Projeto 1 - MS211 - Calculo Numérico

Aluno:

Luiz Eduardo Cartolano - RA: 183012

Professor:

Maicon R. Correa

Github Link:

[Project Repository (https://github.com/luizcartolano2/ms211-numerical-calculus)](https://github.com/luizcartolano2/ms211-numerical-calculus)

Primeira Questão

Projeto 1 - Método de Newton Discreto, do livro Cálculo Numérico, Ruggiero-Lopes, 2a Ed, (p. 206).

In [1]:
```
# import das librarys usadas
import numpy as np
import pandas as pd

# Standard plotly imports
import plotly.plotly as py
import plotly.graph_objs as go
from plotly.offline import iplot, init_notebook_mode

# Using plotly + cufflinks in offline mode
import cufflinks
cufflinks.go_offline(connected=True)
init_notebook_mode(connected=True)
```

Função de Rosenbrock - F(x)

A primeira questão envolve resolver a função de Rosenbrock usando duas versões do método de Newton, o tradicional e o modificado. A função de Rosenbrock é dada pelo seguinte sistema:

$$\begin{aligned} -10 \cdot (x_1)^2 + 10 \cdot x_2 &= 0 \\ 1 - x_1 &= 0 \end{aligned}$$

A função pode ser definida em python do seguinte modo:

In [2]:
```
def F(x):
    return np.array(
        [[-10 * x[0][0] ** 2 + 10 * x[0][1],
        1 - x[0][0]]]
    ).reshape(2,)
```

Jacobiana - J(x)

A fim de não ser preciso calcular a Jacobiana, que é definida como:

$$\mathbf{J} = \begin{vmatrix} \frac{\partial f_1(x)}{\partial x_1} & \frac{\partial f_1(x)}{\partial x_2} \\ \frac{\partial f_2(x)}{\partial x_1} & \frac{\partial f_2(x)}{\partial x_2} \end{vmatrix}$$

O método de Newton Discreto realiza uma aproximação, que calcula a jacobiana a partir de um método discreto, para isso, é preciso definir primeiro:

$$e_j = (0, 0, \ldots 1, 0, 0, \ldots 0)^T$$

onde a posição j tem o valor 1. E a coluna j da jacobiana será dada por:

$$\frac{\mathbf{F}(x + he_j) - \mathbf{F}(x)}{h}$$

A duas funções seguintes foram capaz de implementar a jacobiana utilizando Python.

In [3]:
```
def e_(j, size):
    e_t = np.zeros((size,1))
    e_t[j] = 1

    return e_t
```

In [4]:
```
def J(x, h):
    f_1 = ((F(x + e_(0,2).T * h) - F(x)).T)/h
    f_2 = ((F(x + e_(1,2).T * h) - F(x)).T)/h

    return np.column_stack((f_1, f_2))
```

Método de Newton

O método de Newton consiste em, dado o ponto $x^{(k)}$, a matriz $J(x^{(k)})$ é obtida avaliando-se $J(x)$ em $x^{(k)}$ e, em seguida, o passo de Newton, $s^{(k)}$, é obtido a partir da resolução do sistema linear, $J(x^{(k)})s = -F(x^{(k)})$. Portanto, uma iteração de Newton requer que:

1. a avaliação da matriz Jacobiana em $x^{(k)}$
2. a resolução do sistema linear $J(x^{(k)})s = -F(x^{(k)})$

O algoritmo do método de Newton é dado por:

Dados x_0, $\epsilon_1 > 0$ e $\epsilon_2 > 0$:

1. calcule $F(x^{(k)})$ e $J(x^{(k)})$
2. se $|F(x^{(k)})| < \epsilon_1$, faça $\bar{x} = x^{(k)}$ e pare
3. obtenha $s^{(k)}$, a solução do sistema linear $J(x^{(k)})s = -F(x^{(k)})$
4. faça: $x^{(k+1)} = x^{(k)} + s^{(k)}$

Abaixo segue a implementação do método em Python.

In [5]:
```
def newton(F, J, x0, h, e1=1e-4, e2=1e-4):
    counter = 0
    x = x0
    if abs(np.max(F(x))) < e1:
        return x0
    s = np.linalg.solve(J(x,h),-F(x))
    xk = x + s

    while abs(np.max(F(x))) >= e1:
        counter = counter + 1
        x = xk
        s = np.linalg.solve(J(x,h),-F(x))
        xk = x + s

    return xk, counter
```

Método de Newton Modificado

A alteração feita para o método de Newton consiste em tomar a cada iteração k a matriz $J(x^{(0)})$, em vez de $J(x^{(k)})$: a partir de uma aproximação inicial $x^{(0)}$, a sequência $\{x^{(k)}\}$ é gerada através de $x^{(k+1)} = x^{(k)} + s^{(k)}$, onde $s^{(k)}$ é a solução do sistema linear:

$$J(x^{(0)})s = -F(x^{(k)})$$

Desta forma, a matriz Jacobiana é avaliada apenas uma vez e, para todo k, o sistema linear a ser resolvido terá a mesma matriz Jacobiana. A implementação em Python pode ser feita, com poucas alterações para a função anterior, como mostrado abaixo.

In [6]:
```
def newton_modificado(F, J, x0, h, e1=1e-10, e2=1e-4):
    counter = 0
    x = x0
    if abs(np.max(F(x))) < e1:
        return x0

    J_ = J(x,h)
    s = np.linalg.solve(J_,-F(x))
    xk = x + s

    while abs(np.max(F(x))) >= e1: # and abs(np.max(xk-x)) >= e2:
        counter = counter + 1
        x = xk
        s = np.linalg.solve(J_,-F(x))
        xk = x + s

    return xk, counter
```

Questão 1 - Letra a

A seguir a solução do sistema para $h = 10^{-2}$ usando o método de Newton tradicional.

In [7]:
```
x, iter_ = newton(F=F, J=J, x0=np.array([[-1.2, 1]]), h=1e-2)
print("A solução para equação é dada por: ({},{}), e foram precisas {}
```

```
A solução para equação é dada por: (1.0,0.9999999999999325), e foram
precisas 1 iterações.
```

Questão 1 - Letra b

A seguir a solução do sistema para $h = 10^{-5}$ usando o método de Newton tradicional.

In [8]:
```
x, iter_ = newton(F=F, J=J, x0=np.array([[-1.2, 1]]), h=1e-5)
print("A solução para equação é dada por: ({},{}), e foram precisas {}
```

```
A solução para equação é dada por: (0.9999999999999998,1.00000000018
32312), e foram precisas 1 iterações.
```

Questão 1 - Letra c

A seguir a solução do sistema para $h = 10^{-2}$ usando o método de Newton modificado.

In [9]:
```
x, iter_ = newton_modificado(F=F, J=J, x0=np.array([[-1.2, 1]]), h=1e-
print("A solução para equação é dada por: ({},{}), e foram precisas {}
```

```
A solução para equação é dada por: (0.9999999999999997,0.99999999999
97948), e foram precisas 1 iterações.
```

Questão 1 - Letra d

A seguir a solução do sistema para $h = 10^{-5}$ usando o método de Newton modificado.

In [10]:
```
x, iter_ = newton_modificado(F=F, J=J, x0=np.array([[-1.2, 1]]), h=1e-
print("A solução para equação é dada por: ({},{}), e foram precisas {}
```

```
A solução para equação é dada por: (1.0,0.9999999999999984), e foram
precisas 2 iterações.
```

Segunda Questão

A relação entre a pressão (p) o volume (V) e a temperatura (T) de um gás real pode ser dada por uma equação de estado na forma $p = p(V, T)$ ou

$$\varphi(p, V, T) = 0$$

com $\varphi(p, V, T)$ uma função não-linear obtida a partir da equação de estado. Como condição de partida usaremos podemos usar o volume ocupado por uma gás ideal, que é dado por:

$$V_0 = \frac{nRT}{p}$$

Desse modo podemos definir as funções:

$$\varphi(V) = pV^3 - npV^2b + an^2V - an^3b - nRTV^2$$
$$\varphi'(V) = 3pV^2 - 2npVb + n^2a - 2nRTV$$

Assim, dado V_0, a sequência do método de Newton é gerada pela fórmula:

$$V_{k+1} = V_k - \varphi(V_k) \cdot \varphi'(V_k)^{-1}$$

Volume Inicial

O volume inicial, dado por:

$$V_0 = \frac{nRT}{p}$$

É implementado como visto abaixo.

In [11]:
```
def initial_volume(pressure,n_mols,R,temperature):
    return (n_mols * R * temperature)/pressure
```

Van der Walls

A equação de Van der Walls, dada por:

$$\varphi(V) = pV^3 - npV^2b + an^2V - an^3b - nRTV^2$$

E sua derivada, dada por:

$$\varphi'(V) = 3pV^2 - 2npVb + n^2a - 2nRTV$$

São implementadas como visto a seguir.

In [12]:
```
def van_der_walls(pressure, volume, n_mols, a, b, R, temperature):
    return (pressure * (volume ** 3)) - \
           (n_mols * pressure * (volume ** 2) * b) + \
           (a * (n_mols ** 2) * volume) - \
           (a * (n_mols ** 3) * b) - \
           (n_mols * R * temperature * (volume ** 2))
```

In [13]:
```
def derivative_van_der_walls(pressure, volume, n_mols, a, b, R, temper
    return (3 * pressure * (volume ** 2)) - \
           (2 * n_mols * pressure * volume * b) + \
           ((n_mols ** 2) * a) - \
           (2 * n_mols * R * temperature * volume)
```

Método de Newton Escalar

O que o método de Newton, para valores escalares, faz, na tentativa de garantir e acelerar a convergência, é escolher para função de iteração a função $\varphi(x)$ tal que $\varphi'(\xi) = 0$. Então, dada a equação $f(x) = 0$ e partindo da forma geral para $\varphi(x)$, queremos obter $A(x)$ tal que $\varphi'(\xi) = 0$.

Analisando a equação acima, temos que, $A(x) = \frac{-1}{f'(x)}$. E, portanto, a função de iteração do método será dada por:

$$\varphi(x) = x - \frac{f(x)}{f'(x)}$$

Assim, escolhido x_0, a sequência $\{x_k\}$ será determinada por:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

A implementação do método pode ser vista a seguir.

In [14]:
```
def newton_scalar(f, Df, pressure, n_mols, a, b, R, temperature,  e=1e
    xn = initial_volume(pressure=pressure, n_mols=n_mols, R=R, tempera

    for n in range(0,max_iter):
        fxn = f(pressure=pressure, volume=xn, n_mols=n_mols, a=a, b=b,
        if abs(fxn) < e:
            return xn, n
        Dfxn = Df(pressure=pressure, volume=xn, n_mols=n_mols, a=a, b=
        if Dfxn == 0:
            return None, -1
        xn = xn - fxn/Dfxn

    return None, max_iter
```

Questão 2 - Letra a - Parte 1

Calcule o volume de um mol de gás carbônico ($n = 1$, $a = 0.3656$, $b = 4.283 \times 10 - 5$ e $R = 8.3144621 J/K/mol$) para a Terra, onde $T = 288K$ e $p = 101.325Pa$.

In [15]:
```
volume, iter_ = newton_scalar(f=van_der_walls, Df=derivative_van_der_w
print("O volume é dado por: {}, e foram precisas {} iterações.".format
```

```
O volume é dado por: 0.023522234335820087, e foram precisas 3 iteraç
ões.
```

Questão 2 - Letra a - Parte 2

Calcule o volume de um mol de gás carbônico ($n = 1$, $a = 0.3656$, $b = 4.283 \times 10 - 5$ e $R = 8.3144621 J/K/mol$) para Vênus, onde $T = 734K$ e $p = 9200000Pa$.

In [16]:
```
volume, iter_ = newton_scalar(f=van_der_walls, Df=derivative_van_der_w
print("O volume é dado por: {}, e foram precisas {} iterações.".format
```

```
O volume é dado por: 0.0006489880629778, e foram precisas 3 iteraçõe
s.
```

Questão 2 - Letra b

O fator de compressibilidade $\mathbf{Z}$, pode ser definido como:

$$\mathbf{Z} = \frac{V}{V_0} = \frac{V}{\frac{nRT}{p}}$$

Construa gráficos plotando os valores de $\mathbf{Z}$ do gás carbônico na Terra e em Vênus. Tome n = 1 e um intervalo $[0, 1 \times p_{atm}; 10 \times p_{atm}]$, onde p_{atm} é a respectiva pressão atmosférica, subdividido em intervalos regulares de $0, 1 \times p_{atm}$.

Terra

In [17]:
```
p_terra = 9200000
p_terra_vec = np.arange(0.1*p_terra, 10*p_terra, 0.1*p_terra)
```

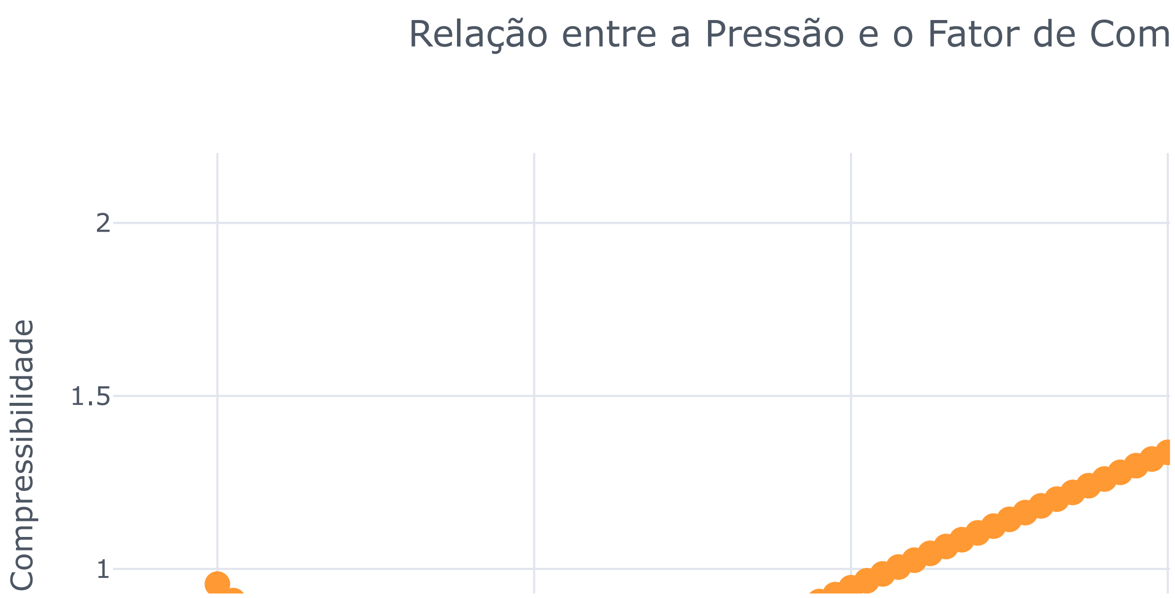
In [18]:
```
volume_terra = [newton_scalar(f=van_der_walls, Df=derivative_van_der_w
```

In [19]:
```
initial_volume_terra = [initial_volume(pressure=p, n_mols=1, R=8.31446
```

In [20]:
```
Z_terra = np.array(volume_terra)/np.array(initial_volume_terra)
```

In [21]:
```
df_terra = pd.DataFrame({'Pressure': p_terra_vec, 'Z': Z_terra})
```

In [22]:
```
df_terra[['Pressure', 'Z']].iplot(
    y='Z', mode='lines+markers',
    xTitle='Pressure', yTitle='Fator de Compressibilidade',
    text='Pressure', title='Relação entre a Pressão e o Fator de Compr
```



Vênus

In [23]:
```
p_venus = 101325
p_venus_vec = np.arange(0.1*p_venus, 10*p_venus, 0.1*p_venus)
```

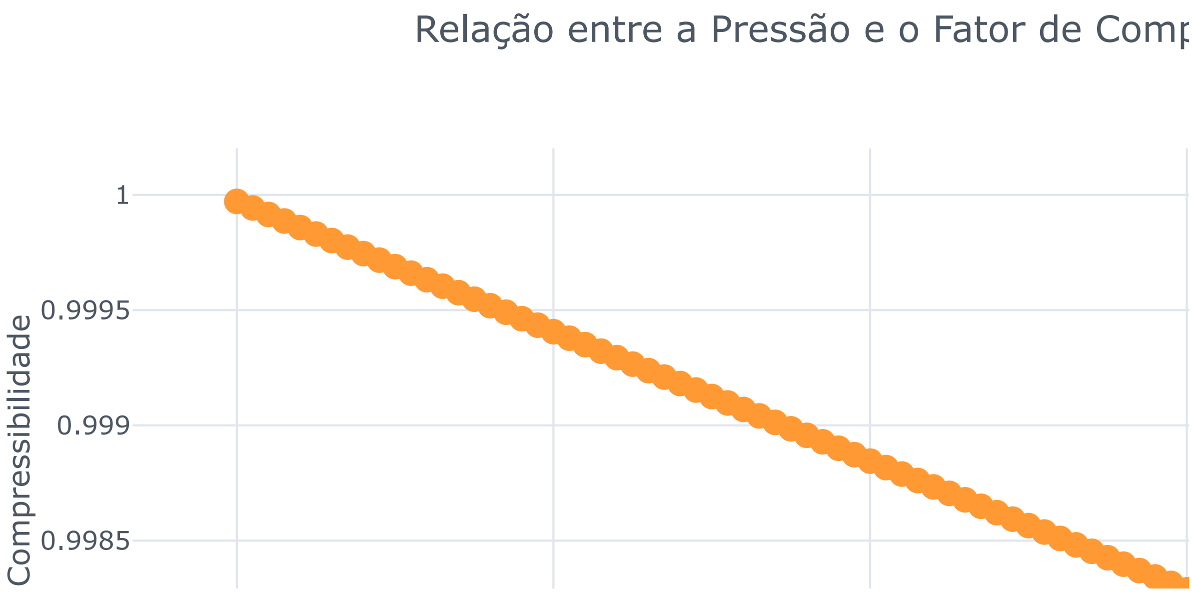
In [24]:
```
volume_venus = [newton_scalar(f=van_der_walls, Df=derivative_van_der_w
```

In [25]:
```
initial_volume_venus = [initial_volume(pressure=p, n_mols=1, R=8.31446
```

In [26]:
```
Z_venus = np.array(volume_venus)/np.array(initial_volume_venus)
```

In [27]:
```
df_venus = pd.DataFrame({'Pressure': p_terra_vec, 'Z': Z_venus})
```

In [28]:
```
df_venus[['Pressure', 'Z']].iplot(
    y='Z', mode='lines+markers',
    xTitle='Pressure', yTitle='Fator de Compressibilidade',
    text='Pressure', title='Relação entre a Pressão e o Fator de Compr
```



In [ ]:

```

```