nome: Bruno da Silva Pinho

matricula:319466

Trabalho 1.
Regressão Linear

Para essa atividade foram implementado na linguagem python as classes para calcular a regreção linear usando o metodo analitico e o gradiente descendente. A implementação das classes podem ser vistas nos aquivos Regressao_Linear_Metodo_Analitico.py e Regressao_Linear_Gradiente_Descendente.py. Nos arquivos .py tambem foi implementado a classe metricas que contem funções de erro MSE e R2

Para os metodos que usam o gradiente, os coeficientes iniciais foram iniciados com 0

O arquivo Regressao_Linear_Metodo_Analitico.py contem a implementação das classes:

- Regressao_Linear_Simples: serve para fazer a regressão linear univariada
- Regressao_Linear_Mutipla: serve para fazer a regressão linear mutivariada
- Regressao_Quadratica: serve para fazer a regressão quadrática
- Regresso_Cubica: serve para fazer a regressão cubica

O arquivo Regressao_Linear_Gradiente_Descendente.py contem a implementação das classes:

- Gradiente_Regressao_Linear_Simples: serve para fazer a regressão linear univariada utilizando gradiente descendente
- Gradiente_Regressao_Linear_Mutipla: serve para fazer a regressão linear mutivariada utilizando gradiente descedente
- Gradiente_Estocastico_Regressao_Linear_Mutipla: serve para fazer a regressão linear mutivariada utilizando gradiente descendente estocástico
- Gradiente_Regressao_Linear_Mutiplar_Reguralizada: serve para fazer a regressão linear mutivariada utilizando gradiente descendente regularizado

Abaixo foram feito os importes dos arquivos e a separação dos dados em 80% para treino e 20% para teste onde usamos LSTAT para atributo preditor e MEDV para variável alvo.

In [30]:

```
import numpy as np
import Regressao_Linear_Gradiente_Descendente
import Regressao_Linear_Metodo_Analitico

data = np.loadtxt("./housing.data")
np.random.shuffle(data)
X = data[:, -2]
y = data[:, -1]

n = X.shape[0]
n_train = int(n*0.8)
n_test = n - n_train
X_train = X[:n_train]
X_test = X[-n_test:]
y_train = y[:n_train]
y_test = y[-n_test:]

metricas = Regressao_Linear_Gradiente_Descendente.Metricas()
```

Abaixao são estanciadas cada uma das classes, cada letra representa uma calsse, as letras são equivalentes as letras da questão 1. do Trabalho 1. Nas regressões com gradiente foram escolhido as epocas e as taxas de aprendizados de forma que a predição dessas classe sejam aproximadas as predições dos metodos analiticos.

In [31]:

```
a = Regressao_Linear_Metodo_Analitico.Regressao_Linear_Simples()

b = Regressao_Linear_Gradiente_Descendente.Gradiente_Regressao_Linear_Simples(10000, 0.
001)

c = Regressao_Linear_Metodo_Analitico.Regressao_Linear_Mutipla()

d = Regressao_Linear_Gradiente_Descendente.Gradiente_Regressao_Linear_Mutipla(50, 0.01)

e = Regressao_Linear_Gradiente_Descendente.Gradiente_Estocastico_Regressao_Linear_Mutip
la(50, 0.01)

f = Regressao_Linear_Metodo_Analitico.Regressao_Quadratica()

g = Regressao_Linear_Metodo_Analitico.Regressao_Cubica()

h = Regressao_Linear_Gradiente_Descendente.Gradiente_Regressao_Linear_Mutiplar_Regurali
zada(50, 0.01, 1)
```

Mesta parte é feito o treinamento utilizando nossas variaveis de treino e as classes instaciadas. Para nosso conjunto de dados utilizamos **a, b, f** e **g**

In [32]:

```
a.fit(X_train, y_train)

b.fit(X_train, y_train)

f.fit(X_train, y_train)

g.fit(X_train, y_train)
```

Para **a, b, f** e **g** é repotado MSE, R2 e os coeficientes. Abaixo podemos ver o resultado de cada classe.

- MSE, R2 e o coeficiente para o **a**:

In [33]:

```
print("teste-------------------------------------------------")
print("MSE = "+ str(metricas.MSE(y_test, a.predict(X_test))))
print("R2 = "+ str(metricas.R2(y_test, a.predict(X_test))))
print("treino------------------------------------------------")
print("MSE = "+ str(metricas.MSE(y_train, a.predict(X_train))))
print("R2 = "+ str(metricas.R2(y_train, a.predict(X_train))))
print("------------------------------------------------------")
print("coeficiente w = " + str(a.w))
```

```
teste-------------------------------------------------
MSE = 26.773433221173907
R2 = 0.6142381897320108
treino------------------------------------------------
MSE = 41.52841083231586
R2 = 0.529158902376902
------------------------------------------------------
coeficiente w = [[34.52147327 -0.92978145]]
```

- MSE, R2 e o coeficiente para o **b**:

In [34]:

```
print("teste-------------------------------------------------")
print("MSE = "+ str(metricas.MSE(y_test, b.predict(X_test))))
print("R2 = "+ str(metricas.R2(y_test, b.predict(X_test))))
print("treino------------------------------------------------")
print("MSE = "+ str(metricas.MSE(y_train, b.predict(X_train))))
print("R2 = "+ str(metricas.R2(y_train, b.predict(X_train))))
print("------------------------------------------------------")
print("coeficiente w = " + str(b.w))
```

```
teste-------------------------------------------------
MSE = 28.7855920831115
R2 = 0.5852462394387645
treino------------------------------------------------
MSE = 43.6693868900102
R2 = 0.5048849295282916
------------------------------------------------------
coeficiente w = [[31.57451254 -0.75677071]]
```

- MSE, R2 e o coeficiente para o **f**:

In [35]:

```
print("teste-------------------------------------------------")
print("MSE = "+ str(metricas.MSE(y_test, f.predict(X_test))))
print("R2 = "+ str(metricas.R2(y_test, f.predict(X_test))))
print("treino------------------------------------------------")
print("MSE = "+ str(metricas.MSE(y_train, f.predict(X_train))))
print("R2 = "+ str(metricas.R2(y_train, f.predict(X_train))))
print("------------------------------------------------------")
print("coeficiente w = " + str(f.w))
```

```
teste-------------------------------------------------
MSE = 22.720564399979505
R2 = 0.6726334653893062
treino------------------------------------------------
MSE = 32.2852194466166
R2 = 0.6339564202774999
------------------------------------------------------
coeficiente w = [43.24973387 -2.37147844  0.04470912]
```

- MSE, R2 e o coeficiente para o **g**:

In [36]:

```
print("teste-------------------------------------------------")
print("MSE = "+ str(metricas.MSE(y_test, g.predict(X_test))))
print("R2 = "+ str(metricas.R2(y_test, g.predict(X_test))))
print("treino------------------------------------------------")
print("MSE = "+ str(metricas.MSE(y_train, g.predict(X_train))))
print("R2 = "+ str(metricas.R2(y_train, g.predict(X_train))))
print("------------------------------------------------------")
print("coeficiente w = " + str(g.w))
```

```
teste-------------------------------------------------
MSE = 23.481808808543104
R2 = 0.6616651663789412
treino------------------------------------------------
MSE = 30.329668540656336
R2 = 0.6561280785848214
------------------------------------------------------
coeficiente w = [ 4.98977988e+01 -4.12021496e+00  1.63373353e-01 -2.233534
37e-03]
```

Acima podemos ver que os piore casos foi para o **a** e **b**, ja que o **b** usa gradiente, e o gradiente fuciona como uma aproximação, então seu resulatdo não foi um dos melhores, talvez esse erro possa diminuir se for mexido no termo de regularização.
O **a** se dá por causa de como os dados estão, eles não estão alinhados como uma linha reta logo o **a** não vai fazer a melhor predição para esses dados.

Para o **f** temos que o erro deu uma diminuida mas ainda, isso mostra que a predição está se ajustando ao real, porém ainda não é a melhor predição

O **g** foi o que teve o melhor resultado, isso se deve ao fato dele se ajustar melhor aos dados.

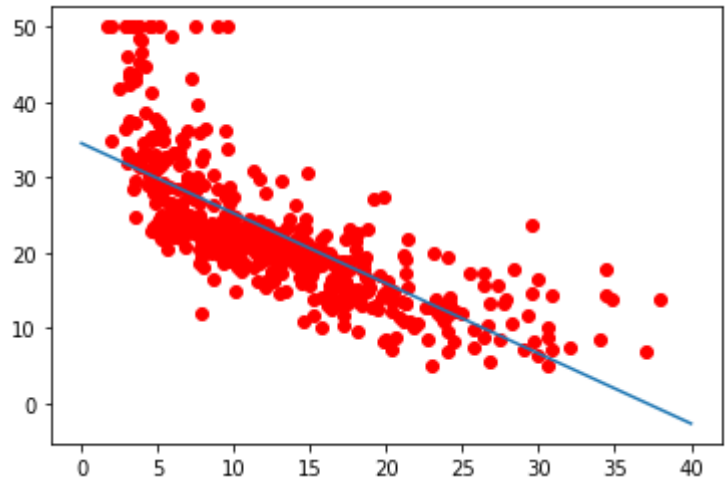
Podemos ver que os erros de teste estão menores que os erros de treino, logo nossos modelos vão se ajustar melhor a variaveis de teste.

Abaixo podemos ver os graficos para **a, b, f, g** que mostram o comportamento que cada metodo teve em relação aos dados reais.

In [37]:

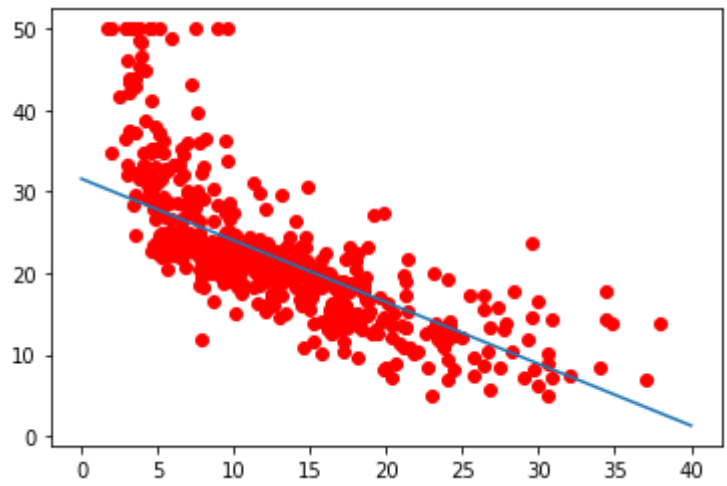
```
import matplotlib.pyplot as pl
X_ = np.linspace(0., 40.)

pl.plot(X, y,"o", color="red", fillstyle='full')
pl.plot(X_, a.predict(X_))
pl.show()
```



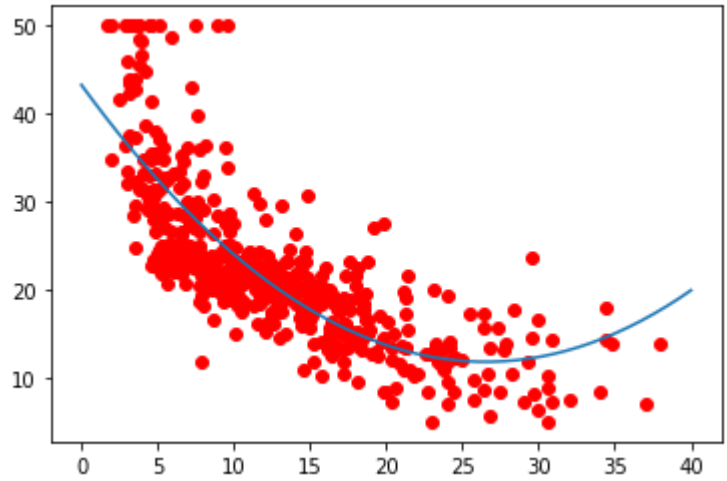
In [38]:

```
pl.plot(X, y,"o", color="red", fillstyle='full')
pl.plot(X_, b.predict(X_))
pl.show()
```



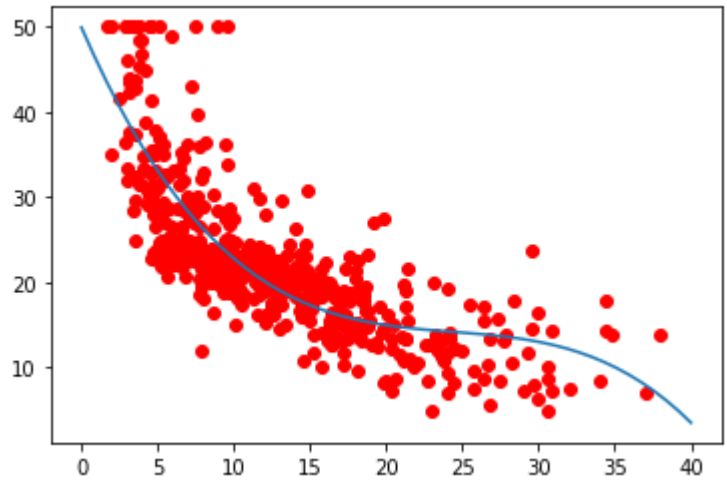
In [39]:

```
pl.plot(X, y,"o", color="red", fillstyle='full')
pl.plot(X_, f.predict(X_))
pl.show()
```



In [40]:

```
pl.plot(X, y,"o", color="red", fillstyle='full')
pl.plot(X_, g.predict(X_))
pl.show()
```



A seguir é utilizado o conjunto de dados trab1_data. Tambem é separado 80% para treino e 20% para testes. Foi utilizado a ultima colunda do dados para atributo alvo, e o restando dos dados como atributo preditor.

In [12]:

```
data = np.loadtxt("./trab1_data.txt")
np.random.shuffle(data)
y = data[: ,-1]
X = data.T[: 5].T
n = X.shape[0]
n_train = int(n*0.8)
n_test = n - n_train
X_train = X[:n_train]
X_test = X[-n_test:]
y_train = y[:n_train]
y_test = y[-n_test:]
```

Abaixo é feito o treinamento utilizando nossas variaveis de treino e as classes instaciadas. Para nosso conjunto de dados utilizamos **c, d, e** e **h**.

In [13]:

```
c.fit(X_train, y_train)
d.fit(X_train, y_train)
e.fit(X_train, y_train)
h.fit(X_train, y_train)
```

Para **c, d, e** e **h** é repotado MSE, R2 e os coeficientes. Abaixo podemos ver o resultado de cada classe.

- MSE, R2 e o coeficiente para o **c**:

In [14]:

```
print("teste--------------------------------------------------")
print("MSE = "+ str(metricas.MSE(y_test, c.predict(X_test))))
print("R2 = "+ str(metricas.R2(y_test, c.predict(X_test))))
print("treino-------------------------------------------------")
print("MSE = "+ str(metricas.MSE(y_train, c.predict(X_train))))
print("R2 = "+ str(metricas.R2(y_train, c.predict(X_train))))
print("-------------------------------------------------------")
print("coeficiente w = " + str(c.w))
```

```
teste--------------------------------------------------
MSE = 0.6410210643413213
R2 = 0.5151143268989098
treino-------------------------------------------------
MSE = 0.34002227046905265
R2 = 0.7821796130742892
-------------------------------------------------------
coeficiente w = [ 0.89871711  1.58285983 -0.12600506 -0.02525211 -0.068045
27 -0.12805251]
```

- MSE, R2 e o coeficiente para o **d**:

In [15]:

```
print("teste-------------------------------------------------")
print("MSE = "+ str(metricas.MSE(y_test, d.predict(X_test))))
print("R2 = "+ str(metricas.R2(y_test, d.predict(X_test))))
print("treino------------------------------------------------")
print("MSE = "+ str(metricas.MSE(y_train, d.predict(X_train))))
print("R2 = "+ str(metricas.R2(y_train, d.predict(X_train))))
print("------------------------------------------------------")
print("coeficiente w = " + str(d.w))
```

```
teste-------------------------------------------------
MSE = 0.9229107808955869
R2 = 0.3018853199985826
treino------------------------------------------------
MSE = 0.6232787913119089
R2 = 0.6007237193644177
------------------------------------------------------
coeficiente w = [0.3303788 0.3303788 0.3303788 0.3303788 0.3303788 0.33037
88]
```

- MSE, R2 e o coeficiente para o **e**:

In [16]:

```
print("teste-------------------------------------------------")
print("MSE = "+ str(metricas.MSE(y_test, e.predict(X_test))))
print("R2 = "+ str(metricas.R2(y_test, e.predict(X_test))))
print("treino------------------------------------------------")
print("MSE = "+ str(metricas.MSE(y_train, e.predict(X_train))))
print("R2 = "+ str(metricas.R2(y_train, e.predict(X_train))))
print("------------------------------------------------------")
print("coeficiente w = " + str(e.w))
```

```
teste-------------------------------------------------
MSE = 0.6644173703747059
R2 = 0.4974167281300863
treino------------------------------------------------
MSE = 0.37047094613070286
R2 = 0.7626740015599406
------------------------------------------------------
coeficiente w = [ 3.37388944e-01  1.24479888e+00  4.03908003e-04  6.580633
48e-01
 -3.60013144e-02 -2.55650973e-02]
```

- MSE, R2 e o coeficiente para o **h**:

In [17]:

```
print("teste-------------------------------------------------")
print("MSE = "+ str(metricas.MSE(y_test, h.predict(X_test))))
print("R2 = "+ str(metricas.R2(y_test, h.predict(X_test))))
print("treino------------------------------------------------")
print("MSE = "+ str(metricas.MSE(y_train, h.predict(X_train))))
print("R2 = "+ str(metricas.R2(y_train, h.predict(X_train))))
print("------------------------------------------------------")
print("coeficiente w = " + str(h.w))
```

```
teste-------------------------------------------------
MSE = 0.9334696156026454
R2 = 0.29389833180287317
treino------------------------------------------------
MSE = 0.6190242329395496
R2 = 0.6034492159902325
------------------------------------------------------
coeficiente w = [0.16472288 0.34714118 0.34714118 0.34714118 0.34714118 0.
34714118]
```

Podemos ver que os erros de teste estão altos, e os erros de treino estão baixos isso mostra que temos um problema de Alto Variância, conhecido como Overfitting. Isso mostra que o modelo esta se ajustando demasiadamente aos dados de treinos.

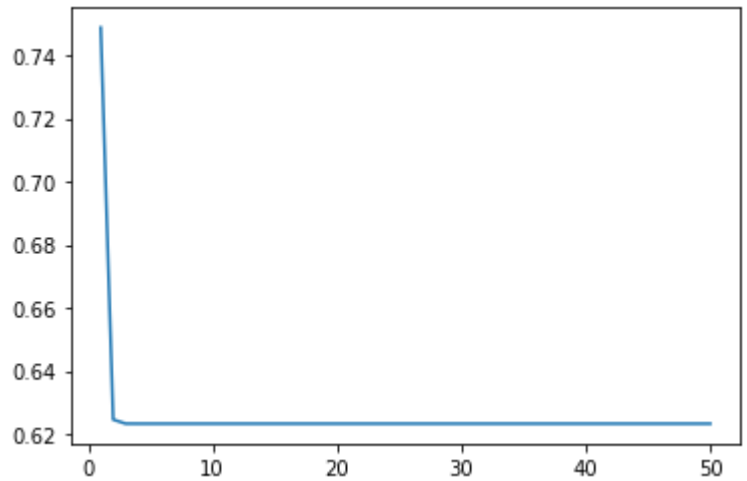
Abaixo temos o plote de dois graficos, onde temos MSE em relação a cada epoca que foi utilizada em **d** e **e**.

O primeiro grafico represenda a relação do MSE com as epocas de **d** e o segundo com o de **e**.

Neste caso podemos observar que o no primeiro grafico o erro diminuiu rapidamente em relação as epocas,a curva do grafico se aproxima de um angulo de noventa graus. Ja no segundo grafico o erro decresceu de forma mais demorada em relação as epocas, como pode ser visto a sua curvatura é bem maior do que a do primeiro grafico.

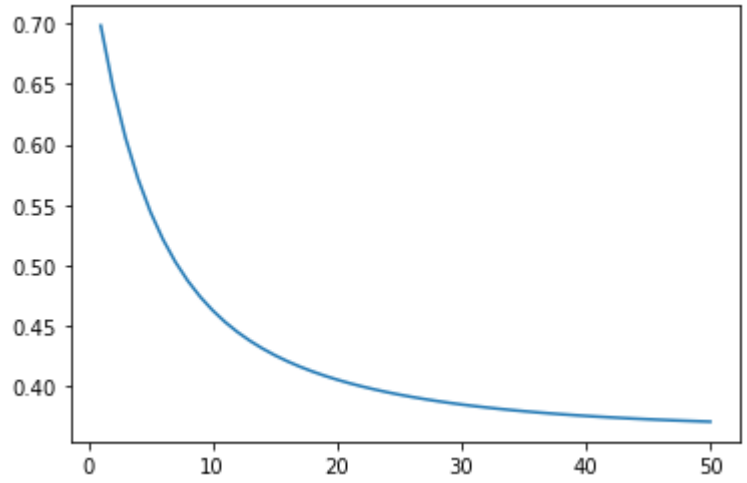
In [18]:

```
pl.plot(d.epocas_, metricas.MSE_(y_train, d.y_predic_epocas))
pl.show()
```



In [19]:

```
pl.plot(e.epocas_, metricas.MSE_(y_train, e.y_predic_epocas))
pl.show()
```

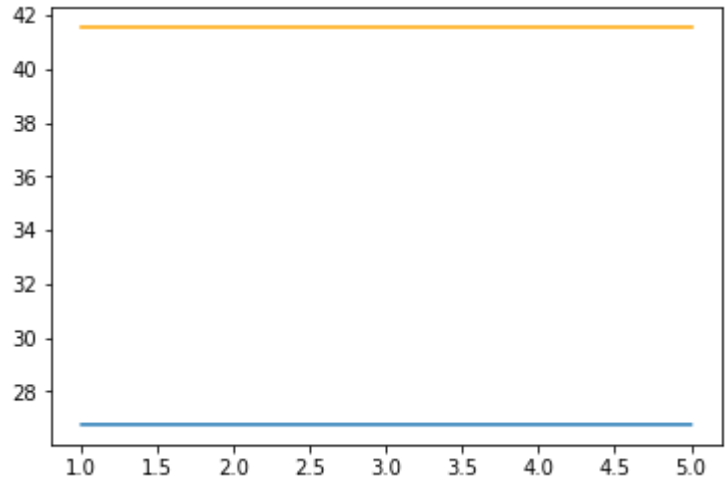


Abaixo é mostrado os graficos para o método **h**, onde temos a relação do MSE com o termo de regularização, onde é dado os valore 1,2,3,4,5 para o termo.

É possivel notar que quando o termo aumenta o erro para as variaveis de treino esta aumetando, porem os erro para os testes estaá diminuindo. Isso implica que para ter uma melhor predição o seria melhor usar o valor 5 para o termo de regularização

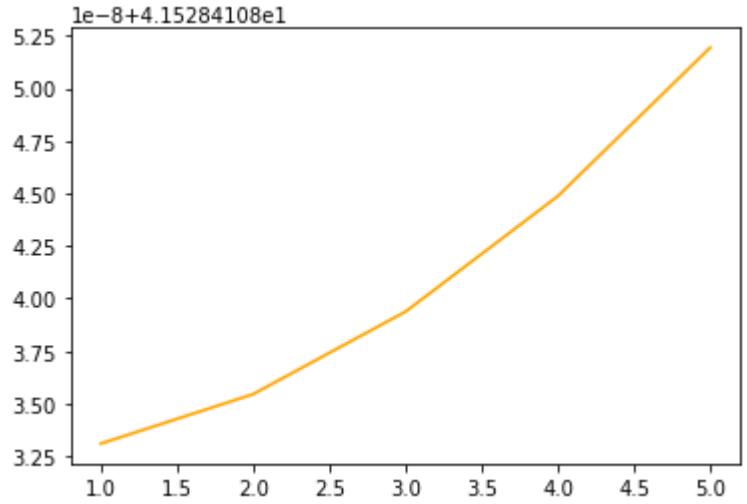
In [41]:

```
termo_de_regularizacao = [1,2,3,4,5]
mse1 = []
mse2 = []
for i in termo_de_regularizacao:
    h.termo_regularizacao = i
    h.fit(X_train, y_train)
    mse1.append(metricas.MSE(y_train , h.predict(X_train)))
    mse2.append(metricas.MSE(y_test , h.predict(X_test)))
pl.plot(termo_de_regularizacao, mse1, color="orange")
pl.plot(termo_de_regularizacao, mse2)
pl.show()
```



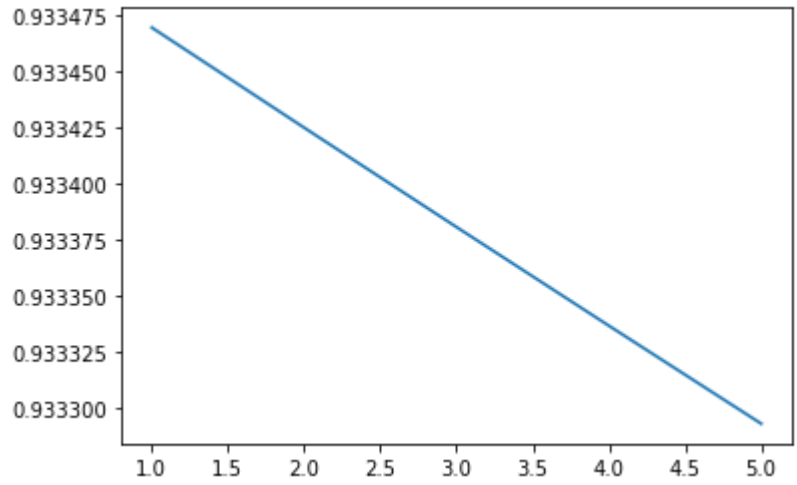
In [42]:

```
pl.plot(termo_de_regularizacao, mse1, color='orange')
pl.show()
```



In [22]:

```
pl.plot(termo_de_regularizacao, mse2)
pl.show()
```



In [ ]:

```

```