

Introdução às Instalações Petrolíferas

Revisando ferramentas computacionais

Objetivo: Revisar e apresentar algumas funções e bibliotecas computacionais úteis que serão necessárias ao longo do curso.



Deepnote

O quê é o deepnote?

O [Deepnote](#) é um [bloco de anotações Jupyter hospedado](#) com uma série de recursos para tornar os notebooks melhores para colaboração e mais fáceis de usar para usuários menos técnicos.

Ao ter [colaboração](#), [versionamento](#) e [comentários](#) em tempo real, é uma ótima plataforma para equipes que precisam colaborar no mesmo notebook. Além disso, as equipes podem organizar seus notebook em uma estrutura wiki-like para que tudo seja fácil de encontrar.



Deepnote

O quê é o deepnote?

Nele adiciona uma série de recursos para facilitar o trabalho em notebooks. Por exemplo, permite que você se conecte diretamente a fontes de dados externas (**ex.** Snowflake, Postgres, etc...) e tem um builder gráfico para visualização.

Deepnote tem um nível gratuito e oferece ainda mais vantagens gratuitas para [usuários educacionais](#).

Características

- Compatível com [jupyter notebook](#);
- Linguagem de programação aceitas:
 - Python
 - Julia
 - R
 - Scala
 - Ruby
 - etc.
- Possibilidade de conexão com:
 - data warehouses (AWS, GCP, etc.)
 - bancos de dados (Postgres, MongoDB, etc.)

Características (cont.)

- O agendamento de notebooks (runs) é integrado
- Execução de notebooks em contêineres (*DockerFile*)
- Edição colaborativa, possibilidade de vários editores ao mesmo tempo!
- Totalmente configuráveis, assim podemos realizar qualquer configuração e gerenciar bibliotecas
- **acesso a GPU's e TPU's**, este último apenas no plano pago
- Não depende de máquina local.

Python

Linguagem de programação que apresenta como pontos fortes:

- **Alto nível;**
- **Interpretada** de scripts;
- Tipagem dinâmica e forte;
- **Multiplataforma;**
- Interativa;
- Interoperabilidade com bibliotecas externas escritas em C, Fortran;
- Python pode se tornar mais rápido por meio de **bibliotecas externas**, compiladores JIT (**numba**) e otimizações com ferramentas como o **cython**;

"Duck" Tipagem

A tipagem dinâmica, por ser mais prática torna a linguagem python lenta!, conhecida como **Duck typing**. pois:

“ "Se parece uma pato, anda como pato... é um pato" ”

Para realizar qualquer operação em uma variável, o python realiza muitas operações e lê muitos metadados.

Atalhos importantes

- Ctrl + Espaço - Lista de Funções, variáveis, opções, etc.
- Ctrl + S - Salva o status atual do notebook
- Ctrl + F9 - Roda todo o notebook
- Ctrl + F8 - Roda a célula anterior
- Ctrl + F10 - Roda a próxima célula
- Ctrl + / - Comenta a linha ou linhas selecionadas
- Shift + Alt + Seta - Seleciona em bloco

Principais tipos de variáveis

1. Lista| array
2. Dicionário
3. Dados tabelados (*data frames*)

Lista | array (**Python**)

Basicamente são "**vetores**" que armazenam qualquer tipo de variáveis (int, float, string, etc...)

```
# Define a lista
param = ["Porosidade", "Permeabilidade", "Vazão de óleo", "Vazão de gás", "Influxo"]

# imprime, depois que rodar a célula, apenas o primeiro elemento da lista
param[0]
```

```
# Define a lista
vals = [0.3, 1000, 1.58e6, 8.65e6, 1.06e5]

# imprime, depois que rodar a célula, a lista criada
vals
```

“ **Atenção:** Indexação de vetores no Python inicia em 0 (zero)

Acessando dados dos Dicionários

O acesso aos dados de um dicionário é feito através da identificação indexada

```
# acessar os dados de permeabilidade do campo
print("Permeabilidade: ", campo["propriedades"]["Permeabilidade"])

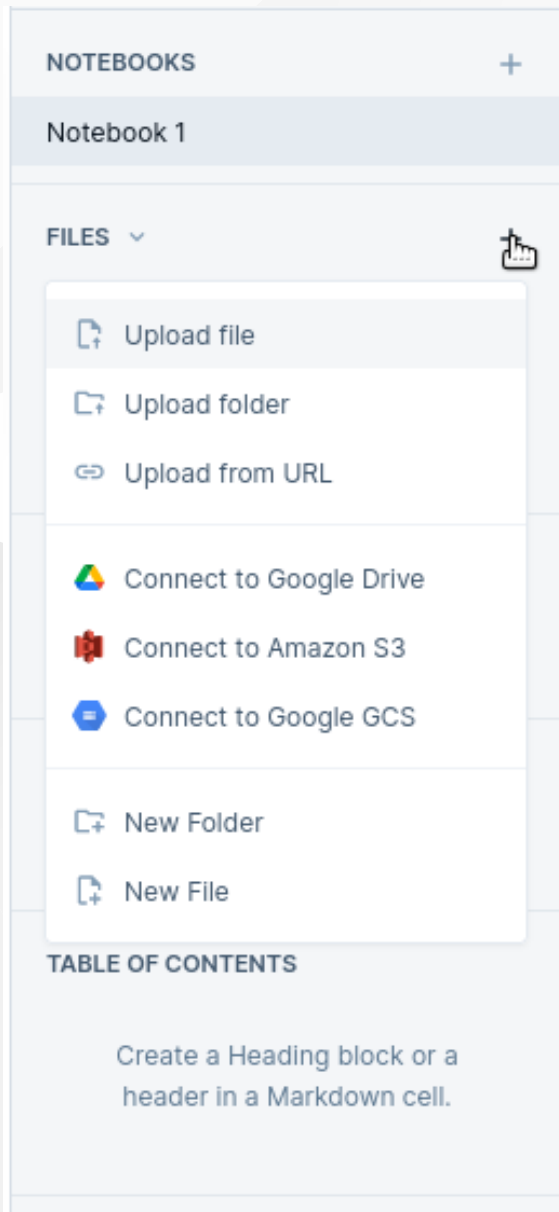
# acessar os a Vazão de óleo do histórico do campo
print("Qo [STB]: ", campo["dados"]["Vazão de óleo"])
```

- O acesso é o mesmo para Julia e Python

	tempo	q
0	0.000000	3.028741e+06
1	21.470588	2.321824e+06
2	42.941176	2.057640e+06
3	64.411765	1.848416e+06
4	85.882353	1.584995e+06
5	107.352941	1.286194e+06
6	128.823529	9.505859e+05
7	150.294118	9.636888e+05
8	171.764706	6.705952e+05
9	193.235294	6.669642e+05
10	214.705882	5.485086e+05
11	236.176471	4.509770e+05
12	257.647059	6.000459e+05
13	279.117647	4.567996e+05
14	300.588235	2.633578e+05

Data frames

É um conjunto de informações dispostos em tabelas, que podem ser analisados, processados, filtrados e adicionar outros dados (também tabelados), similar a uma banco de dados. Pode ser criados a partir de valores separados por vírgula (arquivos .csv), ou a partir de um dicionário. **É necessário** importar a biblioteca `pandas` (para [Python](#)) e `DataFrames` (para [Julia](#)).



Criando Data frame a partir de arquivos csv

Primeiro precisamos subir o arquivo csv para o [deepnote](#) através de **+**, abrindo uma caixa de diálogo para seleção de um arquivo qualquer.


Criando Data frame a partir de arquivos csv II (Python)

Quando o arquivo estiver no diretório de arquivos temporários, basta criar o data frame, através de:

```
# importa a biblioteca pandas
import pandas as pd

# Cria o data frame
df = pd.read_csv("/root/work/data.csv")

# Mostra o data frame ao final do processo
df
```



	x	y
0	0.000000	NaN
1	0.010101	NaN
2	0.020202	NaN
3	0.030303	NaN
4	0.040404	NaN
...
95	0.959596	8.161062
96	0.969697	8.119702
97	0.979798	8.079086
98	0.989899	8.039193
99	1.000000	8.000000

100 rows x 2 columns

Criando Data frame a partir de dicionário (Python)

```
# importa a biblioteca pandas (caso não tenha importado antes)
import pandas as pd

# importante numpy
import numpy as np

# criando aleatoriamente 10 dados entre 100 e 200
x = 100 + 100*np.random.rand(10)
# criando aleatoriamente 10 dados entre 60 e 80
y = 60 + 20*np.random.rand(10)

# Cria o data frame
df = pd.DataFrame.from_dict({'x': x, 'y': y})

# Mostra o data frame ao final do processo
df
```

Acessando dados do Data frame

- Modo 1 (funciona para **Python**)

```
df['x']
```

- Modo 2 (funciona para **Python** e **Julia**)

```
df.x
```


Analizando dados do Data frame (**Python**)

- Descrição geral dos dados

```
df.describe(include='all')
```

- Mostrando apenas os primeiros valores

```
df.head()
```

- Mostrando apenas os últimos valores

```
df.tail()
```

- Mostrando aleatoriamente 5 valores

```
df.sample(5)
```

Modificando dados do Data frame (**Python**)

- Alterando os nomes das colunas

```
df.columns = ['aleatorio1', 'aleatorio2']  
df
```

- Alterando valores

```
df['aleatorio1'] = 0.778 * df['aleatorio1'] + 2.0  
df
```

- Criando novos valores

```
df['aleatorio1_original'] = df['aleatorio1'] / 0.778 - 2.0  
df['aleatorio1xaleatorio1'] = df['aleatorio1'] * df['aleatorio2']  
df
```

Modificando dados do Data frame (**Python**)

- Removendo valores

```
# Para remover definitivamente a coluna, utilizar o inplace=True  
df.drop('aleatorio1', axis='columns', inplace=True)  
df
```

- Criando novos valores nulos

```
df['zeros'] = 0.0  
df
```

Plotagem de dados (**Python**)

A biblioteca `matplotlib` é uma coletânea de comandos e funções, muito parecidas com o `MATLAB`, por simplificação usamos os comandos/funções `inline` usando a macro `%matplotlib inline`, da seguinte forma:

```
# importante biblioteca...  
import matplotlib.pyplot as plt  
%matplotlib inline
```

Gráfico 1D (Python)

A plotagem segue a seguinte sintaxe:

```
“ plt.plot(x, y, linha_marcador_config, label="legenda")  
  plt.xlabel("nome-do-eixo-x")  
  plt.ylabel("nome-do-eixo-y")  
  plt.legend()  
”
```

```
# importante biblioteca...  
import matplotlib.pyplot as plt  
%matplotlib inline  
  
# criando aleatoriamente 10 dados entre 100 e 200  
x = 100 + 100*np.random.rand(10)  
  
plt.plot(x, 5.8 * x + 1 + np.random.normal(0, 3, np.shape(x)), 'o', label="dados aleatórios")  
plt.xlabel("x"); plt.ylabel("y")  
plt.legend()
```

Multiplos gráficos 1D (Python)

```
# importante biblioteca...
import matplotlib.pyplot as plt
%matplotlib inline

# usando funções e arrays do numpy
t = np.arange(0,5,.2)
# adicionar gráfico por gráfico
plt.plot(t,t,'b');
plt.plot(t,t**2, '-.r');
plt.plot(t,t**3, 'g');
plt.plot(t,t**4, '-ok');
# ou inline tudo em uma única chamada
#plt.plot(t,t,'b', t, t**2, 'r', t, t**3, 'g');
plt.legend(['linear', 'quadrática', 'cúbica', 'quarta']);
plt.ylim([0,50]);
```

Gráfico 2D (Python)

A plotagem segue a seguinte sintaxe:

“ plt.contour(x, y, valores, color='cor-da-linha', levels=valores-para-os-contornos)
plt.contourf(x, y, valores)
plt.colorbar()
”

```
import numpy as np

x,y = np.meshgrid(np.arange(-2, 2, .2), np.arange(-2, 2, .25))
z = x*np.exp(-x**2 - y**2)
plt.contourf( x, y, z)
plt.colorbar()
plt.contour(x, y, z, colors = 'k', linewidth=3.0)
```

Gráfico 2D II (Python)

Melhorando o plot

```
# importante biblioteca...
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np

x,y = np.meshgrid(np.arange(-2, 2, .2), np.arange(-2, 2, .25))
z = x*np.exp(-x**2 - y**2)
plt.contourf( x, y, z)
plt.colorbar()
cv = plt.contour(x, y, z, colors = 'k', linewidth=3.0, levels=np.linspace(-2,2, 40))
# Adiciona os valores dos contornos, fig.levels[:,x] controla a frequencia dos valores, menos x mais contornos
plt.clabel(cv, cv.levels[:,2], inline=1, fontsize=10)
```


Outros Matplotlib exemplos (**Python**)

Mais Informações:

- [PyPloy Tutorial](#)
- `plot()` [function reference](#)
- [Samples gallery](#)



SciPy

Biblioteca Scipy

A biblioteca utilizada para a realização de ajustes é o `scipy`. Ele apresenta módulos ou um kit de ferramentas científico (*Scientific Toolkit*) em python, de onde existem uma série de funções de álgebra linear, otimização, zero de funções, "resolvedores" numéricos (*solvers*). Todos estes separados em subpacotes que cobrem diferentes áreas da computação científica.

Módulos do Scipy

- Funções especiais (`scipy.special`)
- Otimização (`scipy.optimize`)
- Integrações (`scipy.integrate`)
- Interpolações (`scipy.interpolate`)
- Álgebra linear (`scipy.linalg`)
- Estatística (`scipy.stats`)
- Processamento de imagem (`scipy.ndimage`)
- Funções de entrada/saída de arquivos (`scipy.io`)
- E muito mais...

Zero de funções não-lineares (Python)

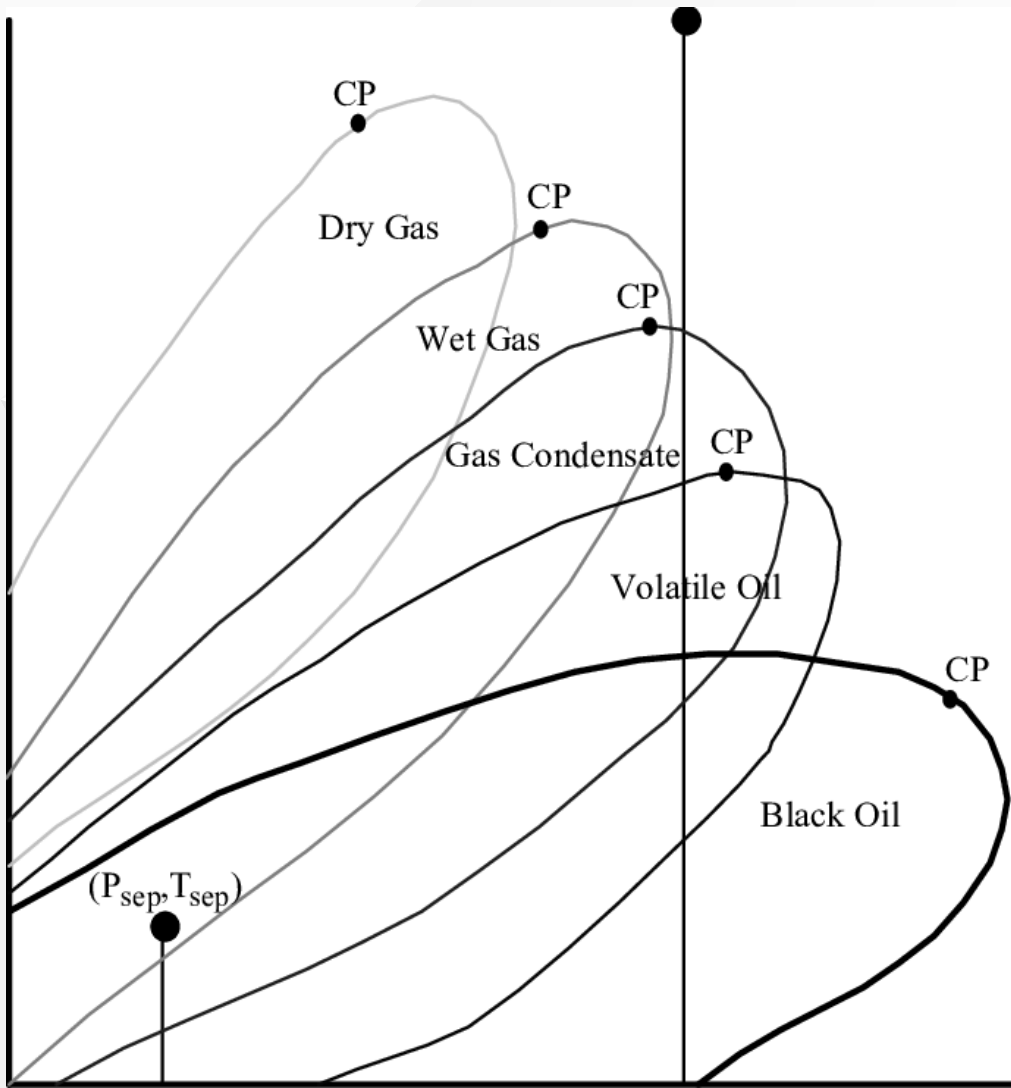
Há varios métodos de encontrar o zero (raiz) usando o `scipy` ou até mesmo o `numpy`. Dentre os métodos que podemos listar são:

- Secante
- Bisseccção
- Newton

Estes métodos, como muitos outros, estão implementados nas funções `fsolve` e o `root`, do módulo `optimize`

Equação de estado (EOS)

Uma equação de estado é uma **equação termodinâmica** que descreve o estado da matéria sob um dado conjunto de condições físicas (PVT). É uma equação constitutiva que provê uma relação matemática entre duas ou mais funções de estado associadas com a matéria, tais como sua temperatura, pressão, volume, energia interna ou entropia. Equações de estado são úteis para a descrição das propriedades de fluidos e de misturas de fluidos (Propriedades dos HC).



EOS de Peng-Robinson

Na engenharia de reservatórios precisamos entender o comportamento dos fluidos (água, gás e óleo), nas diferentes condições de reservatórios e de superfície.

A equação de estado mais utilizada é a EOS de Peng-Robinson (1976), que nada mais é que uma modificação da equação de Van der Waals:

$$p = \frac{RT}{\hat{v} - b_{PR}} - \frac{a_{PR}}{\hat{v}(\hat{v} + b_{PR}) + b_{PR}(\hat{v} - b_{PR})}$$

onde: $a_{PR} = \sum_{i,j=1}^{n_c} y_i y_j a_{ij}$ e $b_{PR} = \sum_{i=1}^{n_c} y_i b_i$

com,

$$a_{ii} = 0.45724 \left(\frac{R^2 T_{ci}^2}{p_{ci}} \right) \alpha_i, \quad a_{ij} = a_{ji} = \sqrt{a_{ii} a_{jj}} (1 - k_{ij}), \quad \alpha_i = [1 + m_i (1 - \sqrt{T_{ri}})]^2_{30}$$

EOS de Peng-Robinson solução

A EOS de Peng-Robinson é uma variação apropriada para moléculas que são assimétricas, típica de mistura de HC e é uma variação importante para a simulação do equilíbrio vapor-líquido e o diagrama de fases.

Quando a temperatura e a pressão de uma mistura gasosa e os parâmetros a_{PR} e b_{PR} são dados, então para encontrar o volume específico você teria que resolver o EOS cúbica para o volume específico, \hat{v} . Isso representa uma equação algébrica em uma incógnita, o volume específico. A EOS Peng-Robinson resulta em:

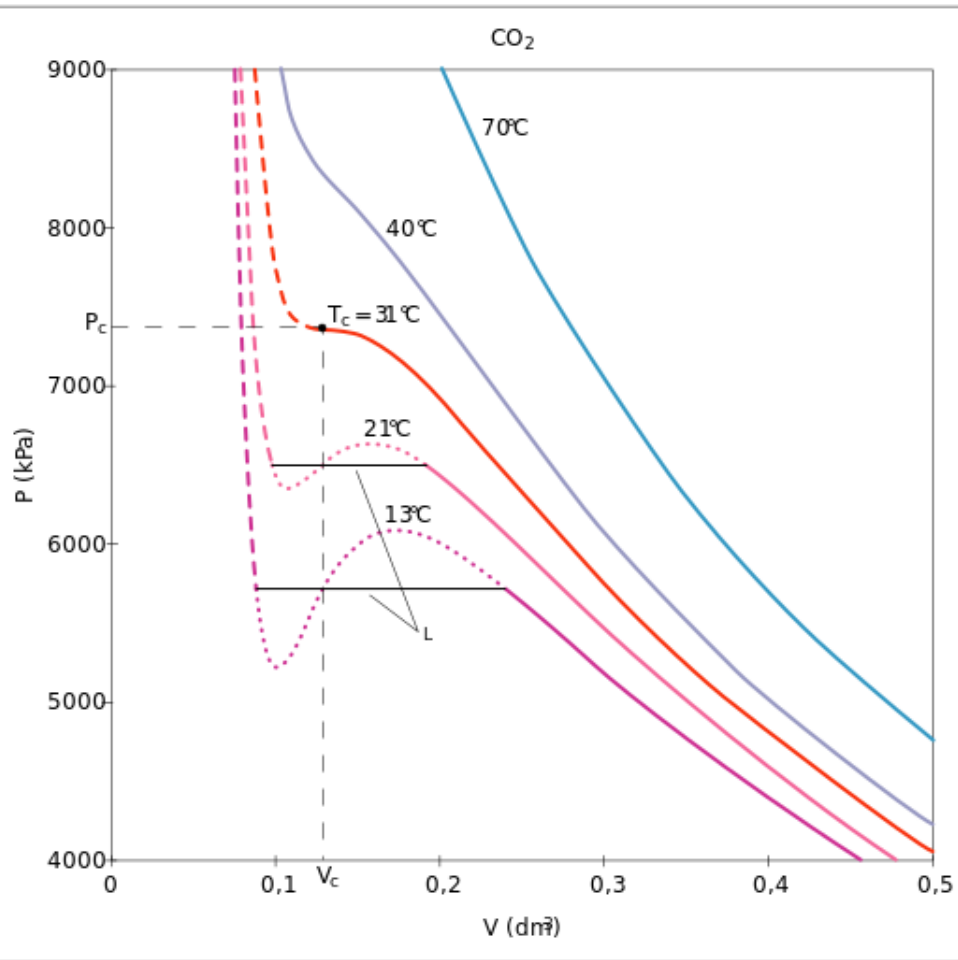
$$p\hat{v}^3 + (b_{PR}p - RT)\hat{v}^2 + (a_{PR} - 3pb_{PR}^2 - 2RTb_{PR})\hat{v} + (pb_{PR}^3 + RTb_{PR}^2 - a_{PR}b_{PR}) = 0$$

Com a resolução desta equação cúbica, podemos determinar quaisquer propriedades do fluido ou mistura.

EOS de Peng-Robinson algoritmo

Passo#1: Primeiro, você precisa preparar uma função que calculará um $f(x)$, neste caso $f(\hat{v})$, para uma dada temperatura, pressão e propriedades termodinâmicas, irá calcular o b_{PR} e a_{PR} e retornar a EOS cúbica. Poderás chama-la de "volume_espec".

Passo#2: Teste! a função "volume_espec", para emitir um resultado. **SEMPRE É BOM FAZER TESTES NO DECORRER DA IMPLEMENTAÇÃO!!!**, pois é para isso que serve o notebook.



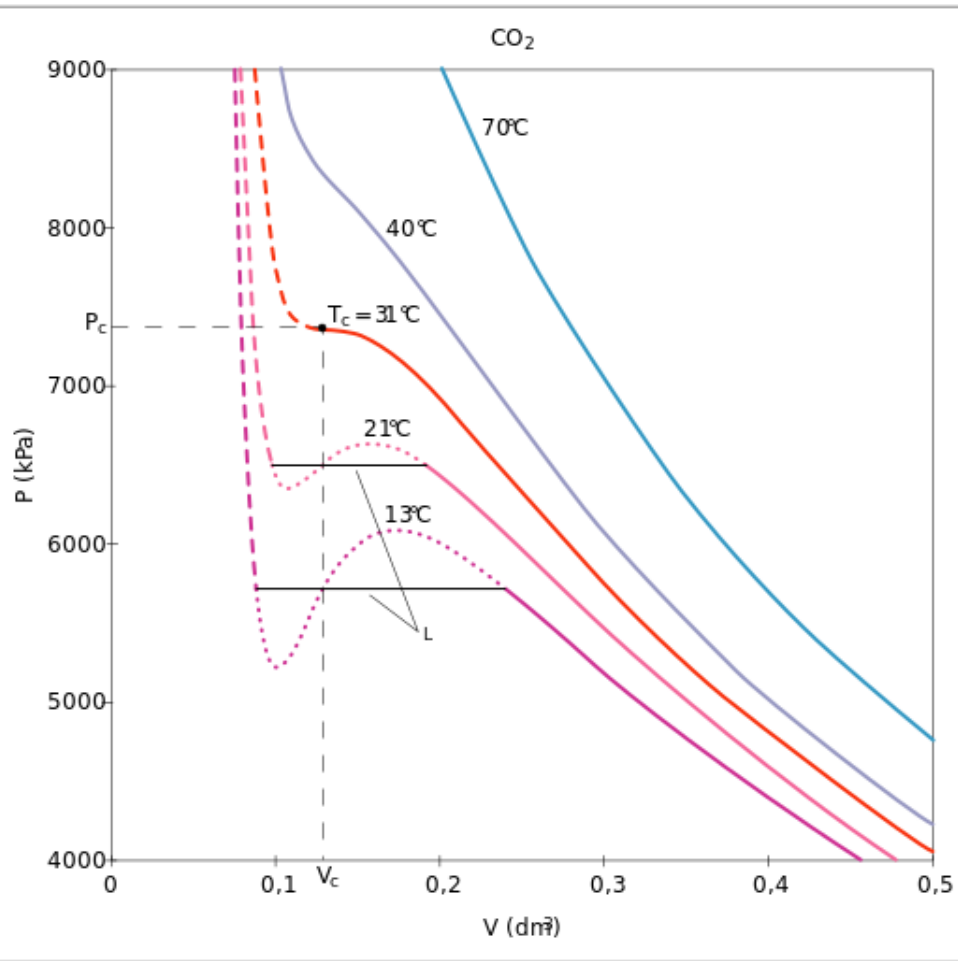
EOS de Peng-Robinson algoritmo

Passo#3: Quando você usa “fzero”, a função “volume_espec” será avaliada para uma variedade de \hat{v} , cujo o valor retornado é o volume específico da mistura de HC nas condições avaliadas.

```
v = fzero(volume_espec, 2)
```

```
v = nlsolve(volume_espec!, [2.0])
```

Com isso, você poderá plotar $p \times Z$ para a mistura de HC.



Ajuste de curvas

Dado um conjunto de dados composto por um grupo de pontos (ex.: um histórico de produção), queremos encontrar o melhor ajuste que represente este conjunto de dados.

Muitas vezes temos um conjunto de dados composto por dados seguindo uma tendência ou uma expressão não-linear conhecida, mas cada dado tem um desvio padrão que os torna dispersos pela linha de melhor ajuste. Podemos obter uma única linha usando a função

`curve-fit()`, usando `SciPy`.

Ajustando curvas (linear e não-linear)

Para esta aplicação, precisamos de dados de teste para implementar o ajuste de curva e podemos definir um simples x , como input, e y , como output, de dados. Você pode aplicar o mesmo método para seus dados de resposta.

Em seguida, definiremos várias funções para usar na função 'curve_fit' e verificaremos suas diferenças no ajuste. Você também pode adicionar ou alterar as equações para obter os melhores parâmetros de ajuste para seus dados.

Usamos as equações abaixo como as funções de ajuste:

$$y = ax^2 + bx + c, \quad y = ax^3 + bx + c, \quad y = ax^2 + bx^2 + c, \quad y = ae^{bx} + c$$

Com as equações definidas, escrevemos em python a seguir.

Ajustando curvas (código 1/2) (Python)

```
from scipy.optimize import curve_fit

# dados
y = np.array([12, 11, 13, 15, 16, 16, 15, 14, 15, 12, 11, 12, 8, 10, 9, 7, 6])
x = np.array(range(len(y)))

# funções de ajuste
def func1(x, a, b, c):
    return a*x**2+b*x+c

def func2(x, a, b, c):
    return a*x**3+b*x+c

def func3(x, a, b, c):
    return a*x**3+b*x**2+c

def func4(x, a, b, c):
    return a*np.exp(b*x)+c
```

Ajustando curvas (código 2/2) (Python)

```
# Ajuste
params, _ = curve_fit(func1, x, y)
a, b, c = params[0], params[1], params[2]
yfit1 = a*x**2+b*x+c

params, _ = curve_fit(func2, x, y)
a, b, c = params[0], params[1], params[2]
yfit2 = a*x**3+b*x+c

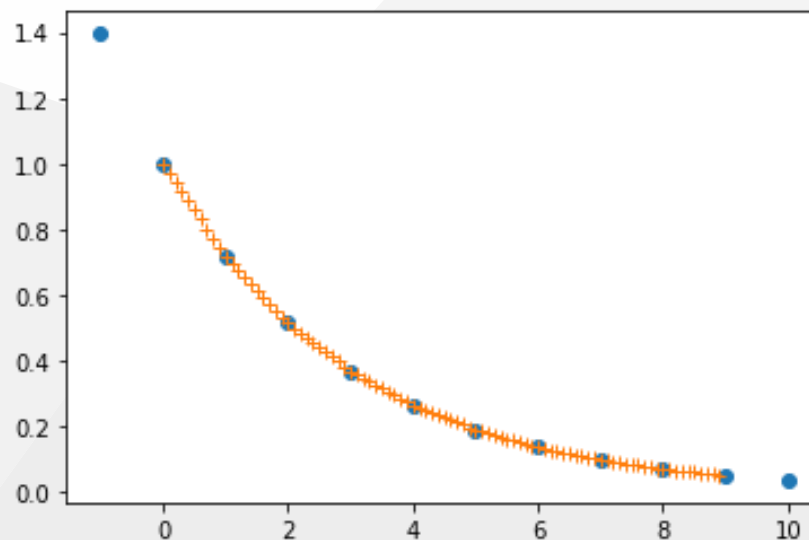
params, _ = curve_fit(func3, x, y)
a, b, c = params[0], params[1], params[2]
yfit3 = a*x**3+b*x**2+c

params, _ = curve_fit(func4, x, y)
a, b, c = params[0], params[1], params[2]
yfit4 = a*np.exp(x*b)+c

# visualizando os resultados...
plt.plot(x, y, 'bo', label="dados")
plt.plot(x, yfit1, label="y=a*x^2+b*x+c")
plt.plot(x, yfit2, label="y=a*x^3+b*x+c")
plt.plot(x, yfit3, label="y=a*x^3+b*x^2*c")
plt.plot(x, yfit4, label="y=a*exp(b*x)+c")
plt.xlabel('x')
plt.ylabel('y')
plt.legend(loc='best', fancybox=True, shadow=True)
plt.grid(True)
plt.show()
```

Interpolação de funções

A **interpolação** consiste em determinar, a partir de um conjunto de **dados discretos**, uma **função** ou um conjunto de **funções** analíticas que possam servir para a determinação de qualquer valor no domínio de definição. Pode-se ver a interpolação como um processo numérico que mapeia uma função discreta para uma função contínua onde não sabemos a expressão!.



Interpolação de funções (código) (Python)

```
import numpy as np
from scipy import interpolate

x = np.arange(-1, 11)
y = np.exp(-x/3.)
f = interpolate.interp1d(x, y);

xnew = np.arange(0, 9, 0.1)
ynew = f(xnew)    # usamos a função de interpolação advinda de interp1d
plt.plot(x, y, 'o', xnew, ynew, '+');
```

Funções Especiais

Como o próprio nome diz... neste módulo encontraremos diversas funções especiais e amplamente utilizadas nas soluções analíticas de problemas de fluxo em meio porosos:

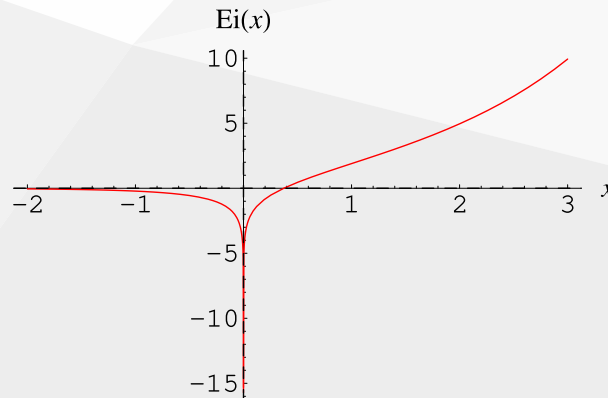
- Função exponencial integral (E_i)
- Função de erro (**erf**)

Função exponencial integral

Esta é uma função especial, que é definida por:

$$E_i(x) = \int_x^{\infty} \frac{e^{-\xi}}{\xi} d\xi$$

e apresenta a seguinte forma:

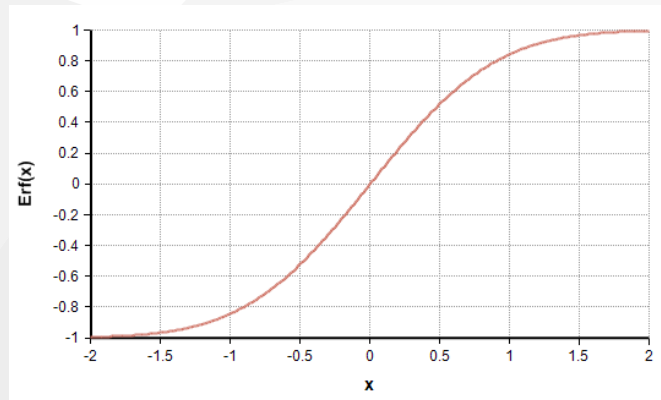


Função de erro (erf)

Outra função especial, que apresenta a seguinte definição:

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

Com o seguinte comportamento:

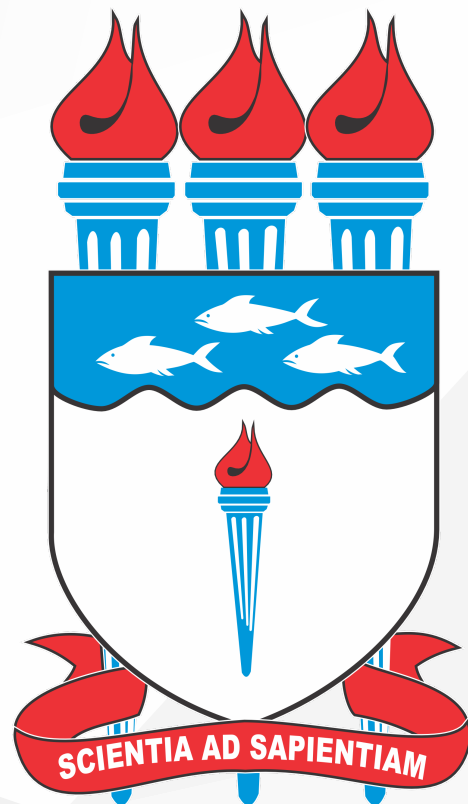


Funções Especiais (código) (**Python**)

```
import scipy.special as sc
x = np.linspace(-3, 3)

sc.expi(x)

plt.plot(x, sc.expi(x))
plt.xlabel('$x$');
plt.ylabel('$E_i(x)$');
plt.grid();
plt.show();
```



Obrigado pela atenção!!!