EDA/CAD para Nanoelectrónica

2º Semestre 2020 / 2021

Relatório

Lab#1: Estudo e Aplicação do Modelo *NPower*

25/04/2021

Número de Aluno	Nome	Turno Prático
50726	Francisco Mendes Micaelo André	P1
51005	José Maria Corrêa Arouca Cortes Tamagnini	P1
43853	Rafael Gonçalves Feio de Oliveira	P1

Docente: Maria Helena Fino

Índice

Índice de Figuras

Índice de Tabelas

Introdução

No âmbito da cadeira de EDA/CAD para Nanoelectrónica, é apresentado o primeiro trabalho laboratorial onde é realizado o estudo do modelo *NPower* para a caracterização de transístores MOSFETs de tecnologia nanométrica [1].

Este trabalho está dividido em várias etapas de desenvolvimento, de modo a estudar e comparar o modelo *NPower* com o modelo tradicional de *Shockley*, comumente conhecido como modelo quadrático, dada a variação quadrática da corrente de "dreno", I_D, com a tensão entre a "gate" e a "source", V_{GS}, na zona de saturação, verificando quais as limitações de ambos os modelos.

Na primeira parte do projeto é realizada a descrição do modelo *NPower*, que permite determinar os parâmetros de funcionamento do circuito através de equações de variáveis simples, minimizado os erros associados ao cálculo das variáveis de decisão do modelo. Após a descrição do modelo teórico utilizado (*NPower*), são retiradas as características das correntes I_D, dependendo da tensão V_{GS} e V_{DS}, através do software *LTspice*.

Na segunda parte do trabalho é feita a implementação do modelo *NPower*, obtendo as características do modelo, calculando o erro relativo entre os dados teóricos e simulados, utilizando o software *Spyder*. Por último foram retiradas as conclusões da realização do trabalho prático, com base nos dados obtidos através da abordagem teórica utilizada.

I-Objetivo

Este projeto tem como objetivo a determinação dos parâmetros de funcionamento dos transístores MOSFET de 90nm para uma relação comprimento e largura igual ($W = L$), verificando quais as limitações do modelo *NPower*. A Tabela 1 apresenta as especificações dos transístores utilizados para aplicação da respetiva metodologia.

Tabela 1 – Transístores utilizados

	Modelo	$\boldsymbol{W}$ $(\boldsymbol{nm})$	$\boldsymbol{L}$ $(\boldsymbol{nm})$
NMOS4	90nm_NMOS_bulkL70n	70	70
NMOS4	90nm_NMOS_bulkL700n	700	700

O modelo de *Shockley* torna-se uma aproximação imprecisa para descrição do funcionamento de dispositivos MOSFET de dimensões nanométricas. Deste modo, este trabalho apresenta os desafios da utilização do modelo *NPower*.

II - Apresentação do Modelo *NPower*

2.1. Descrição do Modelo

O modelo *NPower* é descrito pelo sistema equações (1), onde os parâmetros I_D e I_{DSAT} representam as correntes no dreno e de saturação nos transístores MOSFET. A tensão de saturação no "dreno" é representada por V_{DSAT}, enquanto V_{GS} e V_{DS} descrevem a tensão de entre a "gate-source" e "dreno-source" respetivamente e V_T a tensão de "threshold". As seguintes equações dependem das características físicas da tecnologia utilizada, como a largura do canal W e o seu comprimento de canal L_{eff}. Os parâmetros K e m controlam as características da região linear (tensão linear de saturação), enquanto o B e n determinam as características da região de saturação. Por último, a variável λ expressão uma relação entre a condutância do "dreno" na zona linear de saturação.

$$I_D = \begin{cases} 0, & V_{GS} \leq V_T \\ I_{DSAT}(1 + \lambda V_{DS})\left(2 - \frac{V_{DS}}{V_{DSAT}}\right)\frac{V_{DS}}{V_{DSAT}}, & V_{GS} \geq V_T \wedge V_{DS} \leq V_{DSAT} \\ I_{DSAT}(1 + \lambda V_{DS}), & V_{GS} \geq V_T \wedge V_{DS} \geq V_{DSAT} \end{cases} \tag{1}$$

Onde:

$$V_{DSAT} = K(V_{GS} - V_T)^m \tag{2}$$

$$I_{DSAT} = B\frac{W}{L_{eff}}(V_{GS} - V_T)^n \tag{3}$$

III - Metodologia

3.1. LTspice

Na primeira etapa do projeto foi utilizado o programa LTspice para simular o ponto de funcionamento do circuito CMOS apresentado na Figura 1, tendo em conta dois cenários de simulação distintos.

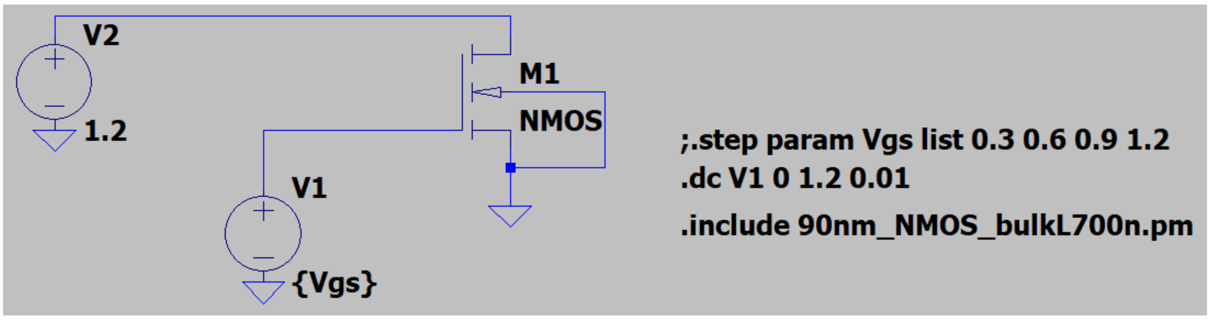


Figura 1 Esquemático para NMOS 700n

O primeiro cenário de simulação corresponde à obtenção da corrente $I_D(V_{DS})$ para valores de V_{GS} fixos onde $V_{GS} = \{0{,}3; 0{,}6; 0{,}9; 1{,}2\}\,V$. Assim, recorrendo ao circuito da figura 1, foi utilizado transístor NMOS de 90nm com W=L=700nm obtendo os seguintes resultados:

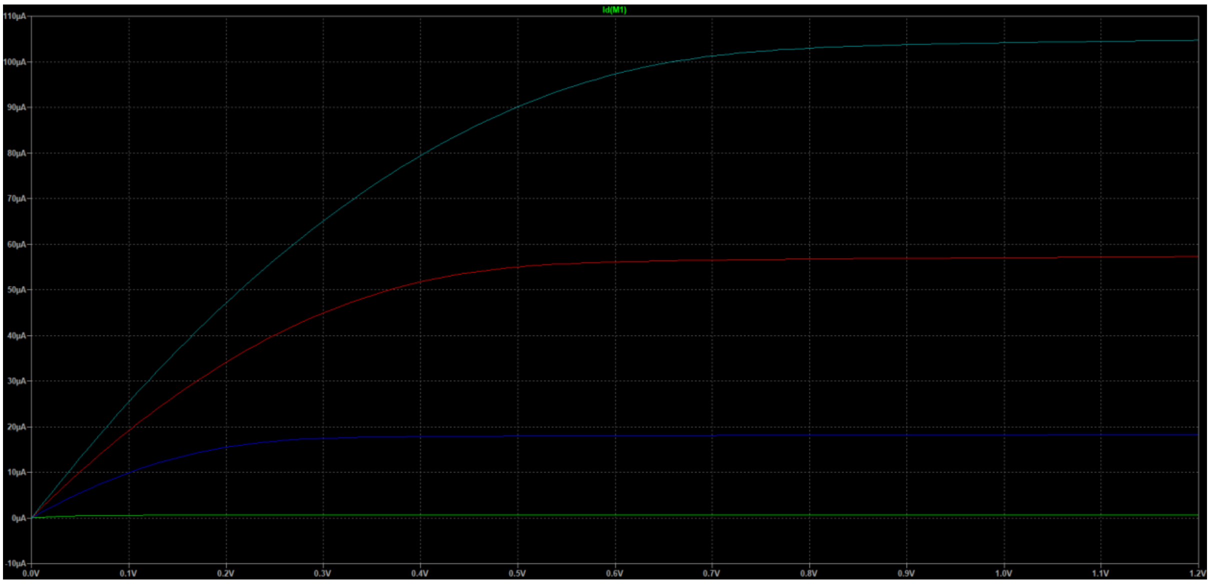


Figura 2 Características I_D em função de V_{DS}, NMOS com W=L=700 nm

Na Figura 2 podemos observar que a característica de I_D tem tendência a com o crescimento de V_{GS} variar entre 0.6μA e 105μA.

De seguida, foi repetida a simulação para um NMOS de 90nm para W=L=70nm. Na Figura 3 observamos que tal como na figura 2, a corrente I_D tende a estabilizar com o aumento de V_{DS} no entanto a amplitude de correntes é menor, variando de 2µA a 67µA

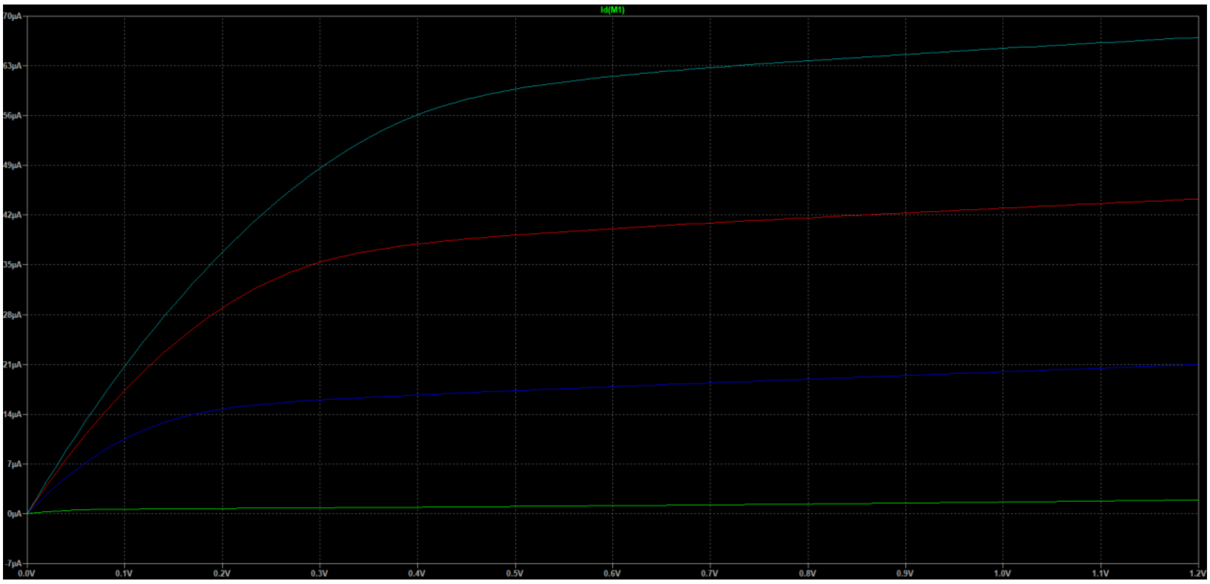


Figura 3-Características de I_D em função de V_{DS}, NMOS com W=L= 70 nm

O segundo cenário de simulação corresponde à obtenção da corrente $I_D(V_{GS})$ para um valor fixo de V_{DS} igual a 1.2V, para os dois circuitos NMOS de 90nm. Os resultados obtidos da simulação do circuito para uma relação W=L=700nm, são apresentados na Figura 4. Podemos verificar que com o aumento da tensão V_{GS} a corrente I_D irá aumentar exponencialmente, atingindo um máximo em 105µA.

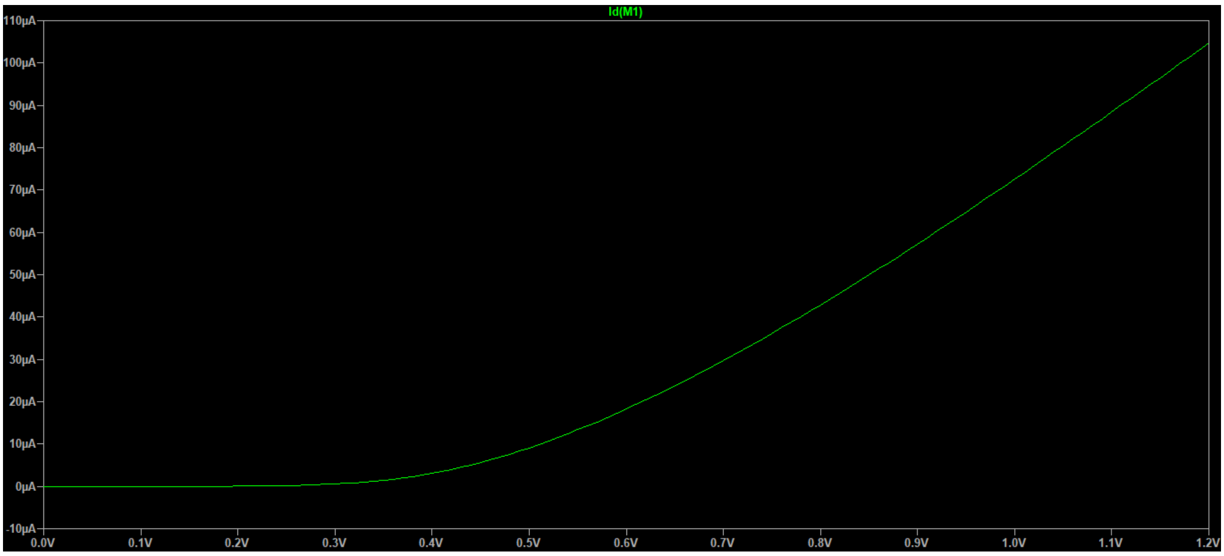


Figura 4 - Característica de I_D em função de V_{GS}, NMOS com W=L= 700 nm

Repetindo a mesma simulação para o NMOS com W=L=70nm, são obtidos os resultados apresentados na figura 5. Podemos verificar a mesma relação entre a corrente I_D e V_{GS}, para um valor fixo de tensão V_{DS} igual a 1.2V.

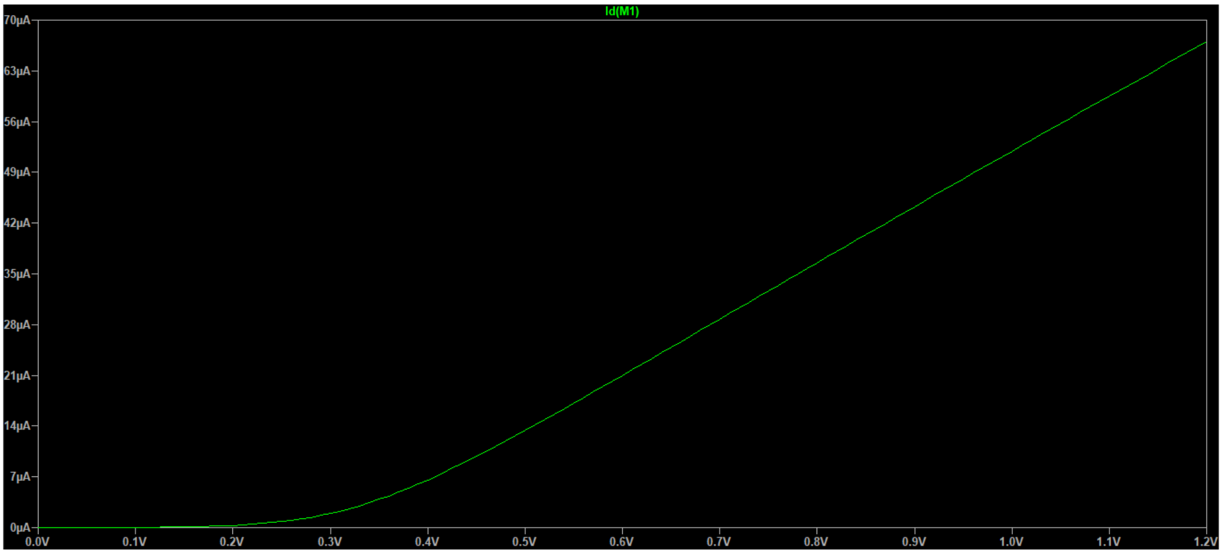


Figura 5 - Característica de I_D em função de V_{GS}, NMOS com W=L= 70 nm

Após a simulação de ambos os circuitos para os dois cenários de simulação estabelecidos, são extraídos os dados referentes às curvas características do ponto de funcionamento do circuito, efetuando a manipulação dos dados com o auxílio do software *Spyder*, aplicando o modelo *NPower*.

3.2. Python - Spyder

Nesta secção é feita a modelação dos dados simulados, de modo a implementar a metodologia *NPower* para o estudo e análise do comportamento do circuito CMOS anteriormente. O modelo é implementado em *Python*.

Uma vez obtidos os dados referentes às curvas $I_D(V_{GS})$ e $I_D(V_{DS})$, os mesmos são carregados no *Spyder* em duas *dataframes* distintas, cuja estrutura está representada na Figura 6.

Index	Vgs	0.3	0.6	0.9	1.2
0	0	7.67879e-11	-3.8077e-10	-6.63765e-09	-4.55334e-08
1	0.01	1.2651e-07	1.18762e-06	2.10014e-06	2.67093e-06
2	0.02	2.32704e-07	2.33105e-06	4.16423e-06	5.34983e-06
3	0.03	3.1945e-07	3.43003e-06	6.18566e-06	7.99081e-06
4	0.04	3.87942e-07	4.48468e-06	8.16448e-06	1.05938e-05
5	0.05	4.39919e-07	5.49514e-06	1.01008e-05	1.31588e-05
6	0.06	4.77733e-07	6.46152e-06	1.19946e-05	1.5686e-05
7	0.07	5.04202e-07	7.38399e-06	1.3846e-05	1.81752e-05
8	0.08	5.22233e-07	8.26267e-06	1.56552e-05	2.06267e-05
9	0.09	5.34385e-07	9.09774e-06	1.74221e-05	2.30404e-05
10	0.1	5.42628e-07	9.88936e-06	1.91468e-05	2.54164e-05

a)

Index	v1	Id(M1)
0	0	-6.76386e-08
1	0.01	-6.35972e-08
2	0.02	-5.97355e-08
3	0.03	-5.60305e-08
4	0.04	-5.2456e-08
5	0.05	-4.89824e-08
6	0.06	-4.5575e-08
7	0.07	-4.21929e-08
8	0.08	-3.87871e-08
9	0.09	-3.52983e-08
10	0.1	-3.16543e-08

b)

Figura 6 - Estrutura das dataframes utilizadas para armazenar: a) $I_D(V_{DS})$; b) $I_D(V_{GS})$ $com\ V_{DS} = 1.2\ V$

De seguida foram obtidos os parâmetros $V_T,\ n,\ \lambda\ e\ B$.

A função *gm(Id,Vgs)* que calcula os vários valores da transcondutância gm, através da equação (3.1).

$$gm = \frac{\partial I_D}{\partial V_{GS}} \tag{3.1}$$

Após o calculo da transcondutância é efetuada a relação $\frac{I_D}{gm}$, variável utilizada para o *curve_fit* à função *get_n_vt(Vgs, n, Vt)*, com os valores de V_{GS} e os valores de $\frac{I_D}{gm}$, a partir do qual é possível determinar os parâmetros n e V_T. A função *get_n_vt* realiza o cálculo de (3.2).

$$\frac{I_D}{gm} = \begin{cases} 0, & V_{GS} < V_T \\ \dfrac{V_{GS} - V_T}{n}, & V_{GS} \geq V_T \end{cases} \tag{3.2}$$

Onde:

$$gm = B \cdot \frac{W}{L} \cdot (V_{GS} - V_T)^{n-1}(1 + \lambda V_{DS}) \tag{3.3}$$

$$I_D = B \cdot \frac{W}{L} \cdot (V_{GS} - V_T)^{n}(1 + \lambda V_{DS}) \tag{3.4}$$

O parâmetro λ é calculado partir do declive das curvas características do gráfico simulado $I_D(V_{GS})$, selecionando dois pontos na região linear de saturação, formulando a sua expressão através da equação (3.5).

$$\lambda = \frac{I_{D1} - I_{D2}}{I_{D2} \cdot V_{DS1} - I_{D1} \cdot V_{DS2}} \tag{3.5}$$

Por último, é determinado o parâmetro B através de um *curve_fit* à função *get_B(Vgs, B)*, que recebe como parâmetros de entrada os valores de V_{GS} e de $I_D(V_{GS})$. A função *get_B* realiza o cálculo apresentado em (3.6).

$$I_D = \begin{cases} 0, & V_{GS} < V_T \\ B \cdot (V_{GS} - V_T)^n \cdot (1 + \lambda \cdot V_{DS}), & V_{GS} \geq V_T \end{cases} \tag{3.6}$$

Calculados todos os parâmetros de entrada necessários para a implementação do modelo descrito, é estabelecida a função *get_Id(Vgs, Vds, Vt, n, L, B)* que executa o cálculo do valores de corrente de I_D teóricos, através do sistema de equações (3.6), armazenados no vetor Id2 referente à implementação computacional.

Por fim calculou-se o erro relativo dos novos valores de I_D, I_{D2}, em relação aos valores de I_D obtidos através do *LTSpice*,I_{DLT}, através de (3.7).

$$Erro(V_{GS}) = \frac{I_{D2} - I_{DLT}}{I_{DLT}} * 100 \tag{3.7}$$

Esta metodologia foi utilizada para os dois ensaios realizados, e o código para os mesmos pode ser consultado em Anexos

IV - Resultados

4.1. Parâmetros Obtidos

Aplicando a metodologia apresentada no Capítulo III, foram obtidos os seguintes parâmetros, registados na Tabela 2.

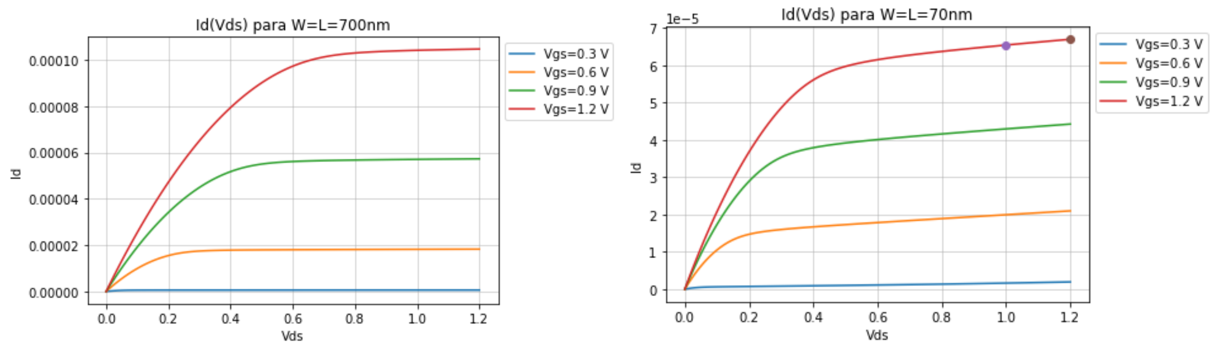
Tabela 2 – Parâmetros obtidos

NMOS 90nm	V_t	n	B	λ
W=L=700nm	$3.160 \cdot 10^{-1}$	1.467	$1.218 \cdot 10^{-4}$	$2.678 \cdot 10^{-2}$
W=L=70nm	$3.002 \cdot 10^{-1}$	1.031	$6.432 \cdot 10^{-5}$	$1.345 \cdot 10^{-1}$

4.2. Analise de Resultados

Neste Capítulo são apresentados os dados obtidos da implementação do modelo *NPower* com recurso ao software *Spyder*.

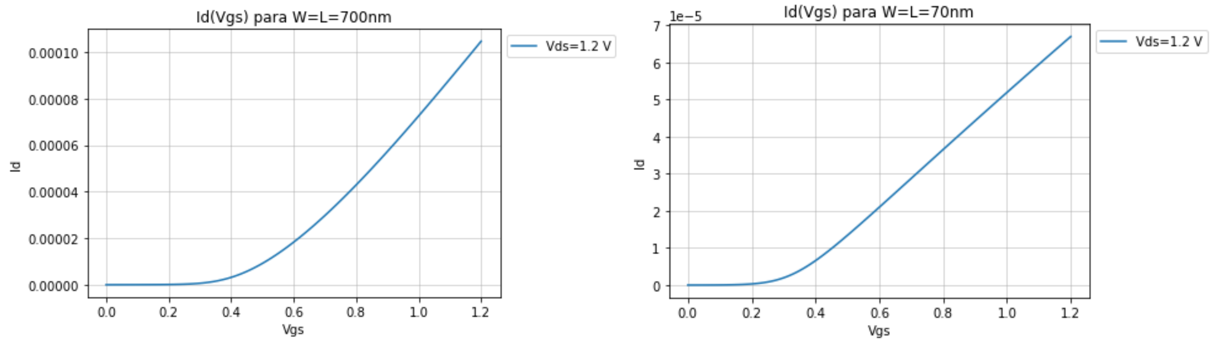
Primeiramente foi aplicada a seguinte metodologia para os transístores NMOS de 90nm com respetivas relações de igualdade W=L, para 700nm e 70nm. Na Figura 7 são apresentadas as curvas das respetivas características referentes a $I_D(V_{DS})$ para $V_{GS} \in \{0.3, 0.6, 0.9, 1.2\}V$.



a) b)

Figura 7 – Curvas $I_D(V_{DS})$, com V_{GS} fixo para: a) W=L=700 nm; b) W=L=70 nm

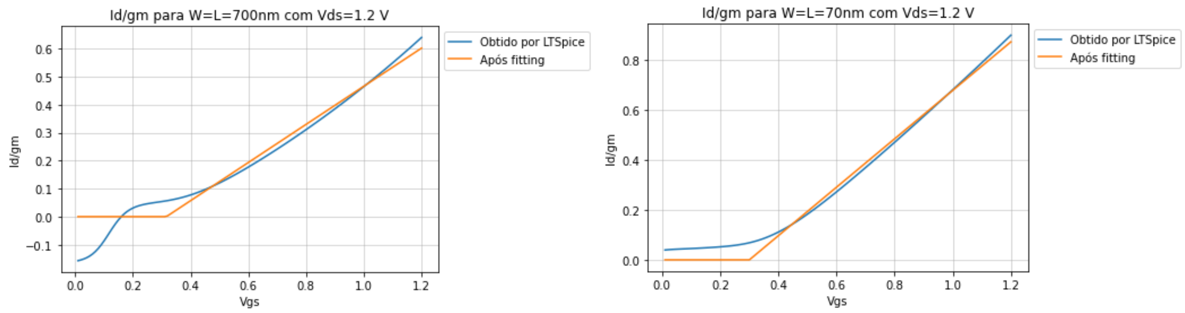
Seguidamente foi realizada a simulação do segundo cenário de operação, onde foram obtidas as curvas características $I_D(V_{GS})$ para V_{DS} igual a 1.2V, considerando as diferentes relações W=L. A Figura 8 apresenta os resultados obtidos da simulação dos circuitos em análise.



a) b)

Figura 8 - Curvas $I_D(V_{GS})$, com $V_{DS} = 1.2\ V$ para: a) W=L=700 nm; b) W=L=70 nm

Tendo em conta os dados recolhidos referentes às simulações realizadas, foi implementada a metodologia descrita na secção anterior. Deste modo, recorrendo à equação (3.2) foram obtidos os gráficos representados na Figura 9 que expressam a relação $\frac{I_D}{gm}$ simulado e modelado para os respetivos circuitos W=L.

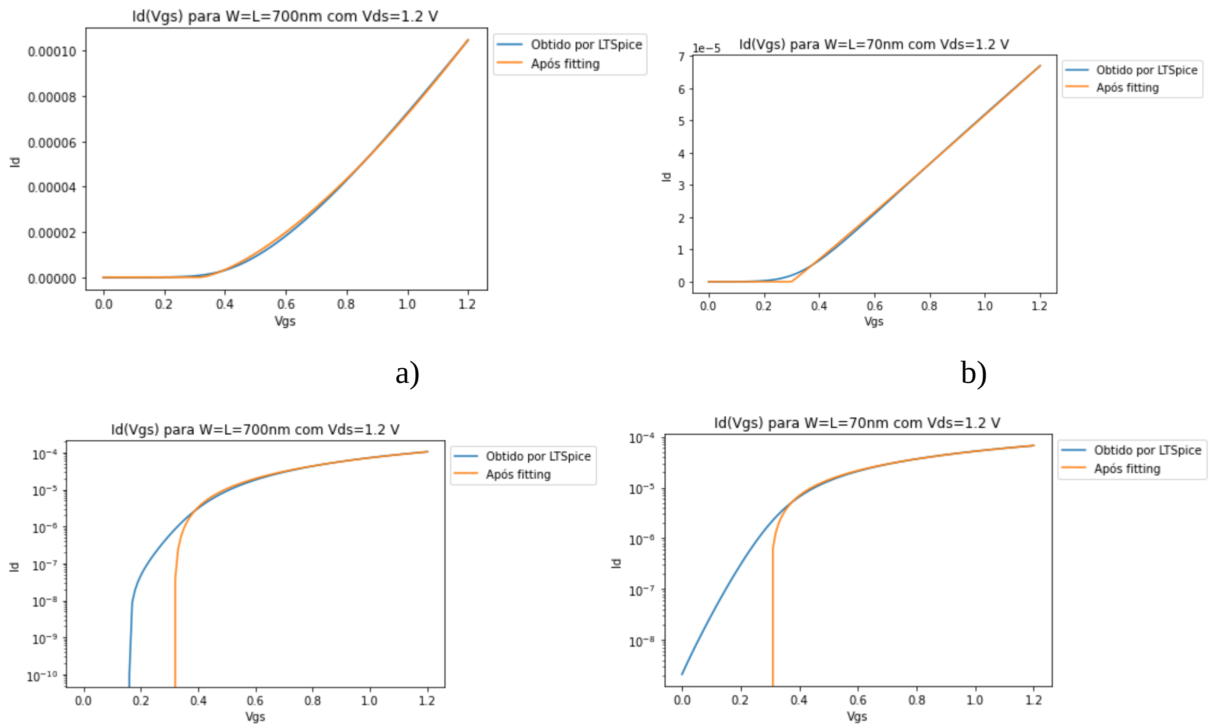


a) b)

Figura 9 - Curvas $\frac{I_D}{gm}$, com $V_{DS} = 1.2\ V$ para: a) W=L=700 nm; b) W=L=70 nm

Analisando os resultados modelados obtidos é possível verificar que as cuvas têm uma boa aproximação aos resultados simulados quando $V_{GS} \geq V_T$. Deste modo, podemos assumir que aproximação realizada para determinar os parâmetros n e V_T é bastante precisa.

Após o cálculo de todos os parâmetros de entrada para a implementação do modelo *NPower*, foram determinados as curvas características $I_D(V_{GS})$ para ambos os circuitos W=L. Na figura 10 são representadas as respetivas curvas $I_D(V_{GS})$ simuladas e modeladas (NPower) para ambos os circuitos W=L, tanto em escala linear como em escala logarítmica.



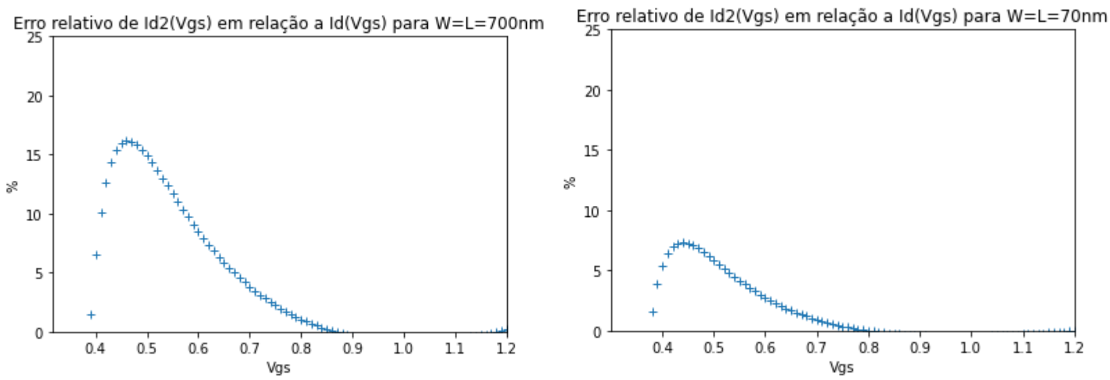
a) b)

c) d)

Figura 10 - Curvas $I_D(V_{GS})$, para: a) W=L=700 nm; b) W=L=70 nm ;c) W=L=700 nm (escala logarítmica) ; d) W=L=70 nm (escala logarítmica)

Estabelecendo uma comparação de modo grosseiro entre os resultados simulados e modelos, podemos afirma que o modelo *NPower* tem uma boa aproximação para a região linear (saturação, $V_{GS} \geq V_T$).

Contudo, de modo a determinar qual o grau de precisão dos resultados obtidos, foi calculado o valor do erro relativo entres os valores simulados e modelados, observando a variação do valor do erro tendo em consideração a tensão V_{GS} do componente NMOS. Na Figura 11 são apresentados os gráficos referentes aos erros relativos para ambos os circuitos W=L, considerando apenas erro pera valores $V_{GS} \geq V_T$.



a) b)

Figura 11 - Erro de $I_{D2}(V_{GS})$ em relação a $I_D(V_{GS})$, para: a) W=L=700 nm; b) W=L=70 nm

Analisando os dados da Figura 11, podemos verificar que o erro relativo é superior para transístores com canais de maiores dimensões. Uma vez que apenas se considerou a região $V_{GS} \geq V_T$, é visível que o maior de valor de erro relativo acontece no limiar da zona de saturação.

V - Conclusões

Com a realização deste trabalho laboratorial foram abordados vários conceitos que nos permitem retirar algumas conclusões em cada uma das partes de desenvolvimento do projeto, descrevendo e analisando o funcionamento de um circuito NMOS 90nm com uma relação W=L.

Assim sendo, inicialmente realizou-se a análise teórica do circuito executando uma simulação para dois pontos de operação diferentes variando a dimensão do transístor, mantendo a relação W=L. Deste modo, segundo a simulação do primeiro ensaio que corresponde à obtenção da corrente $I_D(V_{DS})$ para valores de V_{GS} fixos onde $V_{GS} = \{0{,}3; 0{,}6; 0{,}9; 1{,}2\}\, V$, verificamos que para $V_{GS} \geq V_T$ a corrente I_D se encontra na região de saturação. Por outro lado, executando o segundo cenário de simulação, que corresponde à obtenção da corrente I_D para um valor fixo de V_{DS} igual a 1.2V, observamos que a corrente I_D cresce linearmente ao entrar na região de saturação ($V_{GS} \geq V_t$).

Após as simulações do *LTSpice* é descrito e implementado um modelo *NPower* que determina os parâmetros de saída simulados no *LTSpice*, analisando a precisão e incerteza do modelo. Observando os parâmetros resultantes do fitting das curvas de simulação, podemos afirmar que o valor da constante *n* não corresponde exatamente ao fator 2, característica do modelo de *Shockley*, o que revela uma certa incerteza por parte deste último, Tabela 2.

É possível constatar também que o valor de *B* é responsável pela conversão da corrente em tensão onde o seu valor é maior quanto maior forem as dimensões do transístor MOSFET.

Analisando o valor de λ obtido por regressão linear, retirando dois pontos da curva característica $I_D(V_{DS})$ na região de saturação, podemos averiguar que o valor do λ depende da relação $I_D(V_{DS})$ onde os efeitos da modulação do canal são mais notórios para transístores de canais mais curtos.

Através do cálculo do erro relativo é visível que este converge para 0 com o aumento do valor de V_{GS}. Desta maneira, observando os gráficos do erro relativo, para um transístor com W=L=70nm o erro é menor do que para um transístor de dimensões maiores (com a mesma relação W=L), logo o modelo *NPower* é mais preciso/adequado para transístores com canais mais curtos.

Contudo, podemos afirmar que o modelo *NPower* é um modelo simples e compacto que apresenta uma aproximação robusta para o cálculo dos parâmetros de saída do modelo, nomeadamente da corrente $I_D(V_{DS})$, concluindo também que as aproximações efetuadas por fitting dos parâmetros correspondem a uma boa aproximação para a implementação do modelo.

Finalizando, o modelo *NPower* apresenta algumas falhas para transístores com diferentes dimensões, com W=L, apresentando um comportamento não escalável.

Bibliografia

[1] T. Sakurai and A. Richard Newton, "A Simple MOSFET Model for Circuit Analysis," *IEEE Trans. Electron Devices*, vol. 38, no. 4, pp. 887–894, 1991, doi: 10.1109/16.75219.

Anexos

Trabalho1_70nm.py

```
# -*- coding: utf-8 -*-
"""
Created on Mon Apr 12 10:59:50 2021
@authors: Rafael, Francisco, José
"""
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
os.chdir('G:\\O meu disco\\MIEEC\\5º Ano\\2º Semestre\\EDA\\Trabalho 1')
############################################## PARTE 1 ##########################################
########################### 1 - CARREGAMENTO DE DADOS #########################################
#Dados para Id(Vgs,1.2)
vgs12_70=pd.read_csv('Id_Vgs_12_70nm.txt',sep='\t')
#Dados para Id(Vds), com os 4 ensaios realizados (0.3, 0.6, 0.9, 1.2)
vds_70=pd.read_csv('Id_Vds_70nm.txt',sep='\t')
#Carregar os dados de Id(Vds) para uma dataframe, de modo a ter os 4 ensaios separados
U=[]
U.append(vgs12_70['v1'])
for i in range(0,4,1):
U.append(vds_70['Id(M1)'][1+i*122:122+i*122].to_numpy())
# i*122 - offset entre ensaios, contabilizando a linha da string
vds_70 = pd.DataFrame(U).transpose()
vds_70.columns = ['Vgs','0.3','0.6','0.9', '1.2']
del U, i
###############################################################################################
################################ 2 - PLOT ID(VDS) PARA OS VÁRIOS VGS #############################
plt.figure()
plt.title('Id(Vds) para W=L=70nm')
plt.ylabel('Id')
plt.xlabel('Vds')
for vgs in np.arange(0.3,1.5,0.3):
plt.plot(vds_70['Vgs'], vds_70[str(round(vgs,2))],label='Vgs=%.1f V' %vgs)
plt.plot(vds_70['Vgs'][100], vds_70['1.2'][100],'o', vds_70['Vgs'][120], vds_70['1.2'][120],'o'
plt.legend(bbox_to_anchor=(1, 1))
plt.grid(linewidth=0.5)
del vgs
###############################################################################################
################################ 3 - PLOT ID(VGS) COM VDS = 1.2 V ################################
plt.figure()
plt.title('Id(Vgs) para W=L=70nm')
plt.ylabel('Id')
plt.xlabel('Vgs')
plt.plot(vgs12_70['v1'],vgs12_70['Id(M1)'], label='Vds=1.2 V')
plt.legend(bbox_to_anchor=(1, 1))
plt.grid(linewidth=0.5)
###############################################################################################
############################## 4 - OBTENÇÃO DOS PARÂMETROS LAMBDA, VT, N e B ###################
############ Definição de funções #################
def gm(Id,Vgs): #gm= dId/dVgs
return np.diff(Id)/np.diff(Vgs)
def get_n_vt(Vgs, n, Vt):
return np.piecewise(Vgs, [Vgs < Vt, Vgs >= Vt],[lambda Vgs:0, lambda Vgs:(Vgs-Vt)/n])
```

```
def get_lambda(Id1,Id2,Vds1,Vds2):
return (Id1-Id2)/(Id2*Vds1-Id1*Vds2)
def get_B(Vgs, B):
return np.piecewise(Vgs, [Vgs < Vt, Vgs >= Vt],[lambda Vgs:0, lambda Vgs:B*((Vgs-Vt)**n)* (1+L*1.2)])
##########################################################################################
# Obtenção dos valores de gm
gm_70 = gm(vgs12_70['Id(M1)'],vgs12_70['v1'])
# Obtenção dos valores de Id/gm
id_gm_70=(vgs12_70['Id(M1)'][1:]/gm_70).to_numpy()
#Obtenção de n e Vt
xx,xy=curve_fit(get_n_vt,vgs12_70['v1'][1:].values,id_gm_70)
n=xx[0]
Vt=xx[1]
#Obtenção dos valores Id/gm após o fitting
id_gm_new=get_n_vt(vgs12_70['v1'][1:].values,n,Vt)
#Obtenção de Lambda
L=get_lambda(vds_70['1.2'][100], vds_70['1.2'][120], vds_70['Vgs'][100], vds_70['Vgs'][120])
#Obtenção de B
xx,xy=curve_fit(get_B,vgs12_70['v1'].values,vgs12_70['Id(M1)'].values)
B=xx[0]
del xx, xy
######################## PLOTS ###########################################################
#Id/gm antes e após fitting
plt.figure()
plt.title('Id/gm para W=L=70nm com Vds=1.2 V')
plt.ylabel('Id/gm')
plt.xlabel('Vgs')
plt.plot(vgs12_70['v1'][1:],id_gm_70, label='Obtido por LTSpice')
plt.plot(vgs12_70['v1'][1:],id_gm_new, label='Após fitting')
plt.legend(bbox_to_anchor=(1, 1))
plt.grid(linewidth=0.5)
#Id(Vgs) com Vds = 1.2 V antes e após fitting
plt.figure()
plt.title('Id(Vgs) para W=L=70nm com Vds=1.2 V')
plt.ylabel('Id')
plt.xlabel('Vgs')
plt.plot(vgs12_70['v1'],vgs12_70['Id(M1)'], label='Obtido por LTSpice')
plt.plot(vgs12_70['v1'],get_B(vgs12_70['v1'].values,B),label='Após fitting')
plt.legend(bbox_to_anchor=(1, 1))
#Id(Vgs) com Vds = 1.2 V antes e após fitting em escala logarítmica
plt.figure()
plt.title('Id(Vgs) para W=L=70nm com Vds=1.2 V')
plt.ylabel('Id')
plt.xlabel('Vgs')
plt.plot(vgs12_70['v1'],vgs12_70['Id(M1)'], label='Obtido por LTSpice')
plt.plot(vgs12_70['v1'],get_B(vgs12_70['v1'].values,B),label='Após fitting')
plt.yscale('log')
plt.legend(bbox_to_anchor=(1, 1))

############################################## PARTE 2 ###################################
############################# 1 - get_ID FUNCTION ########################################
def get_Id(Vgs, Vds, Vt, n, L, B):
return np.piecewise(Vgs, [Vgs < Vt, Vgs >= Vt],[lambda Vgs:0, lambda Vgs:B*((Vgs-Vt)**n)* (1+L*Vds)])
###################### 2 - Id2(Vgs,Vds) matrix ###########################################
Id2=get_Id(np.arange(0,1.21,0.01),1.2,Vt,n,L,B)
##################### 3 - Id2(Vgs) plot, with Vds = 1.2 V ##############################
plt.figure()
plt.title('Id(Vgs) para W=L=70nm com Vds=1.2 V')
plt.ylabel('Id')
```

```
plt.xlabel('Vgs')
plt.plot(vgs12_70['v1'],get_B(vgs12_70['v1'].values,B),label='Após fitting')
plt.plot(vgs12_70['v1'],Id2, label='através de get_Id')
plt.yscale('log')
plt.legend(bbox_to_anchor=(1, 1))
##################### 4 - Relative Error of Id2(Vgs) in relation to Id1(Vgs) ##########
# i.e., Error (Vgs) = (Id2 (Vgs) -Id1 (Vgs)) / Id1 (Vgs).
E_rel=((Id2-vgs12_70['Id(M1)'])/vgs12_70['Id(M1)'])*100
plt.figure()
plt.ylabel('%')
plt.xlabel('Vgs')
plt.title('Erro relativo de Id2(Vgs) em relação a Id(Vgs) para W=L=70nm')
plt.plot(vgs12_70['v1'],E_rel,'+')
plt.axis([Vt,1.2,0,25])
```

Trabalho1_700nm.py

```
# -*- coding: utf-8 -*-
"""
Created on Mon Apr 12 10:59:50 2021
@authors: Rafael, Francisco, José
"""
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
os.chdir('G:\\O meu disco\\MIEEC\\5º Ano\\2º Semestre\\EDA\\Trabalho 1')
############################################# PARTE 1 ##########################################
########################## 1 - CARREGAMENTO DE DADOS #########################################
#Dados para Id(Vgs,1.2)
vgs12_700=pd.read_csv('Id_Vgs_12_700nm.txt',sep='\t')
#Dados para Id(Vds), com os 4 ensaios realizados (0.3, 0.6, 0.9, 1.2)
vds_700=pd.read_csv('Id_Vds_700nm.txt',sep='\t')
#Carregar os dados de Id(Vds) para uma dataframe, de modo a ter os 4 ensaios separados
U=[]
U.append(vgs12_700['v1'])
for i in range(0,4,1):
U.append(vds_700['Id(M1)'][1+i*122:122+i*122].to_numpy())
# i*122 - offset entre ensaios, contabilizando a linha da string
vds_700 = pd.DataFrame(U).transpose()
vds_700.columns = ['Vgs','0.3','0.6','0.9', '1.2']
del U, i
#######################################################################################
########################## 2 - PLOT ID(VDS) PARA OS VÁRIOS VGS ###########################
plt.figure()
plt.title('Id(Vds) para W=L=700nm')
plt.ylabel('Id')
plt.xlabel('Vds')
for vgs in np.arange(0.3,1.5,0.3):
plt.plot(vds_700['Vgs'], vds_700[str(round(vgs,2))],label='Vgs=%.1f V' %vgs)
plt.plot(vds_700['Vgs'][100], vds_700['1.2'][100],'o', vds_700['Vgs'][120], vds_700['1.2'][120],'o')
plt.legend(bbox_to_anchor=(1, 1))
plt.grid(linewidth=0.5)
del vgs
############################################################################################
############################### 3 - PLOT ID(VGS) COM VDS = 1.2 V ###############################
plt.figure()
plt.title('Id(Vgs) para W=L=700nm')
plt.ylabel('Id')
plt.xlabel('Vgs')
plt.plot(vgs12_700['v1'],vgs12_700['Id(M1)'], label='Vds=1.2 V')
plt.legend(bbox_to_anchor=(1, 1))
plt.grid(linewidth=0.5)
############################################################################################
############################## 4 - OBTENÇÃO DOS PARÂMETROS LAMBDA, VT, N e B ##################
########### Definição de funções #################
def gm(Id,Vgs): #gm= dId/dVgs
return np.diff(Id)/np.diff(Vgs)
def get_n_vt(Vgs, n, Vt):
return np.piecewise(Vgs, [Vgs < Vt, Vgs >= Vt],[lambda Vgs:0, lambda Vgs:(Vgs-Vt)/n])
def get_lambda(Id1,Id2,Vds1,Vds2):
return (Id1-Id2)/(Id2*Vds1-Id1*Vds2)
```

```
def get_B(Vgs, B):
return np.piecewise(Vgs, [Vgs < Vt, Vgs >= Vt],[lambda Vgs:0, lambda Vgs:B*((Vgs-Vt)**n)*(1+L*1.2)])
####################################################################################################
# Obtenção dos valores de gm
gm_700 = gm(vgs12_700['Id(M1)'],vgs12_700['v1'])
# Obtenção dos valores de Id/gm
id_gm_700=(vgs12_700['Id(M1)'][1:]/gm_700).to_numpy()
#Obtenção de n e Vt
xx,xy=curve_fit(get_n_vt,vgs12_700['v1'][1:].values,id_gm_700)
n=xx[0]
Vt=xx[1]
#Obtenção dos valores Id/gm após o fitting
id_gm_new=get_n_vt(vgs12_700['v1'][1:].values,n,Vt)
#Obtenção de Lambda
L=get_lambda(vds_700['1.2'][100], vds_700['1.2'][120], vds_700['Vgs'][100], vds_700['Vgs'][120])
#Obtenção de B
xx,xy=curve_fit(get_B,vgs12_700['v1'].values,vgs12_700['Id(M1)'].values)
B=xx[0]
del xx, xy
######################### PLOTS ###############################
#Id/gm antes e após fitting
plt.figure()
plt.title('Id/gm para W=L=700nm com Vds=1.2 V')
plt.ylabel('Id/gm')
plt.xlabel('Vgs')
plt.plot(vgs12_700['v1'][1:],id_gm_700, label='Obtido por LTSpice')
plt.plot(vgs12_700['v1'][1:],id_gm_new, label='Após fitting')
plt.legend(bbox_to_anchor=(1, 1))
plt.grid(linewidth=0.5)
#Id(Vgs) com Vds = 1.2 V antes e após fitting
plt.figure()
plt.title('Id(Vgs) para W=L=70nm com Vds=1.2 V')
plt.ylabel('Id')
plt.xlabel('Vgs')
plt.plot(vgs12_700['v1'],vgs12_700['Id(M1)'], label='Obtido por LTSpice')
plt.plot(vgs12_700['v1'],get_B(vgs12_700['v1'].values,B),label='Após fitting')
plt.legend(bbox_to_anchor=(1, 1))
#Id(Vgs) com Vds = 1.2 V antes e após fitting em escala logarítmica
plt.figure()
plt.title('Id(Vgs) para W=L=700nm com Vds=1.2 V')
plt.ylabel('Id')
plt.xlabel('Vgs')
plt.plot(vgs12_700['v1'],vgs12_700['Id(M1)'], label='Obtido por LTSpice')
plt.plot(vgs12_700['v1'],get_B(vgs12_700['v1'].values,B),label='Após fitting')
plt.yscale('log')
plt.legend(bbox_to_anchor=(1, 1))
############################################# PARTE 2 ########################################
############################# 1 - get_ID FUNCTION ############################################
def get_Id(Vgs, Vds, Vt, n, L, B):
return np.piecewise(Vgs, [Vgs < Vt, Vgs >= Vt],[lambda Vgs:0, lambda Vgs:B*((Vgs-Vt)**n)*(1+L*Vds)])
################################## 2 - Id2(Vgs,Vds) matrix ##################################
Id2=get_Id(np.arange(0,1.21,0.01),1.2,Vt,n,L,B)
###################################### 3 - Id2(Vgs) plot, with Vds = 1.2 V ##############################
plt.figure()
plt.title('Id(Vgs) para W=L=700nm com Vds=1.2 V')
plt.ylabel('Id')
plt.xlabel('Vgs')
plt.plot(vgs12_700['v1'],get_B(vgs12_700['v1'].values,B),label='Após fitting')
plt.plot(vgs12_700['v1'],Id2, label='através de get_Id')
```

```
plt.yscale('log')
plt.legend(bbox_to_anchor=(1, 1))
############################ 4 - Relative Error of Id2(Vgs) in relation to Id1(Vgs) #####################
# i.e., Error (Vgs) = (Id2 (Vgs) -Id1 (Vgs)) / Id1 (Vgs).
E_rel=((Id2-vgs12_700['Id(M1)'])/vgs12_700['Id(M1)'])*100
plt.figure()
plt.ylabel('%')
plt.xlabel('Vgs')
plt.title('Erro relativo de Id2(Vgs) em relação a Id(Vgs) para W=L=700nm')
plt.plot(vgs12_700['v1'],E_rel,'+')
plt.axis([Vt,1.2,0,25])
```