

# Curso de Controle Clássico

Prof: Dennis

Aula Prática #0 I C – Simulação de circuito RC com toolbox de controle do python.

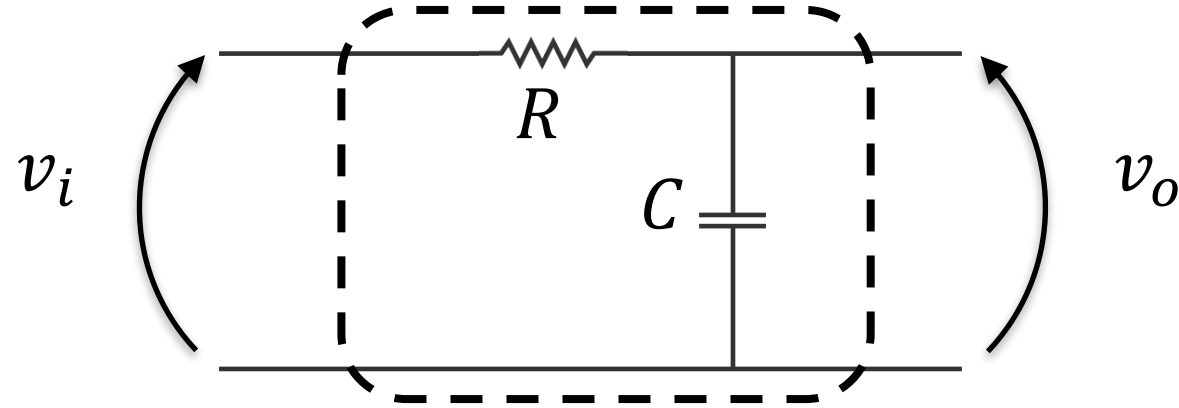
# O que faremos nesta aula

- Nesta aula continuaremos um software gratuito para resolver um exemplo de resposta em malha aberta;
- No entanto como alternativa ao Scilab usaremos o python e a toolbox “control”.
- Existem diversos compiladores python, pagos e gratuitos, inclusive online, que lhe permitirão realizar essas simulações mesmo de um smartfone ou tablet.
- Por simplicidade usaremos uma ferramenta web, o “google colab”;
- Uma vantagem de usar o python, é que se trata de uma linguagem com muito mais bibliotecas e possibilidades do que o Scilab.

# O que faremos nesta aula

- O exemplo escolhido é um circuito RC;
- Nas aulas anteriores, vimos como obter a resposta ao degrau:
  - Graficamente, montando o diagrama do sistema no Xcos do Scilab, como faríamos no simulink do Matlab;
  - Diretamente usando um código na linguagem do Scilab;
- Veremos nesta aula como usar um código em python para fazer a mesma simulação.
- Veremos como instalar a toolbox de controle e como fazer a simulação;
- Novamente obteremos o mesmo resultado, que os dois métodos usando o Scilab, ficando a escolha do estudante qual dos três deseja usar;

Como vimos nas aulas anteriores, resposta a partir do modelo do sistema



$$v_o = v_C = v_i - v_R$$

$$v_R = Ri_C = RC \frac{dv_C}{dt}$$

$$v_o = v_i - RC \frac{dv_o}{dt}$$

$$v_i = RC \frac{dv_o}{dt} + v_o$$

$\mathcal{L}$

$$V_i = RCsV_o + V_o$$

$$\frac{V_o}{V_i} = \frac{1}{1 + RCs}$$

Função de transferência  
do circuito RC com saída  
no capacitor

# Resposta a partir do modelo

- Exemplo: circuito RC em malha aberta:
  - Entrada domínio t: função degrau unitário;
  - Entrada domínio s:  $V_i = \frac{1}{s}$
  - Modelo domínio s:  $\frac{V_o}{V_i} = \frac{1}{1 + RCs}$
  - Saída domínio s:  $V_o = V_i \cdot \frac{V_o}{V_i} = \frac{1}{s(1 + RCs)}$
  - Saída domínio t:  $v_o = 1 - e^{t/RC}$

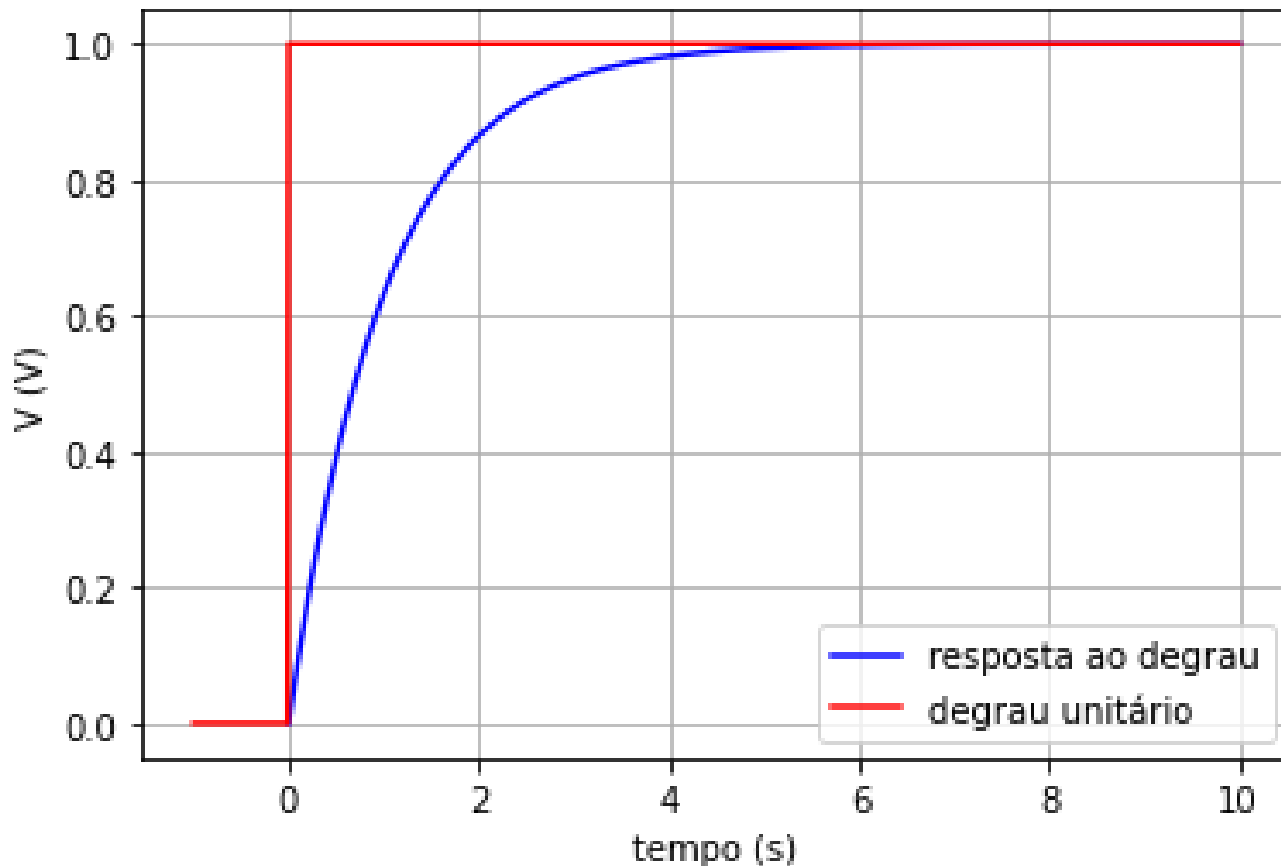
Lembrando que esta teoria está explicada nos vídeos de Controle Clássico, o objetivo desta aula é partir deste resultado e fazer a simulação;

# Biblioteca “control” do python

- É uma toolbox desenvolvida para utilizar as ferramentas numéricas do python, como o “numpy” e o “scipy” e de gráficos como o “matplotlib” aplicados a sistemas de controle;
- Seu site é: <https://python-control.readthedocs.io/en/0.9.0/>
- Deve ser instalada antes de ser usada:
- Sugestão use o comando: **pip install control**

# Código do python

- A direita o código comentado:
- O resultado é:



```
import control as ctl
import matplotlib.pyplot as plt
import numpy as np
# cria a função de transferência em malha aberta
R = 1.;#resistor
C=1.;# capacitor
Tsim=10;
numerador = [1.]
denominador = [R*C,1.]
H = ctl.tf(numerador, denominador)
print(' FT malha aberta= ',H)
#calcula a resposta ao degrau
T, yout = ctl.step_response(H, Tsim)
plt.plot(T,yout,'b-')
#calcula um degrau unitário
T2=np.linspace(-1.,10.,1000)
degrau=np.ones_like(T2)
degrau[T2<0]=0;
plt.plot(T2,degrau,'r-')
plt.ylabel('V (V)')
plt.xlabel('tempo (s)')
plt.legend(['resposta ao degrau','degrau unitário'])
plt.grid()
```

# Comandos importantes: *ctl.tf* (ou *control.tf*)

- Para definir um sistema linear em python, usamos o comando “*control.tf*”;

- Exemplo:

$$H = \text{control.tf}([2., 1.], [1., 4., -3.]) \longrightarrow \frac{1 + 2s}{-3 + 4s + 1s^2}$$

- Os vetores  $[2., 1.]$  e  $[1., 4., -3.]$  são usados para definir os polinômios do numerador e denominador em potência decrescente (o número à esquerda é a maior potência de  $s$ );
- Exemplo do nosso código:

$$\text{numerador} = [1.] \longrightarrow 1$$

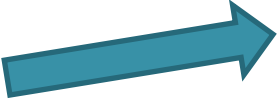
$$\text{denominador} = [1., R * C] \longrightarrow 1 + R * Cs \longrightarrow 1 + 1s$$

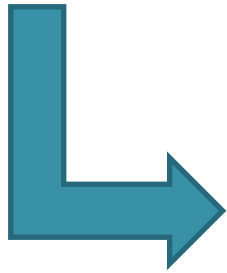
$$H = \text{ctl.tf}(\text{numerador}, \text{denominador}) \longrightarrow \frac{1}{1 + s}$$



# Comandos importantes: `ctl.step_response` (ou `control.step_response`)

- “`ctl.step_response`” Calcula a resposta ao degrau a partir de uma função de transferência;
- Exemplo do nosso código:

`Tsim=10;`  *número que define o tempo final da simulação;*  
`T, yout = ctl.step_response(H, Tsim)`



- *yout: Resposta ao degrau da função de transferência  $H = \frac{1}{1 + s}$  de 0 até 10s;*
- *T vetor de tempos correspondentes aos valores da resposta yout;*

# Entendendo o Código em python

```
import control as ctl
```

```
import matplotlib.pyplot as plt
```

```
import numpy as np
```

```
# cria a função de transferência em malha aberta → comentário
```

```
R = 1. ; #resistor
```

```
C=1. ; # capacitor
```

```
Tsim=10;
```

```
numerador = [1.]
```

```
denominador = [R*C,1.]
```

```
H = ctl.tf(numerador, denominador) → usa os vetores para criar  $H = \frac{1}{1+RCs}$ 
```

```
print(' FT malha aberta= ',H) → Mostra a função de transferência
```

```
#calcula a resposta ao degrau
```

```
T, yout = ctl.step_response(H, Tsim) → calcula a resposta ao degrau (step) para Tsim e H definidos acima
```

```
plt.plot(T,yout,'b-') → Plota o a resposta ao degrau (pode parar aqui, se quiser)
```

```
#calcula um degrau unitário
```

```
T2=np.linspace(-1.,10.,1000)
```

```
degrau=np.ones_like(T2)
```

```
degrau[T2<0]=0;
```

```
plt.plot(T2,degrau,'r-')
```

```
plt.ylabel('V (V)')
```

```
plt.xlabel('tempo (s)')
```

```
plt.legend(['resposta ao degrau','degrau unitário'])
```

```
plt.grid()
```

→ importa as bibliotecas (toolbox) necessárias

→ cria as variáveis R e C e dá valor a elas

→ cria os vetores que serão os polinômios: "1" e "RCs + 1"

→ usa os vetores para criar  $H = \frac{1}{1+RCs}$

→ Mostra a função de transferência

→ calcula a resposta ao degrau (step) para Tsim e H definidos acima

→ Plota o a resposta ao degrau (pode parar aqui, se quiser)

→ calcula e plota um degrau unitário,  
de -1 até 10, para comparar com a saída

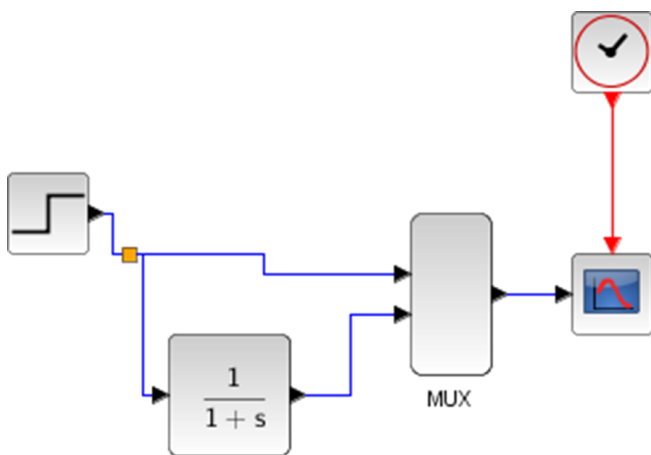
→ coloca títulos nos eixos, uma legenda e um grid  
para deixar o gráfico mais apresentável

# python no navegador com o google colab

- Usaremos o google colab para rodar nosso código em python;
- Link: <https://colab.research.google.com/>
- Ele pode ser executado em qualquer navegador, desde que o usuário faça login com sua conta do google;
- Ao final é possível salvar os códigos no google drive e continuar um outro momento;
- Vamos ver como isso funciona na prática:

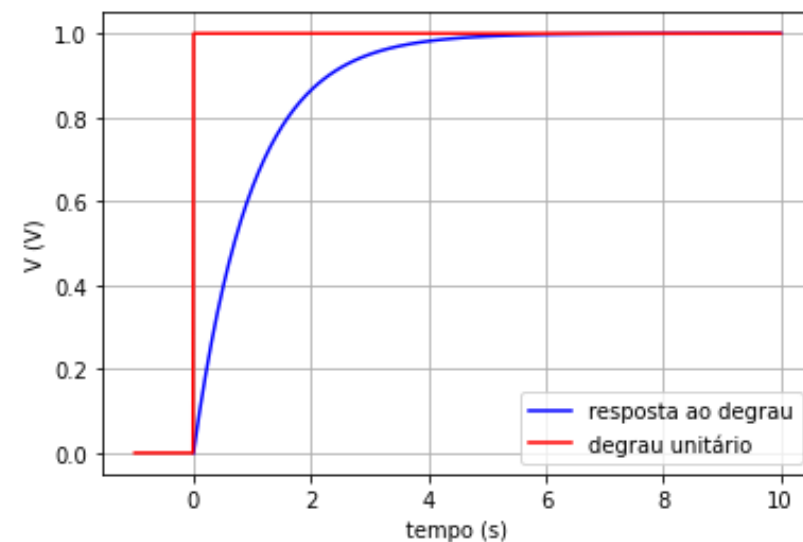
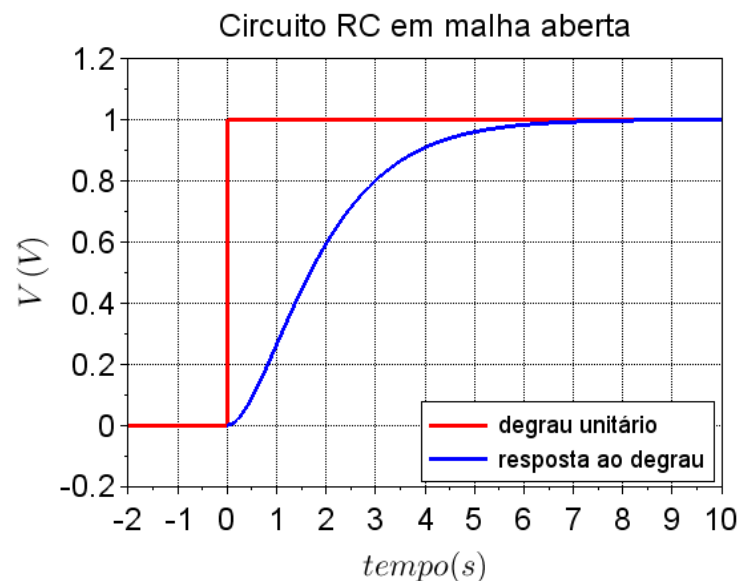
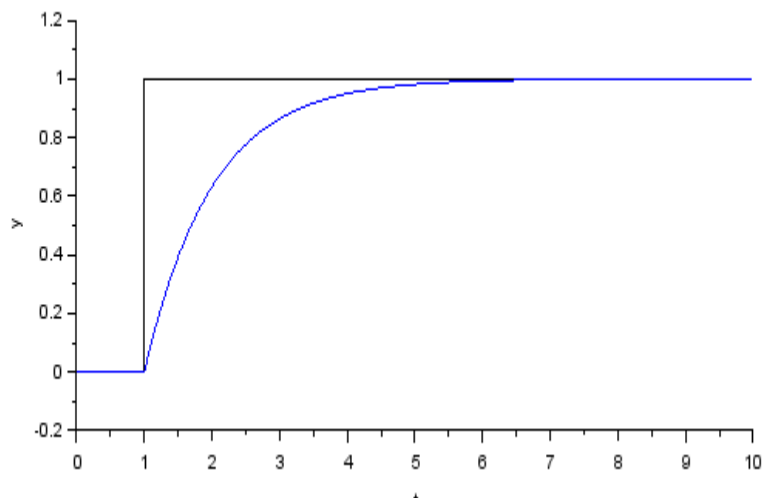
# Comparando Xcos, com scilab, com python:

- Note que o resultado é mesmo. São três caminhos possíveis. Então, qual deles lhe parece mais adequado?



```
// circuito RC em malha aberta
// valores:
R=1;
C=1;
// cria a função de transferência em malha aberta
numerador = poly([1], 's', 'c');
....
```

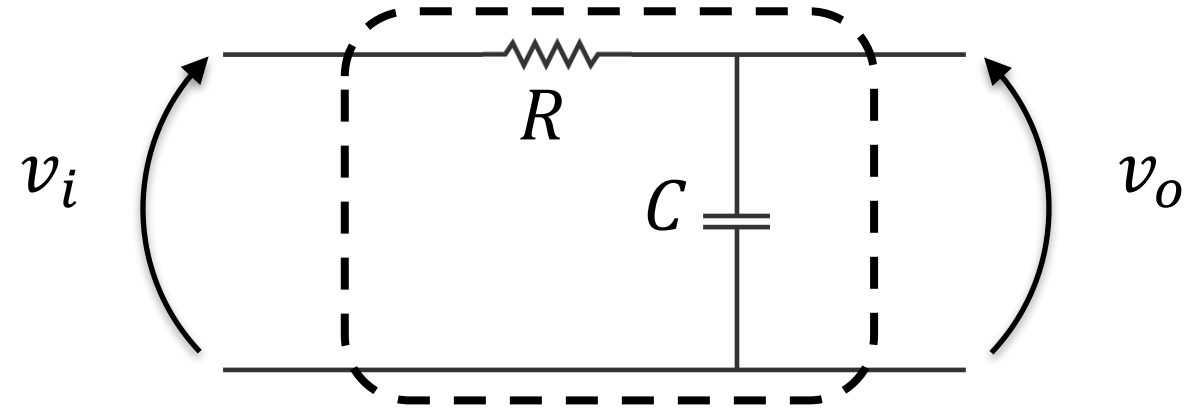
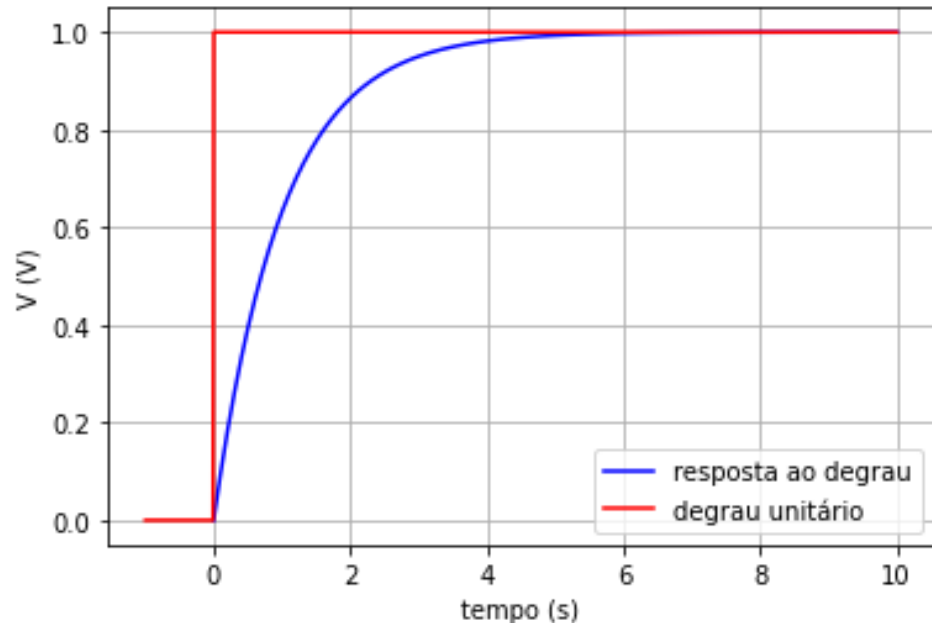
```
import control as ctl
import matplotlib.pyplot as plt
import numpy as np
# cria a função de transferência em malha aberta
R = 1.;#resistor
C=1.;# capacitor
Tsim=10;
numerador = [1.]
...
```



```

import control as ctl
import matplotlib.pyplot as plt
import numpy as np
# cria a função de transferência em malha aberta
R = 1.;#resistor
C=1.;# capacitor
Tsim=10;
numerador = [1.]
...

```



**Controle na prática #0 I C:**  
**Simulação de circuito RC**  
**com toolbox de controle do**  
**python.**