo obtido com relação aos parâmetros da RNA, ou seja, seus pesos, que em RNAs robustas podem passar dos milhares, sendo inviável utilizar as outras abordagens para derivação, para o caso específico do *Backpropagation*, utilizamos o

modo contrário ao visto anteriormente, o chamado "Reverse mode", neste modo ao invés de calcularmos a derivada da função com respeito a apenas uma das variáveis, fazemos o cálculo começando pela saída e então voltamos pelo grafo computacional realizando o cálculo de todas as derivadas intermediárias até chegar nas variáveis da função, podemos notar que este modo é mais complexo que o anterior, porém no *Backpropagation* precisamos de todas as derivadas em relação a cada peso, então este é o mais indicado para a tarefa. Quando o assunto é RNAs na solução de EDs a DA é ainda mais útil, pois necessitamos das derivadas da função aproximada com relação as variáveis de entrada, ou seja, usamos a DA para calcular as derivadas da RNA usando também o Forward mode.

Para as implementações de RNAs feitas neste texto, usaremos da biblioteca "Autograd"[21] do Python [16], ela provê procedimentos para realizar a DA, os quais serão fundamentais para a resolução dos problemas a serem trabalhados.

2.4 Equações Diferenciais

Uma Equação é dita diferencial quando apresenta derivadas de uma função, que depende de uma ou mais variáveis independentes. Quando uma ED depende de apenas uma variável independente, a mesma é dita ser do tipo Ordinária (EDO) e as derivadas que estão presentes são derivadas simples, quando possui duas ou mais variáveis independentes é chamada Parcial (EDP), pois aparecem derivadas parciais em relação as variáveis independentes. (Boyce, 2017)

Como é explicitado em Boyce (2017), uma ED nem sempre tem solução, porém existem teoremas que garantem a existência sobre certas condições, porém em termos práticos quando estamos modelando um fenômeno físico por exemplo, podemos esperar que a Equação tenha solução, se não possuir devemos assumir algum erro ocorreu na formulação.

2.4.1 Equações Diferenciais Ordinárias (EDO)

Uma EDO apresenta o seguinte formato:

$$F(x, y(x), y'(x), y''(x), ..., y^{(n)}(x)) = 0 \tag{2.9}$$

Sendo F um operador diferencial qualquer. Onde $y = y(x)$, assumimos que y é a variável dependente e que x é a variável independente, logo $y^{(k)}$ representa a derivada simples de ordem k da função y em relação a x.

Ordem e Grau de uma EDO

A ordem de uma equação diferencial é a ordem da mais alta derivada da função dependente que ocorre na equação. Grau é o valor do expoente para a derivada mais alta da equação. Por convenção geralmente o índice numérico só é utilizado para derivadas de ordem superior a 3, enquanto que as ordens um, dois e três são respectivamente denotadas por: y', y'' e y''' [26, 1].

Exemplos:

1. $y' = f(x, y)$ tem ordem 1 e grau 1
2. $(y'' + y')^4 - 3y + 6 = x^2$ tem ordem 2 e grau 4
3. $5y + y''' + (y'')^8 + cos(x) + senx(x) = 0$ tem ordem 3 e grau 1

4. $21 + arctan(x)(y')^2 + 2y = x$ tem ordem 1 e grau 2

5. $(y^{(4)})^3 y'' y' y + y'' + y = e^x$ tem ordem 4 e grau 3

2.4.2 Solução de uma EDO

A solução para uma EDO é a função $y(x)$ cujas derivadas satisfaçam a equação informada. O processo de obter a solução objetiva eleminar as derivadas e encontrar a função $y(x)$ em termos unicamente da variável independente, mesmo que implicitamente (Krantz, 2005). Normalmente, caso exista, a solução encontrada não é única. e de modo geral encontramos uma família de soluções, podemos apresentar a solução de uma EDO de duas formas: a Solução Geral que é o conjunto de todas as soluções da EDO e a Solução Particular que é um dos elementos do conjunto de Solução geral, obtido pela submissão da EDO a certas condições [12, 8].

Solução Geral: que é uma expressão que depende de um ou mais parâmetros e engloba todas as soluções da equação. Representam uma família de curvas integrais ou primitivas.

Solução Particular: Como cita Boyce (2017) frequentemente queremos nos focar em uma solução específica presente na família de todas as soluções da EDO, para isso definimos um valor para determinada constante da equação ou ainda impomos condições a equação de tal forma que obtenhamos uma solução única. Quando apenas um ponto é usado e permite que encontremos uma solução particular, chamamos de Problema de Valor Inicial (PVI), quando precisamos de mais de um ponto para encontrar uma solução particular chamamos então de Problema de Valor de Fronteira (PVF).

2.4.3 Problemas de Valor Inicial (PVI)

Como foi elucidado anteriormente, uma EDO geralmente não possui uma solução única, então para que encontremos uma solução particular a mesma, adicionamentos uma condição inicial, como por exemplo:

$$\begin{cases} y' = f(x, y) \\ y(x_0) = y_0 \end{cases} \tag{2.10}$$

onde x_0 representa o tempo/posição inicial e y_0 o valor inicial. Unindo a equação e a condição inicial, temos o chamado **Problema de Valor Inicial(PVI)** .

Métodos Numéricos para Solução de PVIs

Uma equação diferencial não tem uma uma solução única, então tipicamente é adicionada uma condição inicial a equação, a mesma ficando na seguinte forma

$$\begin{cases} y' = f(x, y) \\ y(x_0) = y_0 \end{cases}$$

onde x_0 representa o tempo/posição inicial e y_0 o valor inicial, ou seja o valor de y avaliado no ponto inicial dado. Unindo a EDO e a condição inicial, temos o chamado **Problema de Valor Inicial (PVI)** .

Método de Euler

Como é dito em Ascher (2011) o Método de Euler é o mais simples dos métodos numéricos utilizados para aproximar a solução de um PVI. O mesmo foi criada no século XVIII pelo matemático Leonhard Euler

Considerando um PVI:

$$\begin{cases} y' = f(x, y) \\ y(x_0) = y_0 \end{cases}$$

O método de Euler é baseado na expansão da função $y(x)$ em series de Taylor, assim, pode-se expandir a função $y(x)$ na vizinhança do ponto x_n, até a ordem 1, podemos então escrever [5, 12]:

$$y(x_n + h) \cong y(x_n) + h.y'(x_n)$$

como $(x_n + h) = x_{n+1}$ e $y'(x_n) = f(x_n, y(x_n))$ temos:

$$y(x_{n+1}) = y(x_n) + h.f(x_n, y(x_n))$$

tomando $n = 0$ obtemos:

$$y(x_1) = y(x_0) + h.f(x_0, y(x_0))$$

Portanto o método de Euler aproxima o valor y_1 do valor exato $y(x_1)$ no ponto x_1.

Ao continuar repetindo esse procedimento para os pontos $x_2,\ldots,x_n$, onde $n = 1, 2\ldots$, com N de um intervalo $I = [a, b]$ temos as aproximações $y_2,\ldots,y_n$ dos valores exatos $y(x_2),\ldots,y(x_n)$, obtendo de modo geral a fórmula:

$$y_{n+1} = y_n + h.f(x_n, y_n)$$

Métodos de Runge-Kutta

O Método de Euler é pode ser ineficiente para muitas aplicações, dado que é um método de primeira ordem, ele depende fortemente da escolha de um h muito pequeno para obter acurácia comparável a métodos mais robustos, como é o caso dos Métodos de Runge-Kutta. Os métodos de Runge-Kutta são métodos de passo simples, sendo que y_{i+1} é determinado usando unicamente x_i e y_i.[52, 5]

Um método de Runge-Kutta de ordem p não requer o cálculo de qualquer derivada de f, como o que apresentado posteriormente no Método das Diferenças Finitas, porém requer que façamos a construção de uma função Φ que é definida avaliando f em diferentes pontos.[52]

Os métodos de Runge-Kutta são definidos da seguinte forma:

$$y_{i+1} = y_i + h\Phi(x_i, y_i), \quad i = 0, 1, \ldots, N$$

onde Φ dependende indiretamente de f, já que Φ depende de x e y, mas para realizar o cálculo de y necessitamos de f. h é o passo a ser utilizado a cada vez que se avança na direção x.

O método de Euler, obtido considerando $\Phi = f$, é um método de Runge-Kutta de ordem $p = 1$ (Valle, 2012).

Método de Runge-Kutta de ordem 2

Em VALLE (2012) tomamos conhecimento que o método de Heun é uma versão melhorada do método de Euler, e é um método de Runge-Kutta de ordem 2. Nele definimos: $y_{i+1} = y_i + \frac{h}{2}(k_1 + k_2), \quad i = 0, 1, \ldots, N$ em que $k_1 = f(x_i, y_i)$ e $k_2 = f(x_i + h, y_i + hk_1)$ ou de forma equivalente temos

$$y_{i+1} = y_i + \frac{h}{2}(f(x_i, y_i) + f(x_i + h, y_i + hf(x_i, y_i))), \quad i = 0, 1, \ldots, N$$

Se analisarmos em termos da construção de um Método Runge-Kutta, podems notar que neste caso

$$\Phi(x,y) = \frac{1}{2}(f(x,y) + f(x + h.y + hf(x,y)))$$

Método de Runge-Kutta de ordem 4

De forma análoga, podemos construir o método de Runge-Kutta de ordem 4, sendo ele dado por:

$$y_{i+1} = y_i + \frac{h}{6}(k1 + 2k_2 + 2k_3 + k_4)$$

onde

$$\begin{cases} k_1 = f(x_i, y_i) \\ k_2 = f(x_i + \frac{h}{2}, y_i + k_1\frac{h}{2}) \\ k_3 = f(x_i + \frac{h}{2}, y_i + k_2\frac{h}{2}) \\ k_1 = f(x_i + h, y_i + hk_3) \end{cases}$$

Essa forma do método de Runge-Kutta, segundo Ascher (2011) é a mais clássica a ser utilizada, pois é simples de ser implementada e mesmo assim fornece uma boa acurácia sem a necessidade de utilizar um h demasiadamente pequeno.

2.4.4 Problema de Valor de Fronteira (PVF)

Agora consideremos um problema com dois pontos de fronteira e uma equação diferencial de segunda ordem, com condições de fronteira

$$\begin{cases} y'' = f(x, y, y'), \quad x \in [a, b] \\ y(a) = \alpha, \quad y(b) = \beta \end{cases} \tag{2.11}$$

Uma solução única é garantida quando as três seguintes condições são garantidas: f, f_y e $f_{y'}$ são contínuas no domínio:

$$D = \{(x, y, y') | a \leq x \leq b, -\infty < y < \infty, -\infty < y' < \infty\}$$

$f_n > 0$ em D e $f_{y'}$ é limitada em D. Como dito em [8] quando temos EDOs de ordens superiores, de modo geral, não é possível determinar suas soluções analíticas, porém elas existem e podemos utilizar de métodos numéricos para encontrar.

Métodos das Diferenças Finitas

Para solucionar o PVF em (2.11), a primeira e segunda derivadas da solução $y(x)$ são aproximadas pelas diferenças finitas (a aproximação de derivadas pelas diferenças finitas pode ser encontrada na Seção 3.2 de [24]), ou seja, a EDO é discretizada [12].

O intervalo $[a, b]$ é discretizado em $N + 1$ subintervalos de mesmo tamanho, onde $h = \frac{(b-a)}{N+1}$, onde cada $x_i = a + ih$. Denotamos por y_i uma aproximação para a solução em x_i, ou seja, $y_i \approx y(x_i)$ (Ruggiero, 1997). Assumindo que $y(x)$ é diferenciável até a quarta ordem, aproximamos y' e y'' em cada $x_i, i = 1, 2, ..., N$, pelas Diferenças Finitas:

$$y'(x_i) = \frac{y(x_{i+1}) - y(x_i)}{2h} - \frac{h^2}{6}y'''(\eta_i)$$
$$y''(x_i) = \frac{y(x_{i+1}) - 2y(x_i) + y(x_{i-1})}{h^2} - \frac{h^2}{12}y^{(4)}(\xi_i)$$

onde η_i e ξ_i estão em $[x_{i-1}, x_{i+1}]$. Substituindo essas diferenças finitas no PVF e aplicando as condições de contorno, temos

$$y_0 = \alpha, \quad y_N = \beta$$

obtemos então um sistema de equações

$$\frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} = f(x_i, y_i, \frac{y_{i+1} - y_{i-1}}{2h}), \quad i = 1, 2, ..., N$$

para esses valores da solução em cada x_i, nos quais o erro de truncamento local é da ordem de $O(h^2)$. [5, 24]

Problemas Lineares

Considerendando EDOs lineares do tipo

$$y'' = p(x)y' + q(x)y + r(x)$$

O sistema de equações abaixo também é linear, e podemos expressar da forma matriz-vetor

$$Ay = r,$$

Onde A é uma matriz tridiagonal como é dito em Ruggiero (1997), dado que as aproximações de y' e y'' em x_i usam apenas y_{i-1}, y_i e y_{i+1}, e r é um vetor que inclui os valores de $r(x)$ na malha, assim como termos adicionais para as condições de contorno:

$$a_{ii} = 2 + h^2 q(x_i), i = 1, 2, ..., N,$$

$$a_{i,i+1} = -1 + \frac{h}{2} p(x_i), i = 1, 2, ..., N-1,$$

$$a_{i+1,i} = -1 - \frac{h}{2} p(x_{i+1}), i = 1, 2, ..., N-1,$$

$$r_1 = -h^2 r(x_1) + 1 + \frac{h}{2} p(x_1)\alpha,$$

$$r_i = -h^2 r(x_i), i = 2, 3, ..., N-1$$

$$r_N = -h^2 r(x_N) + 1 - \frac{h}{2} p(x_N)\beta,$$

O sistema de equações é garantido ter uma única solução se A for diagonalmente dominante, que é o caso quando $q(x) \geq 0$ e $h < 2/L$, onde L é o limite superior de $|p(x)|$. (ARCHER, 2011)

Problemas Não-Lineares

No caso da EDO ser não-linear, devemos resolver um sistema de equações não lineares da forma

$$F(y) = 0$$

onde $F(y)$ é uma função vetorial com funções coordenadas $f_i(y)$, para $i = 1, 2, ..., N$, que são definidas da seguinte forma:

$$F_1(y) = y_2 - 2y_1 + \alpha - h^2 f(x_1, y_1, \frac{y_2 - \alpha}{2h}),$$

$$F_2(y) = y_3 - 2y_2 + y_1 - h^2 f(x_2, y_2, \frac{y_3 - y_1}{2h}),$$

$$\vdots$$

$$F_{N-1}(y) = y_N - 2y_{N-1} + y_{N-2} - h^2 f(x_{N-1}, y_{N-1}, \frac{y_N - y_{N-2}}{2h}),$$

$$F_N(y) = \beta - 2y_N + y_{N-1} - h^2 f(x_N, y_N, \frac{\beta - y_{N-1}}{2h}).$$

Esse sistema de equações pode solucionado aproximadamente por um método iterativo, como o Método de Newton ou Método da Secante [5]. Se o Método de Newton for utilizado, então pela regra da cadeia, as entradas da matriz Jacobiana $J_F(y)$, uma matriz tridiagonal, será definida como segue:

$$J_F(y)_{ii} = \frac{\partial f_i}{\partial y_i} = -2 - h^2 f_y(x_i, y_i, \frac{y_{i+1} - y_{i-1}}{2h}), \quad i = 1, 2, .., N$$

$$J_F(y)_{i,i+1} = \frac{\partial f_i}{\partial y_{i+1}} = 1 - \frac{h}{2} f_{y'}(x_i, y_i, \frac{y_{i+1} - y_{i-1}}{2h}), \quad i = 1, 2, .., N-1$$

$$J_F(y)_{i,i-1} = \frac{\partial f_i}{\partial y_{i-1}} = 1 + \frac{h}{2} f_{y'}(x_i, y_i, \frac{y_{i+1} - y_{i-1}}{2h}), \quad i = 2, 3, .., N$$

onde, para conveniência, usamos $y_0 = \alpha$ e $y_{N+1} = \beta$. Então em cada iteração do método de Newton, o sistema de equações

$$J_F(y^{(k)})S_{k+1} = -F(y^{(k)})$$

é resolvido de modo a obter a próxima iteração

$$y^{(k+1)} = y^{(k)} + S_{k+1}$$

da iteração anterior. Um "chute"inicial é a função linear única que satisfaz as condições de contorno:

$$y^{(0)} = \alpha + \frac{\beta - \alpha}{b - a}(x - a),$$

onde x é o vetor de coordenadas $x_1, x_2, ..., x_N$ (Ascher, 2011).

2.4.5 Equações Diferenciais Parciais

Uma equação diferencial parcial (EDP) é uma equação que representa a relação entre duas ou mais variáveis independentes, como cita Fortana (2019) as variáveis independentes mais encontradas são as referentes as dimensões espaciais do problema ou ao tempo (que caracterizam problemas de evolução). De maneira mais precisa podemos definir uma EDP em n variáveis independentes é uma equação que apresenta o seguinte formato:

$$F(x_1, \ldots, x_n, u, \frac{\partial u}{\partial x_1}, \frac{\partial^2 u}{\partial x_1^2}, \ldots, \frac{\partial^2 u}{\partial x_1 x_n^2}, \ldots, \frac{\partial^n u}{\partial x^n}) = 0 \tag{2.12}$$

onde $(x_1, \ldots, x_n) \in \Omega$, sendo Ω um subconjunto aberto de R^n, F é uma função dada e $u(x)$ é a função que se quer determinar [24].

Ordem e grau de uma EDP

A ordem de uma EDP é igual ao caso para EDO, ou seja, a derivada de maior ordem presente. Focaremos nas EDPs de segunda ordem, que segundo Fortana (2019) são classificadas em: elípticas, parabólicas, e hiperbólicas.

Exemplos de Equações Diferenciais Parciais

1. Equação do Calor: $\frac{\partial u}{\partial t} - \alpha \frac{\partial^2 u}{\partial x^2} = \frac{1}{2} e^{-0.1t}(x^2 - 2x + 20\alpha^2)$
2. Equação do Calor: $\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2} + 2.5$
3. Equação do Calor: $\frac{\partial u}{\partial t} = \alpha^2 (\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2})$
4. Equação da Onda: $\frac{\partial^2 u}{\partial t^2} = 4\frac{\partial^2 u}{\partial x^2} + 4$
5. Equação da Onda: $\frac{\partial^2 u}{\partial t^2} = \alpha^2 (\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2})$
6. Equação de Laplace : $\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$
7. $\cos(xy)\frac{\partial u}{\partial x} = 4x + sen(x)y$
8. $\arctan(y)\frac{\partial^4 u}{\partial x^4} + y\frac{\partial^3 u}{\partial x^3} + 2x\frac{\partial u}{\partial x}\frac{\partial u}{\partial y} = xy$

No exemplo anterior, as equações dos itens 1-6 são de segunda ordem, a do ítem 7 é de primeira ordem e a do item 7 é de quarta ordem.

Equações Parabólicas

Representam problemas de evolução (não-estacionários) e não propagam descontinuidades, isto é, a solução de uma equação parabólica é sempre suave dentro do domínio de solução, mesmo quando a condição inicial não é. (Leveque, 2007)

Em 2D, temos que x representa a variável espacial e y a variável temporal.

É comum encontrar equações parabólicas do tipo:

$$\begin{cases} \frac{\partial u}{\partial t} = s(x,t)\frac{\partial^2 u}{\partial x^2} + r(x,t), \quad s(x,t) > 0, a < x < b, 0 < t < T \\ u(x,0) = \varphi(x), a \leq x \leq b \\ u(a,t) = f(t), 0 < t < T \\ u(b,t) = g(t), 0 < t < T \end{cases} \tag{2.13}$$

A seguir temos a discretização do domínio para resolver numéricamente a equação (2.13):

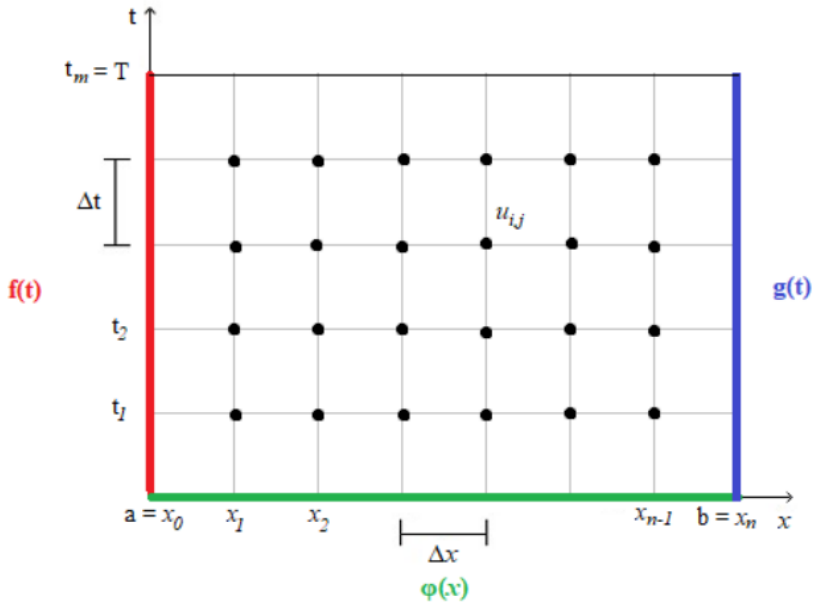


Figura 2.14: Discretização do domínio para equação geral.
Fonte: Google Imagens

Se $s(x,t)$ é contínua e limitada e $r(x,t,u,u_x)$ é monotonicamente decrescente em u então o problema (2.13) tem solução única.

Considerando um caso particular de (2.13), onde $s(x,t) = cte$ e $r(x,t) = 0$, temos então a equação do calor como se segue:

$$\begin{cases} \frac{\partial u}{\partial t} = \alpha\frac{\partial^2 u}{\partial x^2}, \quad \alpha > 0, 0 < x < L, 0 < t < T \\ u(x,0) = \varphi(x), 0 \leq x \leq L \\ u(0,t) = f(t), 0 < t < T \\ u(L,t) = g(t), 0 < t < T \end{cases} \tag{2.14}$$

No caso das equações parabólicas a solução num ponto interior depende de toda condição inicial.

Para podermos construír métodos Numéricos para resolver a equação (2.14), devemos definir uma discretização como se segue:

$$h = \frac{L}{N}, \quad k = \frac{T}{M}, \quad x_i = ih, \quad t_j = jk$$

Onde N e M são o número de pontos para as discretizações da variável espacial e temporal, respectivamente.

Método de Euler Explícito

Aproximando as derivadas parciais em (2.14) é feita utilizando das diferenças finitas. A derivada temporal é aproximada por diferença avançada no tempo j:

$$\frac{\partial u}{\partial t} = \frac{u_i^{j+1} - u_i^j}{k} - \frac{k}{2}\frac{\partial^2 u}{\partial t^2}(x_i, \xi_i) \tag{2.15}$$

A derivada espacial é aproximada por diferença centrada de segunda ordem no tempo j:

$$\frac{\partial^2 u}{\partial t^2} = \frac{u_{i-1}^j - 2u_i^j + u_{i+1}^j}{h^2} - \frac{h^2}{12}\frac{\partial^4 u}{\partial t^4}(t_j, \xi_j) \tag{2.16}$$

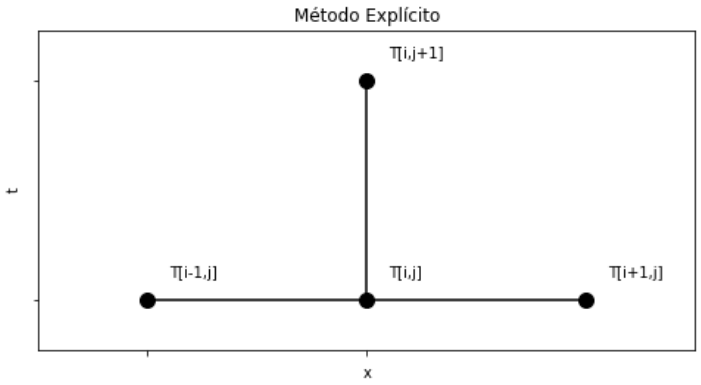


Figura 2.15: Pontos usados na discretização M. Euler Explícito.

Desprezando o erro de truncamento em (2.15) e (2.16) e substituindo em (2.14), temos:

$$\frac{u_i^{j+1} - u_i^j}{k} = \alpha \left[\frac{u_{i-1}^j - 2u_i^j + u_{i+1}^j}{h^2} \right]$$

Fazendo $\sigma = \frac{\alpha k}{h^2}$, ficamos com:

$$u_i^{j+1} = u_i^j + \sigma\left(u_{i-1}^j - 2u_i^j + u_{i+1}^j\right)$$

Ficamos então com o método de **Euler Explícito** [31]:

$$u_i^{j+1} = \sigma u_{i-1}^j + (1-2\sigma)u_i^j + \sigma u_{i+1}^j \tag{2.17}$$

onde $\sigma = \frac{\alpha k}{h^2}$, $i = 1, 2, \ldots, N-1$, $j = 0, 1, \ldots, M-1$

Método de Euler Implícito

Fazendo a derivada temporal em (2.14) ser aproximada por diferença atrasada no tempo $j+1$:

$$\frac{\partial u}{\partial t} = \frac{u_i^{j+1} - u_i^j}{k} - \frac{k}{2}\frac{\partial^2 u}{\partial t^2}(x_i, \xi_i) \tag{2.18}$$

A derivada espacial é aproximada por diferença centrada de segunda ordem no tempo $j+1$:

$$\frac{\partial^2 u}{\partial t^2} = \frac{u_{i-1}^{j+1} - 2u_i^{j+1} + u_{i+1}^{j+1}}{h^2} - \frac{h^2}{12}\frac{\partial^4 u}{\partial t^4}(t_{j+1}, \xi_{j+1}) \tag{2.19}$$

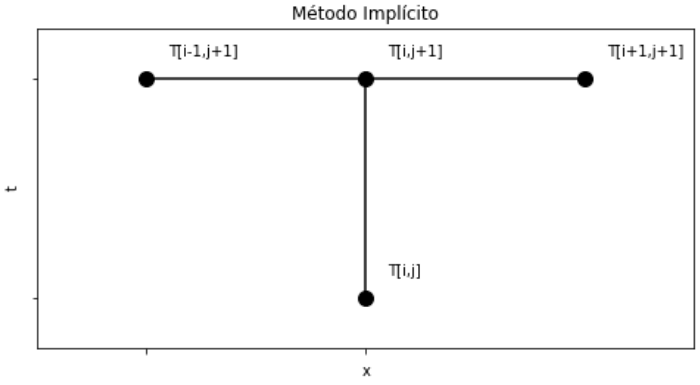


Figura 2.16: Pontos usados na discretização M. Euler Implícito.

Desprezando o erro de truncamento em (2.18) e (2.19) e substituindo em (2.14), temos:

$$\frac{u_i^{j+1} - u_i^j}{k} = \alpha\left[\frac{u_{i-1}^{j+1} - 2u_i^{j+1} + u_{i+1}^{j+1}}{h^2}\right]$$

Fazendo $\sigma = \frac{\alpha k}{h^2}$, ficamos com:

$$u_i^{j+1} = u_i^{j+1} - \sigma \left(u_{i-1}^{j+1} - 2u_i^{j+1} + u_{i+1}^{j+1} \right)$$

Ficamos então com o método de **Euler Implícito** [31]

$$u_i^{j+1} = -\sigma u_{i-1}^{j+1} + (1 + 2\sigma) u_i^{j+1} + \sigma u_{i+1}^{j+1} \tag{2.20}$$

onde $\sigma = \frac{\alpha k}{h^2}$, $i = 1, 2, \ldots, N-1$, $j = 0, 1, \ldots, M-1$

A equação (2.20) sozinha não nos permite determinar os valores de u_i^{j+1}, ao contrário da equação (2.17). Ocorre então o aparecimento de três incógnitas: $u_{i-1}^{j+1}, u_i^{j+1}, u_{i+1}^{j+1}$. Portanto, o valor de $u_i, j+1$ é definido implicitamente pela equação (2.20). Podemos escrever na forma matricial o sistema de equações na forma $Au = r$:

$$\begin{bmatrix} 1-2\sigma & -\sigma & 0 & 0 & 0 & \ldots & 0 \\ -\sigma & 1-2\sigma & -\sigma & 0 & 0 & \ldots & 0 \\ 0 & -\sigma & 1-2\sigma & -\sigma & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \vdots & 0 & 0 & -\sigma & 1-2\sigma & -\sigma \\ 0 & \vdots & 0 & 0 & 0 & -\sigma & 1-2\sigma \end{bmatrix} \begin{bmatrix} u_1^{j+1} \\ u_2^{j+1} \\ u_3^{j+1} \\ \vdots \\ u_{N-2}^{j+1} \\ u_{N-1}^{j+1} \end{bmatrix} = \begin{bmatrix} u_1^j + \sigma u_0^{j+1} \\ u_2^j \\ u_3^j \\ \vdots \\ u_{N-2}^j \\ u_{N-1}^j + \sigma u_N^{j+1} \end{bmatrix}$$

O sistema possui $(N-1)$ equações linearmente independentes e o mesmo número de incógnitas, admitindo, uma única solução. Sua resolução fornece os valores de u_i^{j+1}, em $1 \leq i \leq N-1$

Método de Crank-Nicolson

Segundo Leveque (2007) o método discretiza a equação (2.14) no ponto $(i, j+\frac{1}{2})$ como segue:

$$\left(\frac{\partial u}{\partial t} \right)_i^{j+\frac{1}{2}} = \frac{u_i^{j+1} - u_i^j}{\delta t}$$

$$\left(\frac{\partial^2 u}{\partial x^2} \right)_i^{j+\frac{1}{2}} = \frac{1}{2} \left[\left(\frac{\partial^2 u}{\partial x^2} \right)_i^j + \left(\frac{\partial^2 u}{\partial x^2} \right)_i^{j+1} \right]$$

Substituindo essas aproximações na equação (2.14) e reagrupando os termos temos:

$$\frac{u_i^{j+1} - u_i^j}{\delta t} = \frac{\alpha}{2}\left[\frac{u_{i-1}^j - 2u_i^j + u_{i+1}^j}{(\delta x)^2} + \frac{u_{i-1}^{j+1} - 2u_i^{j+1} + u_{i+1}^{j+1}}{(\delta x)^2}\right] \tag{2.21}$$

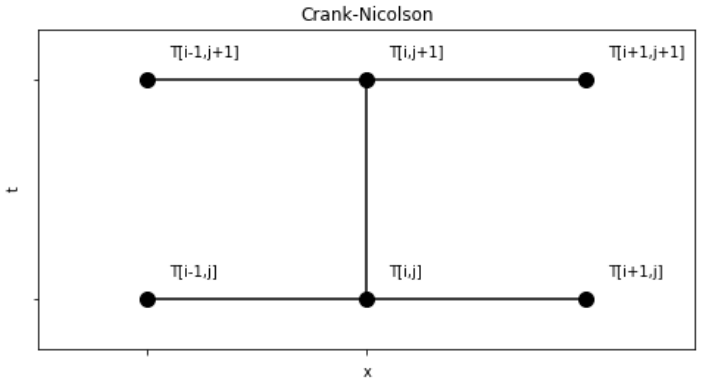


Figura 2.17: Pontos usados na discretização M. de Crank Nicolson.

Podemos então reescrever (2.21) como:

$$u_i^{j+1} = u_i^j = \frac{1}{2}\frac{\alpha\delta t}{(\delta x)^2}\left[u_{i-1}^j - 2u_i^j + u_{i+1}^j + u_{i-1}^{j+1} - 2u_i^{j+1}u_{i+1}^{j+1}\right]$$

onde $\sigma = \frac{\alpha\delta t}{(\delta x)^2}$, ficamos então com:

$$-\frac{\sigma}{2}u_{i-1}^{j+1} + (1+\sigma)u_i^{j+1} - \frac{\sigma}{2}u_{i+1}^{j+1} = \frac{\sigma}{2}u_{i-1}^j + (1-\sigma)u_i^j + \frac{\sigma}{2}u_{i+1}^j$$

Podemos então escrever a solução aproximada U_i^j como:

$$-\sigma U_{i-1}^{j+1} + (2+2\sigma)U_i^{j+1} - \sigma U_{i+1}^{j+1} = \sigma U_{i-1}^j + (2-2\sigma)U_i^j + \sigma U_{i+1}^j, \quad i = 1, 2, 3, \ldots, N-1.$$

Escrevendo na forma matricial:

$$\underbrace{\begin{bmatrix} 2+2\sigma & -\sigma & 0 & 0 & 0 & \dots & 0 \\ -\sigma & 2+2\sigma & -\sigma & 0 & 0 & \dots & 0 \\ 0 & -\sigma & 2+2\sigma & -\sigma & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \vdots & 0 & 0 & -\sigma & 2+2\sigma & -\sigma \\ 0 & \vdots & 0 & 0 & 0 & -\sigma & 2+2\sigma \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} U_1^{j+1} \\ U_2^{j+1} \\ U_3^{j+1} \\ \vdots \\ U_{N-2}^{j+1} \\ U_{N-1}^{j+1} \end{bmatrix}}_{U^{j+1}} =$$

$$\underbrace{\begin{bmatrix} 2+2\sigma & -\sigma & 0 & 0 & 0 & \dots & 0 \\ -\sigma & 2+2\sigma & -\sigma & 0 & 0 & \dots & 0 \\ 0 & -\sigma & 2+2\sigma & -\sigma & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \vdots & 0 & 0 & -\sigma & 2+2\sigma & -\sigma \\ 0 & \vdots & 0 & 0 & 0 & -\sigma & 2+2\sigma \end{bmatrix}}_{B} \underbrace{\begin{bmatrix} U_1^{j} \\ U_2^{j} \\ U_3^{j} \\ \vdots \\ U_{N-2}^{j} \\ U_{N-1}^{j} \end{bmatrix}}_{U^{j}} + \underbrace{\begin{bmatrix} \sigma U_0^{j+1} \\ 0 \\ 0 \\ \vdots \\ 0 \\ \sigma u_N^{j+1} \end{bmatrix}}_{c^j}$$

Ou utilizando da notação vetorial, podemos escrever

$$AU^{j+1} = BU^j + c^j$$

Para se obter a solução em cada estágio é preciso resolver um sistema tridiagonal. Note que, sendo A diagonalmente dominante, o problema discreto tem solução única. (Leveque, 2007)

Equações Elípticas

Representam problemas de equilíbrio que não dependem, em geral, do tempo. Exemplos: problemas de vibração de membranas, problemas de difusão, etc. Problemas de valor no contorno, em duas ou três dimensões, são geralmente descritas por equações elípticas. A discretização de uma equação estacionária fornece um sistema de equações que pode ser resolvido por meio de métodos diretos ou iterativos.

Os conceitos de métodos explícitos e implícitos no tempo não são aplicáveis às equações elípticas devido à ausência da integração temporal. Sempre é necessário resolver um sistema de equações lineares e, como a matriz de coeficientes do sistema não é, em geral, tridiagonal, a solução é obtida por métodos iterativos [36, 31]

Se R uma região do plano e ∂ R a sua fronteira. A equação

$$a(x,y)\frac{\partial^2 u}{\partial x \partial y} + 2b(x,y)\frac{\partial^2 u}{\partial x \partial y} + c(x,y)\frac{\partial^2 u}{y} = d(x,y,u,\frac{\partial u}{x},\frac{\partial u}{y}) \tag{2.22}$$

é elíptica se $b^2 - 4ac < 0, \forall (x,y) \in R$. São muitos os problemas práticos que são governados pela equação (2.12). Com $a = c = -1$ e $b = 0$. Se $d = 0$, temos a quação de Laplace (VOLKER, 2013):

$$-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} = 0 \tag{2.23}$$

Se $d = f(x,y)$, então temos a equação de Poisson:

$$-\frac{\partial^2 u}{x} - \frac{\partial^2 u}{y} = f(x,y) \tag{2.24}$$

Como (2.22) é uma equação linear, se u e v são soluções de (2.22) então $w = \alpha u + \beta v$ também será solução de (2.22), isso implica que a equação (2.22) tem infinitas soluções.

Para que (2.22) tenha solução única é preciso especificar condições de contorno de u de tal forma que a equação (2.22) tenha solução única (as condições de contorno não podem ser arbitrárias).

Em geral, a equação (2.22) é resolvida sujeita a 3 tipos de condições de contorno: Dirichlet, Neumann e Robbin (ou mista).[48]

Para resolver (2.22) vamos utilizar o método de diferenças finitas que consiste em substituir as derivadas parciais em (2.22) por aproximações por diferenças finitas.

Equações Elípticas com Contorno de Dirichlet

Para ilustrar o método, vamos considerar a equação de Poisson,

$$-\Delta u = -(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}) = f(x,y) \tag{2.25}$$

definido num retângulo $R = \{0 < x < a, 0 < y < b\}$, com condição de Dirichlet: $u(x,y) = g(x,y)$ na fronteira ∂R.

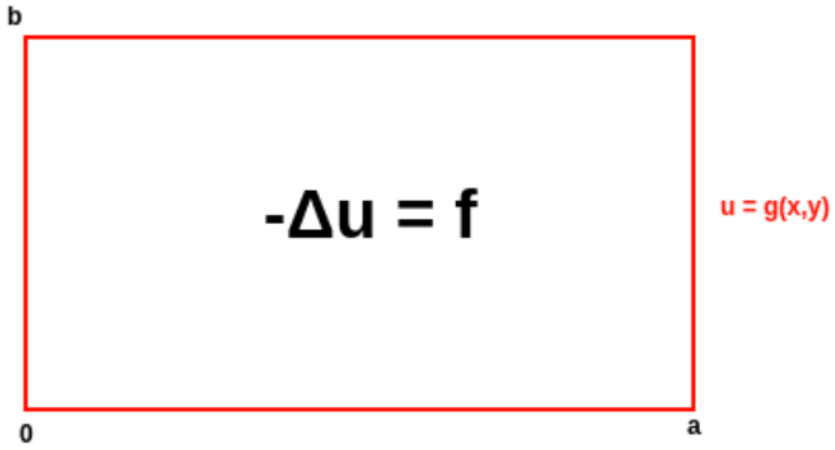


Figura 2.18: Região do problema
Fonte: Google Imagens

Definimos uma malha:

$$x_i = ih, \quad i = 0, 1, \ldots, N. \quad h = \frac{a}{N}.$$
$$y_j = jk, \quad j = 0, 1, \ldots, M. \quad k = \frac{b}{M}.$$

Considere a seguinte malha discretizada

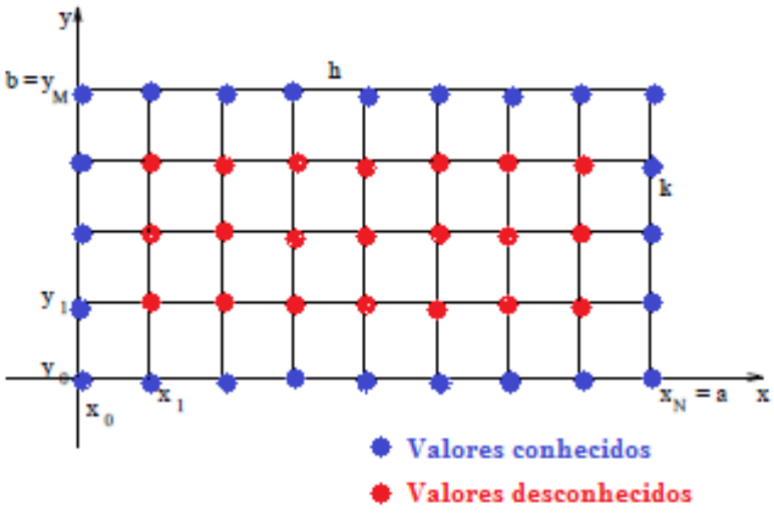


Figura 2.19: Geometria discretizada
Fonte: Google Imagens

Dessa forma, pode-se descrever os pontos no interior da malha e os pontos da fronteira da seguinte forma

$R_\delta = \{(x_i, y_j) | x_i = ih, y_j = jk, 1 \leq i \leq N-1, 1 \leq j \leq M-1\}$

$$R_\delta = \{(x_i, y_0), (x_i, y_M), \quad i = 0, 1, \ldots, N,$$
$$(x_0, y_j), (x_N, y_j), \quad j = 0, 1, \ldots, M\}.$$

A equação (2.25) é válida para qualquer ponto de R_δ , ou seja,

$$-\left(\frac{\partial^2 u}{\partial x^2}(x_i, y_j) + \frac{\partial^2 u}{\partial y^2}(x_i, y_j)\right) = f(x_i, y_j) \tag{2.26}$$

para $i = 1, 2, \ldots, N-1$ e $j = 1, 2, \ldots, M-1$. As aproximações para as derivadas por diferenças finitas são da- das por

$$\frac{\partial^2 u}{\partial x^2}(x_i, y_j) = \frac{u(x_i + h, y_j) - 2u(x_i, y_j) + u(x_i - h, y_j)}{h^2} \tag{2.27}$$

$$\frac{\partial^2 u}{\partial y^2}(x_i, y_j) = \frac{u(x_i, y_j + k) - 2u(x_i, y_j) + u(x_i, y_j - k)}{k^2} \tag{2.28}$$

Substituindo as equações (2.27) e (2.28) em (2.26), obtemos

$$-\left(\frac{u(x_i + h, y_j) - 2u(x_i, y_j) + u(x_i - h, y_j)}{h^2}\right.$$
$$\left.+\frac{u(x_i, y_j + k) - 2u(x_i, y_j) + u(x_i, y_j - k)}{k^2}\right) \approx f(x_i, y_j) \tag{2.29}$$

Observamos que a equação (2.29) não representa uma equação porque o primeiro membro é uma aproximação para o segundo membro. Para transformarmos (2.29) em uma equação trocamos o sinal de aproximado pelo sinal de igualdade e denotamos o valor de $u(x_i, y_j)$ por $u_{i,j}$ e a solução aproximada por $U_{i,j}$. Neste caso, obtemos a equação de diferenças:

$$-\left(\frac{U_{i+1,j} - 2U_{i,j} + U_{i-1,j}}{h^2} + \frac{U_{i,j+1} - 2U_{i,j} + U_{i,j-1}}{k^2}\right) = f(x_i, y_j) \tag{2.30}$$

Para pontos adjacentes à fronteira, a equação (2.30) envolverá pontos na fronteira que são obtidos diretamente da condição de fronteira de Dirichlet, ou seja, $U_{i,j} =$

$g(x_i, y_j), \forall (x_i, y_j) \in \partial R_\delta$. (Leveque, 2007)

Para simplificar a notação, vamos introduzir o operador

$$-\Delta_\delta U_{i,j} = -\left(\frac{U_{i+1,j} - 2U_{i,j} + U_{i-1,j}}{h^2} + \frac{U_{i,j+1} - 2U_{i,j} + U_{i,j-1}}{k^2}\right) \tag{2.31}$$

Com essa notação, a equação (2.30) juntamente com a condição de contorno, pode ser escrita como:

$$-\Delta_\delta U_{i,j} = f(x_i, y_j), (x_i, y_j) \in R_\delta \tag{2.32}$$

$$U_{i,j} = g(x_i, y_j), (x_i, y_j) \in \partial R_\delta \tag{2.33}$$

A equação (2.32) quando aplicada nos pontos interiores R_δ , resulta num sistema linear simétrico com $(N-1)*(M-1)$ equações. (Leveque, 2007) Espera-se que resolvendo esse sistema linear, os valores $U_{i,j}$ encontrados seja uma boa aproximação para $u_{i,j}$. De fato, veremos que a solução obtida por (2.32) converge para $u_{i,j}$ se a malha for refinada.

O sistema linear proveniente (2.32) matricial $AU = r$, tal que:

$$\underbrace{\begin{bmatrix} a & b & & & & c & & & \\ b & a & b & & & & \ddots & 0 & \\ & \ddots & \ddots & \ddots & & 0 & & \ddots & \\ & & \ddots & \ddots & \ddots & & & & c \\ & & & \ddots & \ddots & \ddots & & & \\ c & & & & \ddots & \ddots & \ddots & & \\ & \ddots & & 0 & & \ddots & \ddots & \ddots & \\ & 0 & \ddots & & & & b & a & b \\ & & & c & & & & a & b \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} U_{1,1} \\ \vdots \\ U_{N-1,1} \\ U_{1,2} \\ U_{2,2} \\ \vdots \\ U_{N-1,2} \\ \vdots \\ U_{N-1,M-1} \end{bmatrix}}_{U} = \underbrace{\begin{bmatrix} f_{1,1} + \frac{g(x_0,y_1)}{h^2} + \frac{g(x1,y_0)}{k^2} \\ f_{2,1} + \frac{g(x_2y_0)}{h^2} \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ f_{N-1,1} + \frac{g(x_N,y_1)}{h^2} + \frac{g(x_{N-1},y_0)}{k^2} \end{bmatrix}}_{r}$$

onde $a = \frac{2}{h^2} + \frac{2}{k^2}$, $b = \frac{-1}{h^2}$ e $c = \frac{-1}{k^2}$.

Condição de Fronteira de Neumann: Domínio Retangular

Quando usamos a condição de Dirichlet, o valor de u é conhecido na fronteira. A condição de Neumann envolve o valor da derivada de u e, portanto, é necessário determinar u nos pontos da fronteira. Para isso, aplica-se a equação de Poisson na fronteira.

Considerando o ponto C da figura anterior, obtemos a equação de Poisson discreta,

$$-\left[\frac{U_O - 2U_C + U_L}{h^2} + \frac{U_N - 2U_C + U_S}{k^2}\right] = f_C \tag{2.34}$$

O ponto fictício/fantasma [48] U_O é obtido em função dos pontos interiores pela aplicação da condição de Neumann.

$$\frac{\partial^2 u}{\partial x^2}[C] = f_1 y_C \Rightarrow \frac{U_L - U_O}{2h} = f_1(y_C) \Rightarrow U_O = U_L - Shf_1(u_C) \tag{2.35}$$

Substituindo a equação (2.35) na equação (2.34), obtemos:

$$\frac{2U_L - 2U_C}{h^2} + \frac{U_N - 2U_C + U_S}{k^2} = -f_C + \frac{2}{h}f_1(y_C) \tag{2.36}$$

Observando a equação (2.36), vemos que o efeito da condição de Neumann na equação discretizada é a modificação do termo independente e de alguns elementos da matriz. As modificações da matriz são as mais importantes visto que essas modificações podem destruir propriedades importantes da matriz tais como simetria e dominância da diagonal. (Leveque, 2007)

Capítulo 3

Trabalhos Relacionados

Na literatura podemos observar que a aplicação de RNAs sempre evolui para tentar auxiliar na resolução de problemas envolvendo EDs em situações onde os MNC apresentam certa dificuldade: como a dependência da malha computacional e questões como a interpolação da aproximação obtida [33, 34, 10].

3.1 Hard assignment

A ideia de resolver equações diferenciais usando de *Machine Learning* foi introduzida em 1996 por Isaac Lagaris, Aristidis Likas e Dimitrios Fotiadis (University of Ioannina, Greece) [33].
Essa abordagem tem as seguinte vantagens:

- A solução aprendida pela RNA tem é diferenciável e de forma analítica fechada
- O método é geral
- Implementação eficiente em arquiteturas paralelas
- A solução aprendida pela RNA tem boa generalização em pontos não treinados
- Uma acurácia alta pode ser alcançada com poucos parâmetros.
- Não apresenta risco de overfitting

Definimos que a RNA $u_t = N(w, b)$ é uma aproximação para u, onde u é a solução para o problema de valor de contorno:

Definimos $\Omega \subseteq \mathbb{R}^n$. Dada uma equação diferencial geral da forma:

$$G(\vec{x}, \Psi(\vec{x}), \nabla\Psi(\vec{x}), \nabla^2\Psi(\vec{x})) = 0 \quad \in \Omega,$$

sujeita a condição de contorno: $\Psi(\vec{x}) = g(\vec{x}) \in \Gamma \subseteq \partial\Omega$, onde $\Psi : \Omega \subseteq \mathbb{R}^n \to \mathbb{R}, G : \mathbb{R}^n \times \mathbb{R} \times \mathbb{R}^n \times \mathbb{R}^{n\times n} \to \mathbb{R}$ e $g : \Gamma \to \mathbb{R}$. Encontre a solução Ψ. Parametrizamos então Ψ com uma RNA, com $\Psi = \Psi_\theta$ para os parâmetros θ

Definição da Função de Custo

Definimos a função de custo como sendo o resíduo de Ψ_θ aproximado pela RNA aplicado em G:

$$C(\Psi_\theta, \vec{x}) = G \tag{3.1}$$

Onde C representa a função de custo (resíduo), nosso objetivo é que esse valor seja o mais próximo possível de zero. Porém como vem da teoria, para que o PVF tenha uma solução única devemos adicionar condições de contorno ao mesmo.

Definição da Função de Loss

Definimos a função de Loss como sendo o resíduo entre o Ψ_θ aproximado pela RNA e as derivadas de grau inferior aplicadas na função f:

$$E(\theta) = \frac{1}{N}\left(\sum_{i=1}^{N} C(N(\theta, \vec{x_i}), \vec{x_i})^2\right)_{(\Omega)} \tag{3.2}$$

Onde E representa a função de *loss* (perda), nosso objetivo novamente é que esse valor seja o mais próximo possível de zero. E será justamente este valor como visto anteriormente que usaremos no Gradient Descendent.

Definição da Arquitetura da RNA

Como foi discutido, uma RNA com apenas uma camada oculta podemos aproximar qualquer função, desde que as condições para função de ativação e suficiente treinamento seja realizado.

Porém o Teorema da Aproximação Universal pode ser estendido de forma análoga para RNAs mais profundas, ou seja, com mais camadas ocultas. O maior número de

camadas geralmente influencia positivamente na aproximação, dado que precisaremos logaritimicamente de um menor número de neurônios para aproximar as curvas da função, o que leva a um menor custo computacional para realizar o treinamento [25, 41].

Para o caso em que se está trabalhando com EDOs, uma RNA com apenas uma camada oculta é o suficiente para realizar a aproximação. Porém quando estamos trabalhando com EDPs, como citado em Berg e Nyström (2018), geralmente 4 camadas ocultas demonstram melhores resultados tanto em aproximação, quanto em velocidade de treinamento.

Lidando com as condições de contorno

Em [33] a primeira abordagem conhecida para lidar com as condições de contorno foi proposta, nela a ideia é criar "trial solutions"que satisfaçam automaticamente as condições de contorno fornecidas pelo problema. Sejam elas do tipo Newmman, Diriclet ou mistas.

$$\Psi_t(\vec{x}) = A(\vec{x}) + F(\vec{x}, N_\theta(\vec{x}))$$

A função A é definida de modo a satisfazer as condições de contorno, F não contribui para as condições de contorno (define uma "distância"em relação aos contornos) e N_θ é uma RNA de parâmetros θ. A e F precisam ser definidas "na mão"dependendo do problema a ser trabalhado. O que se torna algo não trivial para contornos complexos e mesmo para contornos do tipo misto em geometrias mais regulares, encontrar A não é uma tarefa simples, que será visto nas próximas seções, essa dificuldade levou a o surgimento do Soft Assigment e posteriormente abordagens híbridas.

Função A em EDPs

Quando se está trabalhando com EDPs, a única modificação a ser realizada na RNA é o número de neurônios presentes na camada de entrada, por exemplo, se estivermos trabalhando duas variáveis espaciais no problema, termos dois neurônios na camada de entrada e assim sucessivamente, este fato facilita bastante o uso do método em problemas com maior dimensão, dado que a construção é natural, fato este que não é tão simples de se lidar quando se trabalha com métodos numéricos.

- Para contornos do tipo Dirichlet:

$$\begin{aligned} A(x,y) &= (1-x)f_0(y) + xf_1(y) + \\ &+ (1-y)\{g_0(x) - [(1-x)g_0(0) + xg_0(1)]\} + \\ &+ y\{g_1(x) - [(1-x)g_1(0) + xg_1(1)]\} \end{aligned}$$

- Para contornos mistos:

$$\begin{aligned} B(x,y) &= (1-x)f_0(y) + xf_1(y) + \\ &+ g_0(x) - [(1-x)g_0(0) + xg_0(1)] \\ &+ y\{g_1(x) - [(1-x)g_1(0) + xg_1(1)]\} \end{aligned}$$

As funções f_i e g_j geram os valores nos pontos de contornol.

3.2 Soft Assignment

Uma outra abordagem para lidar com os contornos em uma RNA que aproxima a solução de uma ED sujeita a condições de contorno ou iniciais foi proposta então por Fernandez [23] como uma maneira mais simples de trabalhar no problema, ainda utilizando de uma abordagem mais familiar a aqueles que trabalham com RNAs, dado que não se faz o uso de funções intermediárias (trial solution), a saída da RNA é diretamente a aproximação da solução.

Para tal a função de *loss* é modificada de modo a "penalizar"a RNA caso os valores de contorno não sejam satisfeitos. Para isso adicionamos os valores de contorno ficando então com o seguinte *loss*:

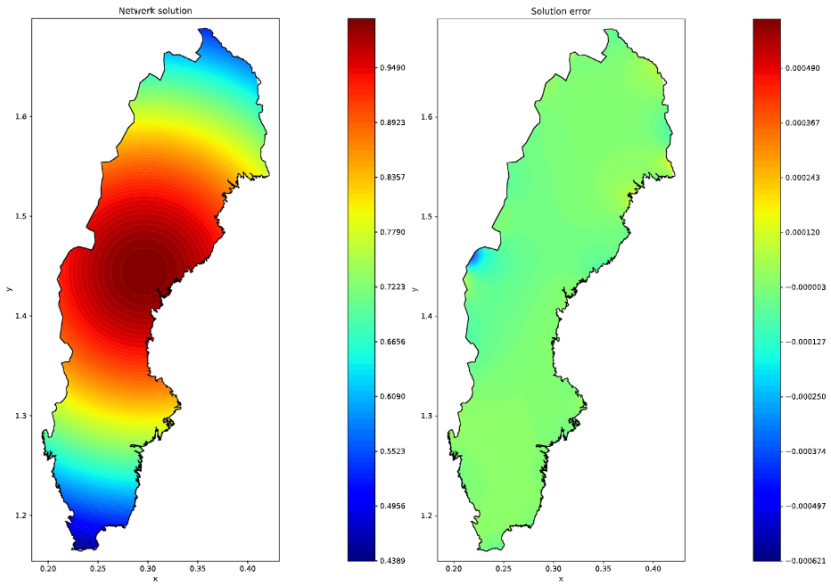
$$E(\theta) = \frac{1}{N}\left(\sum_{i=1}^{N} C(N(\theta, \vec{x_i}), \vec{x_i})^2\right)_{(\Omega)} + \sum [\Psi_\theta - g]^2_{(\Gamma)} \tag{3.3}$$

Onde $N(\theta, \vec{x_i}) = \Psi_\theta(\vec{x})$, ou seja, o valor de saída da RNA é usado diretamente na aproximação.

3.3 Contornos Irregulares

Em Lagaris et al (1999) foi introduzida uma aplicação interessante do uso de RNAs na solução de EDPs com tornos irregulares, como for exemplo do domínio da Figura:

A malha computacional gerada não altera como se trabalha com os pontos em relação a um domínio normal, já que simplesmente temos um conjunto de ponto que não são dependentes uns dos outros, enquanto que em métodos numéricos como mostrado em [31], muitas vezes se faz necessário o uso de **pontos fantasmas** para conseguir aproximar as derivadas numéricamente. Em Berg e Nyström (2018) a ideia é elevada a domínios ainda mais complexos, neste caso trabalharam com um domíno no formato do mapa da Suécia como mosta a Figura 3.1:



(a) ANN solution to the 2D stationary diffusion equation.

(b) Difference between the exact and computed solution.

Figura 3.1: Solução pela RNA com 5 camadas ocultas e 10 neurônios cada.
Fonte: Berg e Nyström (2018)

Berg e Nyström (2018) relataram que uma malha ótima não pode ser gerada mesmo depois de 16h de processamento com um gerador de malha do pacote FEniCS (estado da arte no publicação do artigo). Domı nio representado por um polı gono com 160.876 vértices. Treinar a RNA em um laptop demorou 10 minutos, ainda tendo margem para o desenvolvimento em bibliotecas como PyTorch ou TensorFlow [2]. Um dos problemas observados por Berg e Nyström (2018) foi que o erro para os contornos em problemas 2D eram os valores mais elevados, como pode ser visto nas Figuras 3.2 e 3.3, que mostram o erro da solução aproximada para uma EDP do tipo Poisson, para melhorar a aproximação nesses pontos, a ideia foi mesclar as abordagens de soft assignment com hard assignment,

pra tanto uma nova RNA com apenas uma camada oculta foi treinada de modo a representar o papel da função A citada anteriormente, essa RNA tem como objetivo informar o quão próximo um ponto está das bordas de modo a simular a satisfação automática dos valores de contorno, como descreve a abordagem de hard assignment.

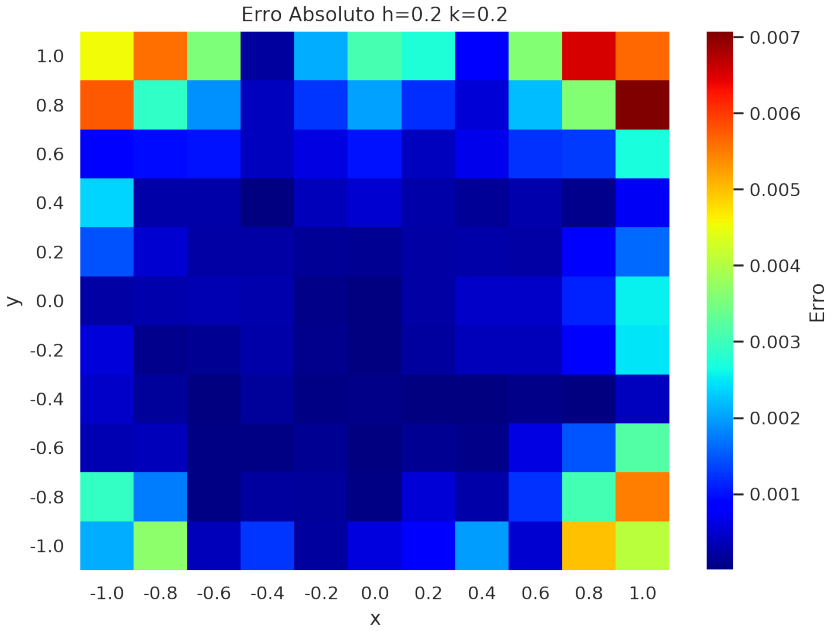


Figura 3.2: Erro Absoluto RNA

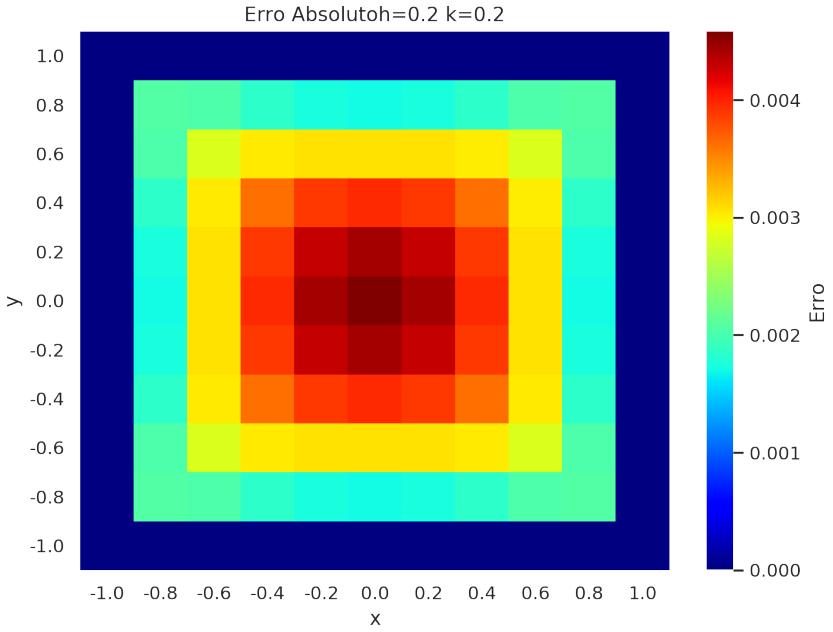


Figura 3.3: Erro Absoluto Gauss-Seidel

Os autores ainda fizeram um interessante estudo da influência do número de camadas ocultas na convergência e valores de *loss* obtidos como mostra a Figura 3.4. Mostrando que para esse método, de um modo geral 4 camadas parecem lidar melhor com problemas

em duas dimensões, enquanto que o uso de apenas uma camada como proposto em Lagaris et al (1997) mostra uma estagnação no *loss* obtido.

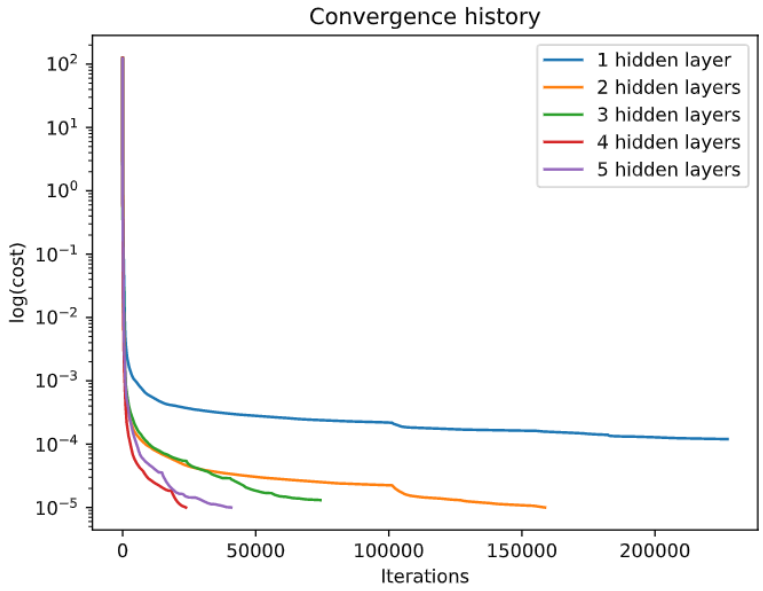


Figura 3.4: De um modo geral 4 camadas com 10 neurônios cada, se adequam melhor a problemas 2D.
Fonte: Berg e Nyström (2018)

Outro resultado interessante observado, são as grandes capacidades da RNA na interpolação em pontos que não foram utilizados para o treinamento da RNA, como podemos observar na tabela da Figura, a ordem do erro se mantém a mesma para os pontos que não foram treinados pela RNA, equanto que em métodos numéricos, como neste caso o Método dos Elementos Finitos, o erro geralmente aumenta em uma ordem de 10^{-2}.

	Neural Method		*Finite Element*	
Problem No.	Training set	Interpolation set	Training set	Interpolation set
5	5×10^{-7}	5×10^{-7}	2×10^{-8}	1.5×10^{-5}
6	0.0015	0.0015	0.0002	0.0025
7	6×10^{-6}	6×10^{-6}	7×10^{-7}	4×10^{-5}
8	1.5×10^{-5}	1.5×10^{-5}	6×10^{-7}	4×10^{-5}

Figura 3.5: Erro para pontos interpolados.
Fonte:Lagaris et al (1998)

3.4 Solução numérica de EDPs via análise de dados

Raissi et al (2017) propõe uma nova abordagem de RNA, fisicamente informada, mais especificamente fazendo uso de Deep Leaning. A ideia é direcionado por dados, treinar uma RNA supervisionada para resolver leis físicas descritas por uma ED não linear geral.

A solução inferida pela RNA para as equações diferenciais parciais são completamente diferenciáveis com relação as coordenadas de entrada e parâmetros livres.

Ao se analisar sistemas complexos em física, biologia e engenharia o custo de aquisição de dados sempre se mostra como um fator proibitívo, o problema a ser lidado é o de tirar conclusões com base em informações parciais, com essa pequena quantidade de dados a grande maioria das técnicas estado-da-arte em *Machine Learning* falham em prover robustez e garantias de convergência. Para ultrapassar essas difuculdades a ideia é tomar como base conhecimentos apriori de leis que regem fenômenos, essas informações podem agir como fator de regularização, de tal modo diversas possíveis soluções inferidas pela RNA podem ser descartadas, o que acelera o quão rápido se pode aproximar da solução real. Novamente a abordagem assim como nos trabalhos elucidados anteriormente, faz uso da Diferenciação Automática para encontrar as derivadas da RNA com relação as variáveis de entrada ou parâmetros.

A abordagem é implementada usando do framework TensorFlow [2], que é uma das mais populares bibliotecas open source para *Machine Learning*. Diferentemente dos trabalhos anteriores a minimização dos parâmetros é feita usando do Mean Squered Error(MSE) do *loss*.

$$MSE = MSE_u + MSE_f \tag{3.4}$$

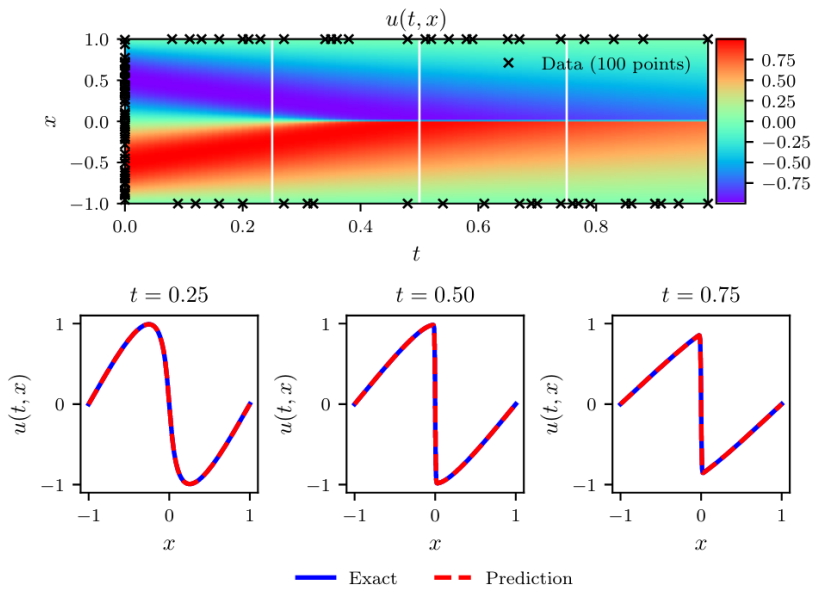


Figura 3.6: Solução predita para a equação de Burguer
Fonte: Raissi, Perdikaris e Karniadakis (2017)

Onde MSE_u representa os valores iniciais e de contorno do conjunto de treinamento,

enquanto MSE_f é o *loss* calculado em um conjunto finito de pontos de colocação, que força a estrutura imposta pela ED original do problema.

Layers \ Neurons	10	20	40
2	7.4e-02	5.3e-02	1.0e-01
4	3.0e-03	9.4e-04	6.4e-04
6	9.6e-03	1.3e-03	6.1e-04
8	2.5e-03	9.6e-04	5.6e-04

Figura 3.7: Erro entre a solução predita e a solução para a equação de Burguer
Fonte: Raissi, Perdikaris e Karniadakis (2017)

Demonstraram ainda como mostra a Figura 3.7, que ao aumentar o número de camadas ocultas e neurônios por camada a acurácia também aumentou, dado o aumento da capacidade da RNA em aproximar funções mais complexas.

Capítulo 4

Conclusão

O desenvolvimento desta revisão bibliográfica proporcionou o embasamento teórico na área de solução de problemas envolvendo EDs usando de MNC e como aplicar RNAs para resolver essas equações, essencial para dar inicio a próxima etapa que será a implementação da ferramenta para solução de diversos conjuntos de problemas envolvendo PIV e PVF, fazendo uso direto e indireto de RNAs.

Os MNCs tem uma grande robustez e conseguem resolver uma enorme gama de problemas, provendo ótimas aproximações, que podem ser ainda melhoradas com modificações seja no algoritmo, como uso de bibliotecas de vetorização [11] e paralelização, ou no método em si, como é o caso do algoritmo de Euler melhorado [5].

Uma das modificações possíveis é utilizar de uma RNA para auxiliar os MNCs, como por exemplo gerando parâmetros para eles de modo a acelerar sua convergência [20]. Também é possível usar RNAs para resolver problemas principalmente quando são necessários métodos mais generalistas e que sejam independentes de malha, como foi elucidado por Berg e Nyström (2018). O método também pode ganhar grande eficiência dada a possibilidade de poder ser implementado em arquiteturas paralelas [25].

Depois de treinada uma RNA apresenta grande versatilidade e conta com um menor custo computacional (espaço de memória) já que não é necessário aumentar proporcionalmente o número de pontos conhecidos para melhorar a aproximação e qualquer ponto do domínio pode ter seu valor correspondente gerado pela RNA (interpola automaticamente), como já discutido por Lagaris et al (1998), porém ao contrário dos MNCs não conseguimos ter um controle mais rigoroso do erro e convergência [5], e muitas das etapas da construção/treinamento de uma RNA só podem ser feitas empiricamente como cita [25] e [41].

Bibliografia

[1] URL: http://www.uel.br/projetos/matessencial/superior/calculo/derivada/derivada2.htm. (Acesso em: 14.10.20).

[2] Martin Abadi et al. «Tensorflow: A system for large-scale machine learning». Em: 2016.

[3] Luis Antonio Aguirre. *Introdução à Identificação de Sistemas–Técnicas Lineares e Não-Lineares Aplicadas a Sistemas Reais*. Editora UFMG.

[4] Almir Olivette Artero. *Inteligência Artificial: Teórica e Prática*. Livraria da Física, 2009.

[5] Uri M Ascher e Chen Greif. *A first course on numerical methods*. Vol. 7. Siam, 2011.

[6] Renato Assunção. *Deep Learning*. URL: https://homepages.dcc.ufmg.br/~assuncao/AAP/Sem%5C%2002%5C%20-%5C%20Aula%5C%2003.pdf. (Acesso em: 28.10.20).

[7] Jorge Muniz Barreto. «Inteligência artificial no limiar do século XXI». Em: *Florianópolis: PPP edições* 97 (1999).

[8] R. C. Bassanezi. *Equações Diferenciais Ordinárias: Um curso introdutório*. BC&T - UFABC.

[9] Atılım Günes Baydin et al. «Automatic differentiation in machine learning: a survey». Em: *The Journal of Machine Learning Research* 18.1 (2017), pp. 5595–5637.

[10] Jens Berg e Kaj Nyström. «A unified deep artificial neural network approach to partial differential equations in complex geometries». Em: *Neurocomputing* 317 (2018), pp. 28–41.

[11] Aaron Birkland. *Vectorization Lab Parallel Computing at TACC: Ranger to Stampede Transition*. Cornell Center for Advanced Computing, 2013.

[12] William E Boyce, Richard C DiPrima e Douglas B Meade. *Elementary differential equations*. John Wiley & Sons, 2017.

[13] A de P Braga. *Redes neurais artificiais: tRedes neurais artificiaiseoria e aplicações*. Livros Técnicos e Cientı ficos, 2000.

[14] Frederico Ferreira Campos Filho. *Algoritmos numéricos*. LTC, 2007.

[15] Tian Qi Chen et al. «Neural ordinary differential equations». Em: *Advances in neural information processing systems*. 2018, pp. 6571–6583.

[16] Python Community. *Python*. URL: `https://www.python.org/`. (Acesso em: 10.08.20).

[17] José Alberto Cuminato e Messias Meneguette. *Discretização de equações diferenciais parciais: técnicas de diferenças finitas*. Sociedade Brasileira de Matemática, 2013.

[18] Haskell B Curry. «The method of steepest descent for non-linear minimization problems». Em: *Quarterly of Applied Mathematics* 2.3 (1944), pp. 258–261.

[19] George Cybenko. «Approximation by superpositions of a sigmoidal function». Em: *Mathematics of control, signals and systems* 2.4 (1988), pp. 303–314.

[20] M Dehghanpour, A Rahati e E Dehghanian. «ANN-based modeling of third order runge kutta method». Em: *Journal of Advanced Computer Science & Technology* 4.1 (2015), pp. 180–189.

[21] Matt Johnson e Jamie Townsend Dougal Maclaurin David Duvenaud. *Autograd*. URL: `https://github.com/HIPS/autograd`. (Acesso em: 10.08.20).

[22] Laboratório de Estatística e Geoinformação - LEG/UFPR. *Gradiente descendente*. URL: `http://cursos.leg.ufpr.br/ML4all/apoio/Gradiente.html`. (Acesso em: 12.04.20).

[23] Anita Maria da Rocha Fernandes. «Inteligência artificial: noções gerais». Em: *Florianópolis, SC: VisualBooks Editora* (2003).

[24] Éliton Fontana. *Introdução ao Método de Diferenças Finitas com Aplicações em Engenharia Química*. URL: `https://fontana.paginas.ufsc.br/files/2017/02/apostila_metII_20191.pdf`. (Acesso em: 27.08.20).

[25] Ian Goodfellow, Yoshua Bengio e Aaron Courville. *Deep learning*. MIT press, 2016.

[26] H.L. Guidorizzi. *Um curso de cálculo*. LTC, 2013.

[27] Simon Haykin. «Feedforward neural networks: An introduction». Em: *Retrieved June* 20 (2004), p. 2010.

[28] Simon Haykin. *Redes neurais: princı pios e prática*. Bookman Editora, 2001.

[29] Kurt Hornik, Maxwell Stinchcombe, Halbert White et al. «Multilayer feedforward networks are universal approximators.» Em: *Neural networks* 2.5 (1989), pp. 359–366.

[30] Volker John. «Numerical methods for partial differential equations». Em: *Lecture notes* (2013).

[31] D. F. Mayers K. W. Morton. *Numerical Solution of PartialDifferential Equations*. Cambridge University Press, 2005.

[32] Diederik P Kingma e Jimmy Ba. «Adam: A method for stochastic optimization». Em: *arXiv preprint arXiv:1412.6980* (2014).

[33] Isaac E Lagaris, Aristidis Likas e Dimitrios I Fotiadis. «Artificial neural networks for solving ordinary and partial differential equations». Em: *IEEE transactions on neural networks* 9.5 (1998), pp. 987–1000.

[34] Isaac Elias Lagaris, Aristidis Likas e Dimitrios G Papageorgiou. «Neural network methods for boundary value problems defined in arbitrarily shaped domains». Em: *arXiv preprint cs/9812003* (1998).

[35] James V Lambers e Amber C Sumner. *Explorations in numerical analysis*. 1. : World Scientific, 2019.

[36] Randall J LeVeque. *Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems*. SIAM, 2007.

[37] Andrew J Meade Jr e Alvaro A Fernandez. «The numerical solution of linear ordinary differential equations by feedforward neural networks». Em: *Mathematical and Computer Modelling* 19.12 (1994), pp. 1–25.

[38] Marvin Minsky e Seymour Papert. «Perceptrons: An essay in computational geometry». Em: *MIT Press.* (1969).

[39] Tom M. Mitchell. *Machine Learning*. McGraw-Hill Science Engineering Math, 1997.

[40] Luiz Henry Monken, Ivo Neitzel, Ed Pinheiro Lima et al. «Resolução de equações diferenciais por redes neurais artificiais: problemas com gradientes elevados e domı nios arbitrários». Em: *Acta Scientiarum. Technology* 27.1 (2005), pp. 7–16.

[41] Andrew Ng. «Machine learning yearning». Em: *URL: http://www. mlyearning. org/(96)* (2017).

[42] Robert E O'Malley Jr e Robert E O'Malley. *Thinking about ordinary differential equations*. Cambridge University Press, 1997.

[43] Boris T Polyak. «Some methods of speeding up the convergence of iteration methods». Em: *USSR Computational Mathematics and Mathematical Physics* 4.5 (1964), pp. 1–17.

[44] Alfio Quarteroni, Fausto Saleri e Paola Gervasio. *Scientific computing with MATLAB and Octave*. Vol. 2. 2006.

[45] Maziar Raissi, Paris Perdikaris e George Em Karniadakis. «Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations». Em: *arXiv preprint arXiv:1711.10561* (2017).

[46] Márcia A Gomes Ruggiero e Vera Lúcia da Rocha Lopes. *Cálculo numérico: aspectos teóricos e computacionais*. Makron Books do Brasil, 1997.

[47] Malcon A Tafner, Marcos de Xerez e Ilson W Rodrigues Filho. *Redes neurais artificiais: introdução e princı pios de neurocomputação*. Eko, 1995.

[48] Tesfu Tezera. *Explicit Finite Difference Schemes using Ghost pointsfor the Heat equation with Insulated ends*. URL: `http://etd.aau.edu.et/bitstream/handle/123456789/9312/Tesfu%5C%20Tezera.pdf`. (Acesso em: 02.09.20).

[49] Tijmen Tieleman e Geoffrey Hinton. «Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude». Em: *COURSERA: Neural networks for machine learning* 4.2 (2012), pp. 26–31.

[50] *Uma Introdução às Redes Neurais*. URL: `http://www.din.uem.br/ia/neurais`. (Acesso em: 10.08.20).

[51] Adhemar Maria do Valle Filho et al. «Um modelo para implementação de consciência em robôs móveis». Em: (2003).

[52] Marcos Eduardo Valle. *Slides de Cálculo Numérico: Métodos de Runge-Kutta*. URL: `https://www.ime.unicamp.br/~valle/Teaching/MS211/Aula21.pdf`. (Acesso em: 10.09.20).

[53] Sangeeta Yadav e Sashikumaar Ganesan. «How Deep Learning performs with Singularly Perturbed Problems?» Em: (2019), pp. 293–297.

[54] S. P. Zambiasi. *Introdução as Redes Neurais Artificiais*. URL: `https://www.gsigma.ufsc.br/~popov/aulas/ia/modulo9.pdf`. (Acesso em: 10.09.20).

[55] ZaparoliAlexsandra. «Protótipo de Software para Controle de Acesso de Funcionários utilizando Redes Neurais para Identificação de Impressão Digital». Tese de doutoramento. UniversidadeRegionalDeBlumenau, 2002.

[56] Zheyan Zhang et al. «MeshingNet: A New Mesh Generation Method based on Deep Learning». Em: *arXiv preprint arXiv:2004.07016* (2020).