Redes Neurais Informadas pela Física

Temas

- Contexto Histórico
- Funcionamento PINNs
- Exemplo PINN
- Bibliografia

Contexto Histórico

- Aproximadamente em 2015 começaram a estudar as redes neurais multilayer perceptron para resolver as equações de Navier–Stokes;
- Até que em 2019, Raissi implementou um modelo informado pela física, conhecido originalmente por Physics-informed Neural Networks (PINNs);
- O modelo foi desenvolvido para resolver as EDP presentes na equação de Navier–Stokes

$$\rho\left(\frac{\partial V_x}{\partial t}+\frac{\partial V_x}{\partial x}+\frac{\partial V_x}{\partial y}+\frac{\partial V_x}{\partial z}\right)=-\frac{\partial p}{\partial x}+\rho g_x+\mu\left(\frac{\partial^2 V_y}{\partial x^2}+\frac{\partial^2 V_y}{\partial y^2}+\frac{\partial^2 V_y}{\partial z^2}\right)$$

$$\rho\left(\frac{\partial V_y}{\partial t}+\frac{\partial V_y}{\partial x}+\frac{\partial V_y}{\partial y}+\frac{\partial V_y}{\partial z}\right)=-\frac{\partial p}{\partial y}+\rho g_y+\mu\left(\frac{\partial^2 V_y}{\partial x^2}+\frac{\partial^2 V_y}{\partial y^2}+\frac{\partial^2 V_y}{\partial z^2}\right)$$

$$\rho\left(\frac{\partial V_z}{\partial t}+\frac{\partial V_z}{\partial x}+\frac{\partial V_z}{\partial y}+\frac{\partial V_z}{\partial z}\right)=-\frac{\partial p}{\partial z}+\rho g_z+\mu\left(\frac{\partial^2 V_z}{\partial x^2}+\frac{\partial^2 V_z}{\partial y^2}+\frac{\partial^2 V_z}{\partial z^2}\right)$$

Contexto Histórico

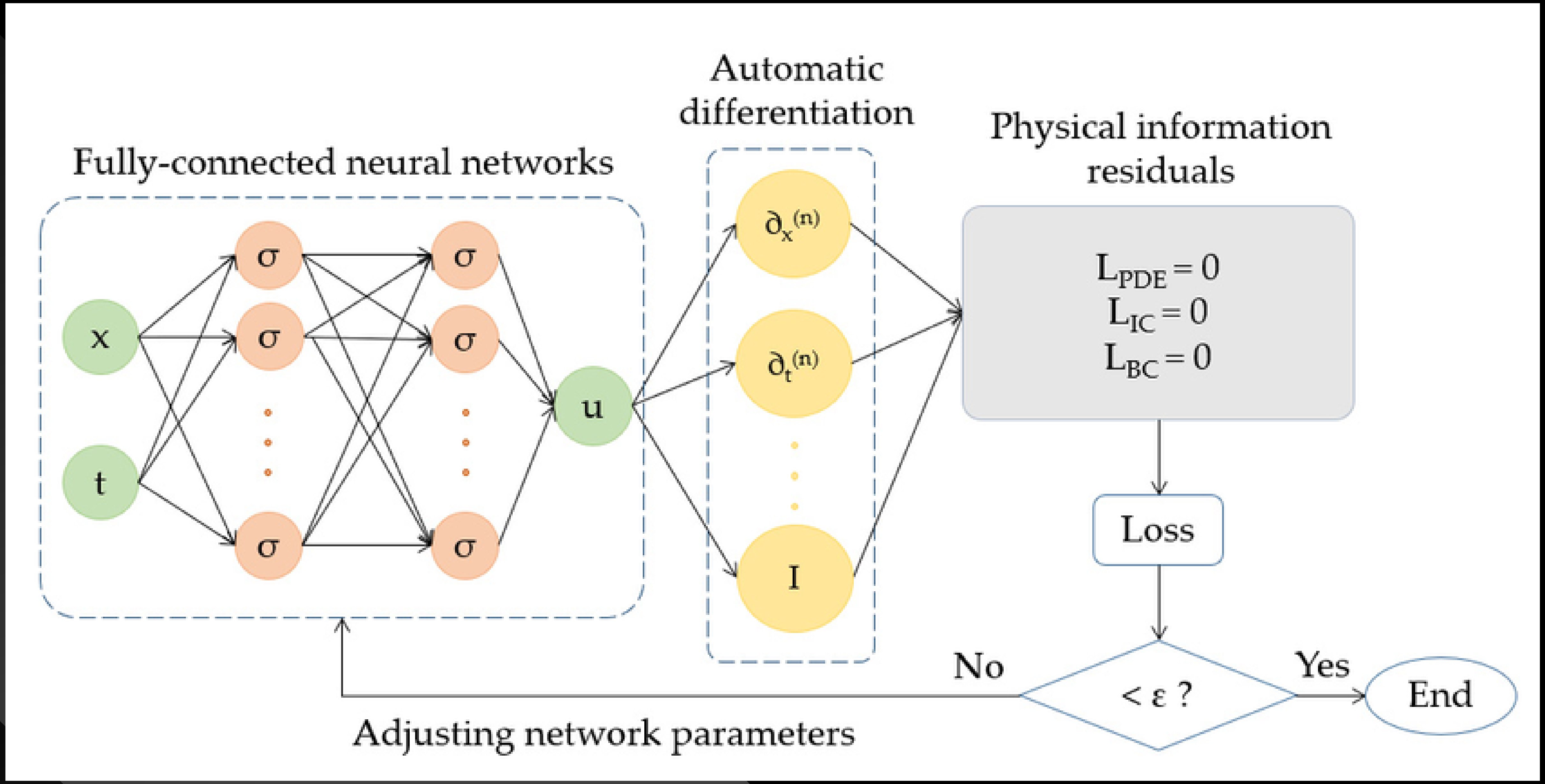
- A rede neural está sendo considerada uma alternativa para substituir o método dos elementos finitos;
- Em 2021, Jin et al realizou um estudo para analisar a acurácia e a convergência das redes neurais.
  - O estudo foi baseado na variação no número de épocas, taxa de aprendizado e do otimizador, além de variar os pesos iniciais para ver o comportamento da rede.

Funcionamento PINNs

- As redes neurais informadas pela física são baseadas em duas propriedades das redes neurais MLP:
  - Uma rede neural MLP é um aproximador de função universal;
  - É possível calcular as derivadas, de qualquer ordem, de uma saída da rede neural, com relação a qualquer uma das entradas usando a automatic differentiation – Diferenciação automática (cálculo automático do gradiente descendente);
  - Com esses dois conceitos, a função de custo da rede neural é construída para que, ao ser minimizada, a EDP seja satisfeita.
  - Ou seja, o custo é tomado como o erro médio quadráfico da EDP, como isso não fornece uma solução única, uma segunda função custo é construída, sendo ela o erro médio quadrático da condição de contorno em relação a saída obtida pela rede.

Funcionamento PINNs

- Em um esquemático, as PINNs funcionam assim:



Funcionamento PINNs

- O começo da PINN é o mesmo de uma RNA MLP, a diferença está no fim, em que utilizamos a diferenciação automática.
  - AD (diferenciação automática) é o cálculo automático dos gradientes, de ordem desejada.
- Com os gradientes tomados, podemos calcular o valor de saída para os pontos por meio da EDP informada.
- Tendo feito isto, em seguida é calculada o custo, constituído por:
  - Custo na condição de contorno;
  - Custo na condição inicial;
  - Custo da EDP.
- Com o custo calculado, ele é informado na etapa de backpropagation, para atualizar os pesos e vieses da rede neural, afim de se adaptar adequadamente para a função desejada.

Funcionamento PINNs

- Por fim, a forma geral da EDP para a PINN é a seguinte:

$$u_t + \mathcal{N}[u] = 0,\ x \in \Omega,\ t \in [0,\ T],$$

- Sendo:
  - u(t,x) o valor de saída que desejamos;
  - N um operador diferencial não linear;

Exemplo PINN

- Para construir a PINN, foi tomado como base o exemplo do Maziar, em que era utilizada a equação de Burgers para treinar uma rede neural.
- A equação utilizada no exemplo foi a de um sistema massa mola amortecido

$$m\frac{d^2x}{dt^2} + \mu\frac{dx}{dt} + kx = 0$$

$$f := mu_{tt} + \mu u_t + ku = 0, x \in [0,2]$$

- As condições de contorno/inicial são:

$$x(0) = A \, , \frac{dx}{dt} = 0$$

Exemplo PINN

- Para construir a PINN, foi tomado como base o exemplo do Maziar, em que era utilizada a equação de Burgers para treinar uma rede neural.
- A equação utilizada no exemplo foi a de um sistema massa mola amortecido

$$m\frac{d^2x}{dt^2} + \mu\frac{dx}{dt} + kx = 0$$

$$f := mu_{tt} + \mu u_t + ku = 0, x \in [0,2]$$

- As condições de contorno/inicial são:

$$x(0) = A\,, \frac{dx}{dt} = 0$$

Exemplo PINN

- O custo da função vai ser composto da condição de contorno e dos dados tomados, assim:

$$MSE = MSE_f + MSE_{c.i.},$$

$$MSE_f = \frac{1}{N}\sum_{1}^{N}\left|f_{pinn} - f_{real}\right|^2$$

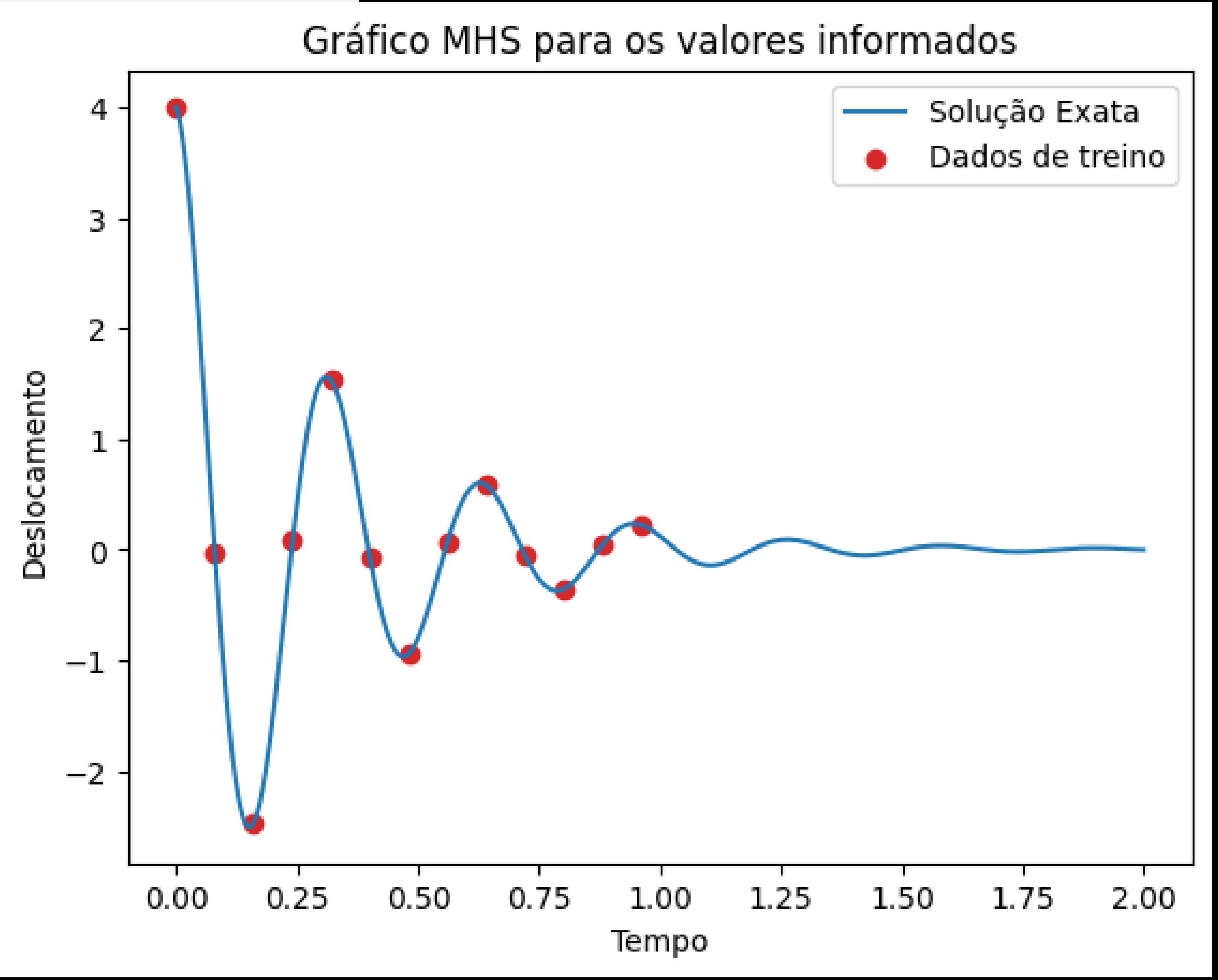
$$MSE_{c.i.} = \frac{1}{N}\sum_{i=1}^{N}\left|f_{pinn}(0) - f_{real}(0)\right|$$

Exemplo PINN

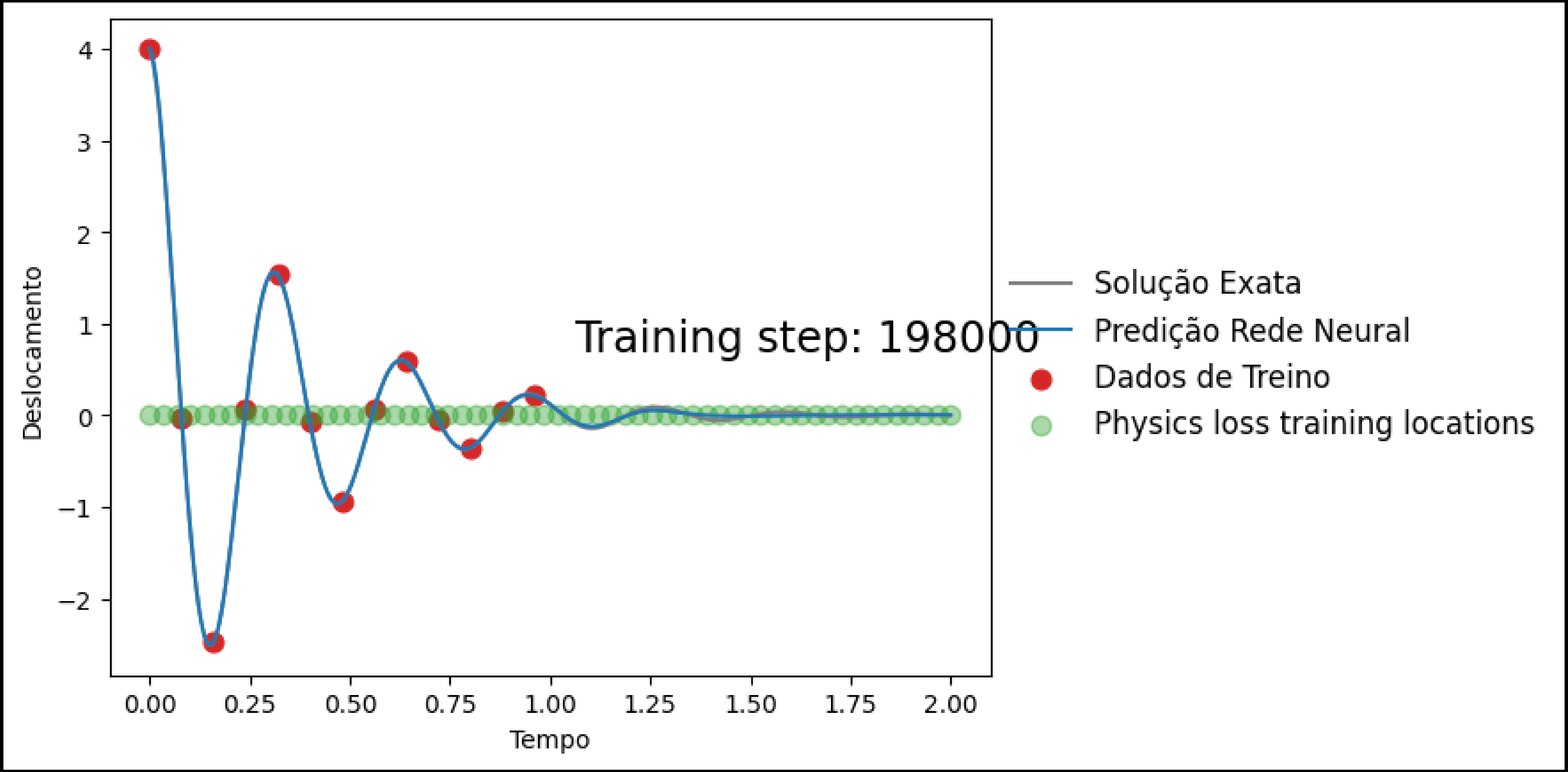
- No código, a aplicação é a seguinte:

```python
class modelo_RNA(nn.Module):

    def __init__(self, no_entrada, no_oculto, no_saida, n_camada_oculta):
        super().__init__()
        activation = nn.Tanh
        self.entrada_oculta = nn.Sequential(*[
                        nn.Linear(no_entrada, no_oculto),
                        activation()])
        self.oculta_oculta = nn.Sequential(*[
                        nn.Sequential(*[
                            nn.Linear(no_oculto, no_oculto),
                            activation()]) for _ in range(n_camada_oculta)])
        self.oculta_saida = nn.Linear(no_oculto, no_saida)

    def forward(self, x):
        x = self.entrada_oculta(x)
        x = self.oculta_oculta(x)
        x = self.oculta_saida(x)
        return x
```

```python
#Construindo os dados de x e y

x = torch.linspace(0,2,1000).view(-1,1) #uma matriz coluna com 1000 valores indo de 0 até 2
m = 1 #Massa unitária em unidades de Kg
d = 3 #delta, deslocamento
w0 = 20 #frequencia natural
w = np.sqrt(w0**2 - d**2)
phi = np.radians(0)
A = 2

cos = torch.cos(phi+w*x) #Cosseno para valores de x
exp = torch.exp(-d*x) #Função exponencial
y = exp*2*A*cos #Valores de altura em função de x
```



```python
x_fisica = torch.linspace(0,2,60).view(-1,1).requires_grad_(True)
mi = 2*d*m
k = (w0**2)*m

torch.manual_seed(123)
modelo = modelo1(1,32,1,3)
otimizador = torch.optim.Adam(modelo.parameters(),lr=1e-4)

for i in range(200000):
    otimizador.zero_grad()
    y_pred = modelo(x_data)
    loss1 = torch.mean((y_pred-y_data)**2) #MSE da função

    #Custo para função física
    y_predfisica = modelo(x_fisica)
    dx  = torch.autograd.grad(y_predfisica, x_fisica, torch.ones_like(y_predfisica), create_graph=True)[0]#gradiente dy/dx
    dx2 = torch.autograd.grad(dx,  x_fisica, torch.ones_like(dx),  create_graph=True)[0]#gradiente d^2y/dx^2
    physics = dx2 + mi*dx + k*y_predfisica #calculando o valor segundo a fisica
    loss2 = (1e-4)*torch.mean(physics**2)

    #Backpropagation
    loss = loss1 + loss2
    loss.backward()
    otimizador.step()
```



Training step: 198000
Solução Exata
Predição Rede Neural
Dados de Treino
Physics loss training locations
Deslocamento
Tempo
4
3
2
1
0
−1
−2
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

Código PINN feito

O código está no google colab: https://colab.research.google.com/drive/1f88zqj8ahfy2Kqt-rf5w-Yk9-DFs7APu#scrollTo=Z2GhIoNPSbIO

Bibliografia

Documentação PyTorch, disponível em: https://pytorch.org/docs/stable/index.html

MOSELEY, B., “So, what is a physics-informed neural network”, disponível em: https://benmoseley.blog/my-research/so-what-is-a-physics-informed-neural-network/

JIN, X., CAI, S., LI, H., “NSFnets (Navier-Stokes flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations”

RAISSI, M., PERDIKARIS, P., KARNIADAKIS, G.E., “Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations”

DAGRADA, M., “https://towardsdatascience.com/solving-differential-equations-with-neural-networks-afdcf7b8bcc4”