Multivariate Regression & Feature Scaling

From Lectures 3/4

ML structures problems around identifying inputs (features) and outputs (labels); the rest is algorithmic details.

- ***supervised learning***
  - make predictions on labels from features
  - (semi- and self-) supervised address data scarcity by principled cheating
- ***unsupervised learning***
  - learn patterns and relationships in data from features
- ***reinforcement learning***
  ***-** agent learns to operate (actions) within environment (state) subject to a reward system*

Supervised ML is indeed fancy curve-fitting

- *the loss function (user-defined) determines what is optimal*
- *most familiar with l2-loss (MSE)*
- *optimization frequently proceeds by taking gradients of loss function with respect to model parameters*

To understand basic mechanics

- ***linear regression***
  - a model that is linear in its parameters (not necessarily linear in output-input relationship)
- ***gradient descent***
  ***-** workhorse in parameter optimization with most other algorithms correcting deficiencies*

Unpacking "Linear Least-Squares" Regression

Linear model

$$f(x) = \sum_{i=0}^{m} \theta_i g_i(x)$$

*A linear model is linear in its **parameters***

Example:

$$m = 2; g_0 = 1; g_1 = x; g_2 = x^2$$

$$f(x) = \theta_0 + \theta_1 x + \theta_2 x^2$$

Least-squares

General notion of optimization

Given $\{(\boldsymbol{x}_i, y_i)\}$ produce "optimal" f $\quad \hat{y} = f(\boldsymbol{x}, \boldsymbol{\theta})$

that minimizes some error metric ("loss") $\quad \mathcal{E}(\{y_k, \hat{y}_k\})$

The "least squares" aspect specifies the loss as related to L2-norm

$$\min_{\boldsymbol{\theta}} \mathcal{E}(f) = \min_{\boldsymbol{\theta}} \sum_{k=1}^{n} |e_k|^2 \qquad \mathcal{E}_2(f) = \sqrt{\frac{1}{n} \sum_{k=1}^{n} |e_k|^2}$$

- **Linear least-squares regression** can be exactly solved in the framework of linear algebra using techniques for matrix inversion/diagonalization (see notes)
- **Machine learning** is distinguished by having *non-linear parametric dependence* and (possibly) *different loss functions;* this necessitates alternatives to pure linear algebra.

Unpacking "non-Linear Least-Squares" Regression

non-Linear model

$$f(x) = \sum_{i=0}^{m} g_i(x, \boldsymbol{\theta})$$

Example:

$$f(x, \boldsymbol{\theta}) = \theta_0 \cos(\theta_1 x + \theta_2) + \theta_3$$

not important that the x is in the cosine, but rather the non-linear relationship amongst thetas is the sticking point

Least-squares

$$\mathcal{E}(\boldsymbol{\theta}) = \sum_{k=1}^{n} (f(x_k, \boldsymbol{\theta}) - y_k)^2; \quad \frac{\partial \mathcal{E}}{\partial \theta_i} = 0 \ \forall \ i$$

$$\implies \sum_{k=1}^{n} (f(x_k, \boldsymbol{\theta}) - y_k) \frac{\partial f}{\partial \theta_i} = 0 \ \forall \ i$$

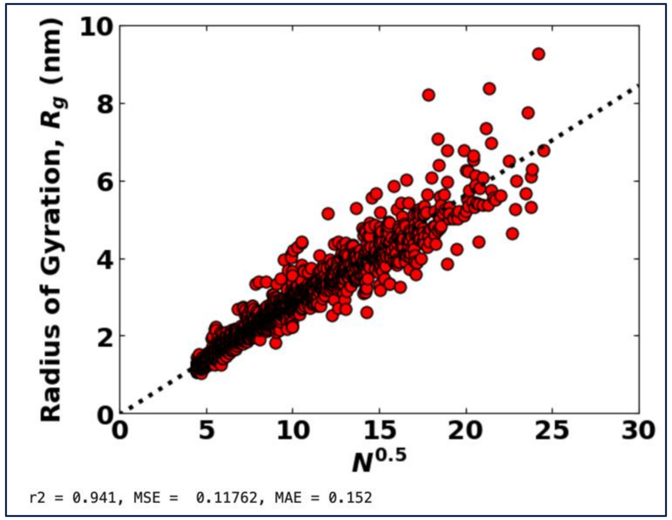
$$\sum_{k=1}^{n} e_k \frac{\partial f}{\partial \theta_i} = 0 \ \forall \ i$$

chain rule is your friend, and this strategy can be applied to other losses

- **Linear least-squares regression** can be exactly solved in the framework of linear algebra using techniques for matrix inversion/diagonalization (see notes)
- **Machine learning** is distinguished by having *non-linear parametric dependence* and (possibly) *different loss functions;* this necessitates alternatives to pure linear algebra. Machine learning models have <u>*multivariable (high-dimensional) inputs*</u>!

Multivariate Regression

Previous example illustrates working with simple bivariate data; there is only one feature per label



While we may often work with one label as output, the labels will be modeled as a function of many variables

$$f(x) = \theta_0 + \theta_1 x$$

$$\boldsymbol{\theta}^{(i+1)} = \boldsymbol{\theta}^{(i)} - \gamma^{(i)} \left[\nabla \mathcal{E}(f(\boldsymbol{\theta}^{(i)}))\right]^T$$

Extension to multiple variables is straightforward

$$f(\boldsymbol{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

Size (ft^2)	No. Units	Capital Cost ($)
2104	3	399900
1600	3	329900
2400	3	369000
1416	2	232000
3000	4	539900

More generally…

$$f(\boldsymbol{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots = \sum_{i=1}^{m} \theta_i x_i; x_0 = 1$$

$$= \boldsymbol{\theta}^T \boldsymbol{x}$$

Notice that nothing related to cost function or gradient descent becomes any more complicated!

Multivariate Regression

$$\{(\boldsymbol{x}_k, y_k)\} \text{ for } k = 1, \cdots, n \qquad f(\boldsymbol{x}) = \boldsymbol{\theta}^T \boldsymbol{x} \qquad \boldsymbol{\theta} \in \mathbb{R}^m$$

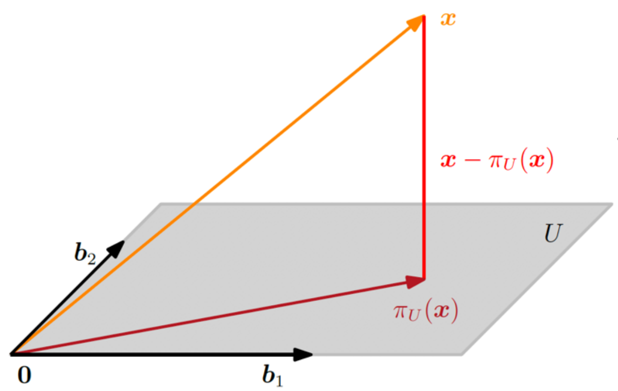
$$\boldsymbol{X} = [\boldsymbol{x}_1 \ \boldsymbol{x}_2 \ \ldots \ \boldsymbol{x}_n]; \ \boldsymbol{y} = [y_1, y_2, \ldots, y_n]^T$$

$$\mathcal{E} = \sum_{k=1}^{m} (\boldsymbol{\theta}^T \boldsymbol{x}_k - y_k)^2$$

$$\mathcal{E} = (\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y})^T (\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y})$$

$$\nabla_{\boldsymbol{\theta}} \mathcal{E} = 0$$

$$\implies \boldsymbol{\theta} = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{y}$$



$$\pi_U(\boldsymbol{x}) = \sum_{i=1}^{M} \lambda_i \boldsymbol{b}_i = \boldsymbol{B}\boldsymbol{\lambda}$$

$$\boldsymbol{\lambda} = (\boldsymbol{B}^T \boldsymbol{B})^{-1} \boldsymbol{B}^T \boldsymbol{x}$$

Recall again our construction of the Normal Equation as the solution to minimizing the projection error onto a subspace

Practice: Multivariate Regression

`ex1data2.csv`

Size (ft^2)	No. Units	Capital Cost ($)
2104	3	399900
1600	3	329900
2400	3	369000
1416	2	232000
3000	4	539900

⋮

Task:

1. Build a regression model with the data provided to estimate capital cost as a function of both size and number of units.
2. Use the model to predict cost for space of 1650 ft^2 and 3 units.
3. Solve the same problem with the normal equation to verify the numerical result

Feature Scaling

`ex1data2.csv`

Size (ft^2)	No. Units	Capital Cost ($)
2104	3	399900
1600	3	329900
2400	3	369000
1416	2	232000
3000	4	539900

⋮

Task:

1. Build a regression model with the data provided to estimate capital cost as a function of both size and number of units.
2. Use the model to predict cost for space of 1650 ft^2 and 3 units.
3. Solve the same problem with the normal equation to verify the numerical result

Spin:

We are going to introduce here the concept of ***feature scaling*** , which is a transformation of data (mapping) such that all values are on similar scales (*usually in the range of -1 to 1 or 0 to 1*):

- is typically good practice for constructing machine learning models but is essential for any algorithms that compute ***distances***; it is not necessary for algorithms that rely on "rules" (**decision trees**)
- facilitates faster convergence during model training (*why?*)
- provides better initial choice over hyperparameters
- avoids unintentional bias of some inputs

Feature Scaling

Common feature scaling approaches

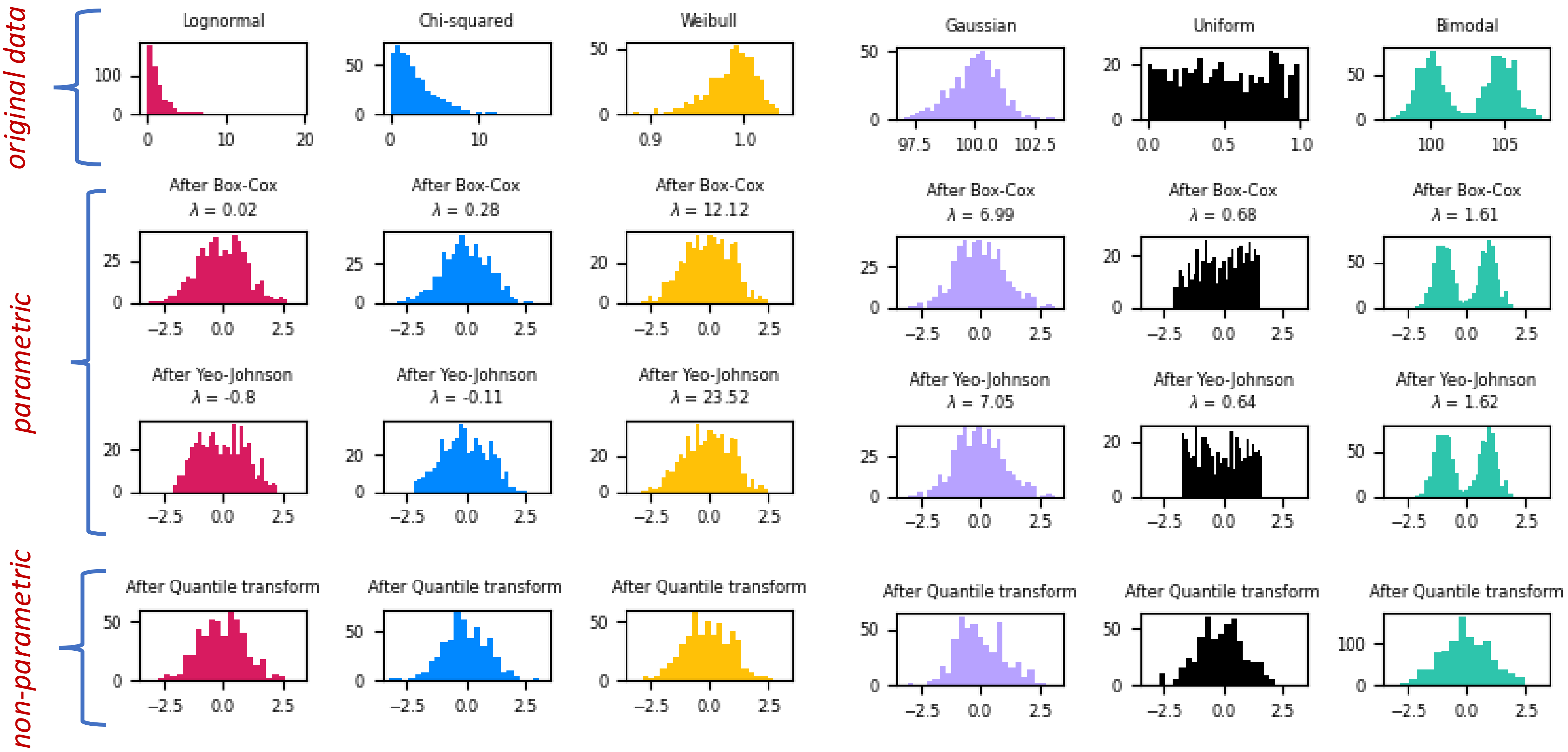
linear mappings

- `Min Max Scaling` $$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$
- `Standard Scaling` $$x' = \frac{x - \mu}{\sigma}$$
- `Max Abs Scaling` $$x' = \frac{x}{\max(|x|)}$$
- `Robust Transformer` $$x' = \frac{x - q_2}{q_3 - q_1}$$

nonlinear mappings

- `Quantile/Rank Transformer` *can map data to <u>uniform or Gaussian distribution</u> (can distort correlations/distances)*
- `Power Transformer` *usually for mapping data to <u>Gaussian distribution</u> (Box-Cox, Yeo-Johnson are different flavors)*
- `Norm Scaling` $$\boldsymbol{x}' = \frac{\boldsymbol{x}}{||\boldsymbol{x}||}$$

Feature Scaling: Comparing Non-Linear Transforms



Feature Scaling: Scikit-learn

Basic Functionality

```
X ← (nsamples,mfeatures)
y ← (nsamples,1)
```

```
from sklearn import preprocessing

Xscaler = preprocessing.TheScaler().fit(X)
X_sc    = Xscaler.transform(X)

yscaler = preprocessing.TheScaler().fit(y)
y_sc    = yscaler.transform(y)

Xagain  = Xscaler.inverse_transform(X_sc)
yagain  = yscaler.inverse_transform(y_sc)
```

Notebook
Exercise

Activity: Premise and Objective

Machine learning excels (over standard methods and human intuition) when confronted with ***high-dimensional*** data. We will explore multivariate regression in the notebook to see how this does not remarkably change problem structure or present major challenges

```
# modules used by Prof. Webb
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from   sklearn import preprocessing
from   scipy.optimize import minimize
```

linear multivariate model: $c(s, n) = \theta_0 + \theta_1 s + \theta_2 n$

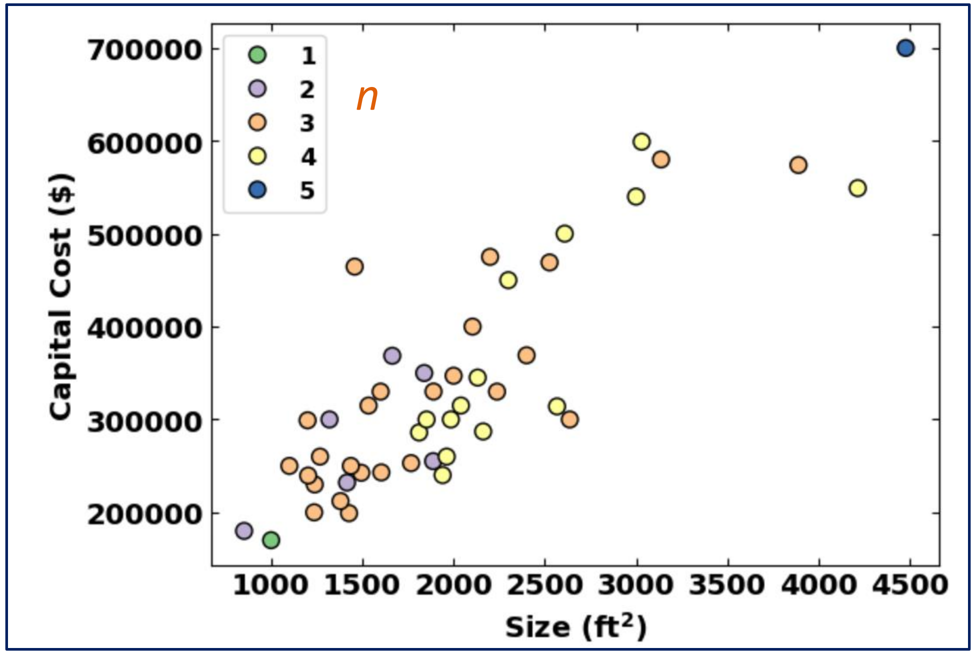
capital cost, c *equipment size, s* *number of units, n*

Progression of the notebook:

- Look at the data
- Perform feature scaling
- Optimize using gradient descent (from scratch)
- Compare to linear-algebraic solutions
- Compare to python optimization

```
Size,Units,Capital
2104,3,399900
1600,3,329900
2400,3,369000
1416,2,232000
3000,4,539900
1985,4,299900
1534,3,314900
1427,3,198999
1380,3,212000
```

⋮



Activity: Data, Scaling, Gradient Descent

```
# extract the data
url_for_data = "https://raw.githubusercontent.com/\
data = pd.read_csv(url_for_data)
print(data.head(5))

# partition into features (X)/ labels (y)
X = np.array(data.iloc[:,:2])
y = np.array(data.iloc[:,2])
```

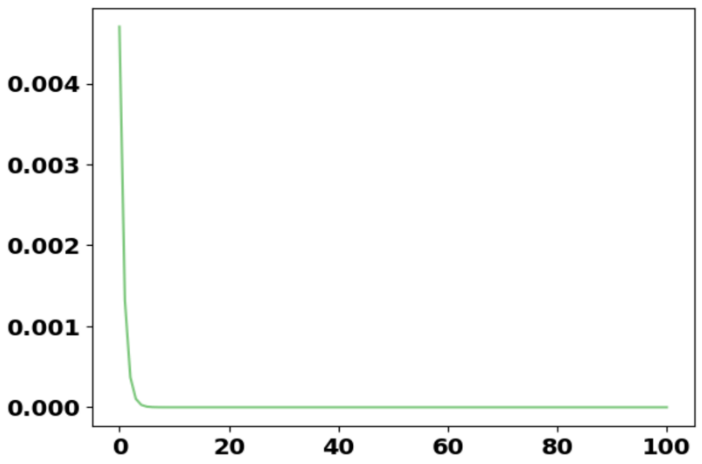
Using scikit-learn to perform variable scaling/transformation

```
# example for scikit learn
X_stdscaler = preprocessing.StandardScaler().fit(X)
X_std       = X_stdscaler.transform(X)
y_stdscaler = preprocessing.StandardScaler().fit(y.reshape(-1,1))
y_std       = y_stdscaler.transform(y.reshape(-1,1))
```

```
def E2loss(ypred,y):
    return np.sum(ypred[:]-y[:])**2/len(y)

def Grad_Descent(X,y,theta,alpha,nIters):
  '''Gradient descent algorithm
  Inputs:
  X = dependent variables, comes in at Nx2
  y = training data, comes in at Nx1
  theta = parameters, comes in as 3x1
  alpha = learning rate
  iters = number of iterations
  Output:
  theta = final parameters
  E = array of cost as a function of iterations
  '''
  n        = len(y) #number of training examples
  features = np.ones((n,len(theta)))
  features[:,1:] = X[:]
  ypred  = features@theta # predictions with current hypothesis

  E_hist = [E2loss(ypred,y)]
  for i in range(nIters):
    e     = ypred[:,0] - y[:,0]
    theta = theta - (alpha*e[:,np.newaxis].T@features).T #
    ypred = features@theta # predictions with current hypothesis
    E_hist.append(E2loss(ypred,y))

  return theta,E_hist
```

```
###BEGIN SOLUTION
theta0 = np.array([[0.01],[0.01],[0.01]])
theta_GD, E_GD = Grad_Descent(X_std,y_std,theta0,0.01, 100)
plt.plot(E_GD[:])

###END SOLUTION
Xtest    = np.array([[1650,3]]) # input
Xtest_sc = X_stdscaler.transform(Xtest)
Ytest_sc = theta_GD[0] + Xtest_sc@theta_GD[1:,0]
Ytest    = y_stdscaler.inverse_transform(Ytest_sc.reshape(-1,1))
print("The optimal parameters found are \n", theta_GD)
print("The prediction from ", Xtest, "is ", Ytest)
```

```
The optimal parameters found are
 [[-7.52436308e-17]
 [ 8.84765988e-01]
 [-5.31788196e-02]]
The prediction from  [[1650    3]] is  [[293081.464336]]
```



Activity: Comparison to Normal Equations

$$c(s,n) = \theta_0 + \theta_1 s + \theta_2 n$$

$$\mathcal{E} = \sum_{k=1}^{m} (\boldsymbol{\theta}^T \boldsymbol{x}_k - y_k)^2$$

$$\mathcal{E} = (\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y})^T (\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y})$$

$$\nabla_{\boldsymbol{\theta}} \mathcal{E} = 0$$

$$\implies \boldsymbol{\theta} = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{y}$$

```python
##Normal Equations

Xtest    = np.array([[1,1650,3]]) # input

###BEGIN SOLUTION

#Set up feature matrix
n        = len(y) #number of training examples
features = np.ones((n,3))
features[:,1:] = X[:]

XXinv = np.linalg.inv(features.T @ features)
theta = XXinv @ features.T @ y

cost = np.dot(Xtest,theta)

print("The optimal parameters found are \n", theta)
print("The prediction from ", Xtest, "is ", cost)
###END SOLUTION
```

```
The optimal parameters found are
 [89597.9095428    139.21067402 -8738.01911233]
The prediction from  [[   1 1650    3]] is  [293081.4643349]
```

Activity: Comparison to Package Optimization

scipy.optimize.minimize

```
scipy.optimize.minimize(fun, x0, args=(), method=None, jac=None, hess=None,
hessp=None, bounds=None, constraints=(), tol=None, callback=None, options=None)
```

Minimization of scalar function of one or more variables. [source]

Parameters: **fun** : *callable*

The objective function to be minimized.

`fun(x, *args) -> float`

where `x` is a 1-D array with shape (n,) and `args` is a tuple of the fixed parameters needed to completely specify the function.

x0 : *ndarray, shape (n,)*

Initial guess. Array of real elements of size (n,), where `n` is the number of independent variables.

args : *tuple, optional*

Extra arguments passed to the objective function and its derivatives (*fun*, *jac* and *hess* functions).

method : *str or callable, optional*

Type of solver. Should be one of

- 'Nelder-Mead' (see here)
- 'Powell' (see here)
- 'CG' (see here)
- 'BFGS' (see here)
- 'Newton-CG' (see here)
- 'L-BFGS-B' (see here)
- 'TNC' (see here)
- 'COBYLA' (see here)
- 'SLSQP' (see here)
- 'trust-constr'(see here)
- 'dogleg' (see here)
- 'trust-ncg' (see here)
- 'trust-exact' (see here)
- 'trust-krylov' (see here)
- custom - a callable object, see below for description.

If not given, chosen to be one of `BFGS`, `L-BFGS-B`, `SLSQP`, depending on whether or not the problem has constraints or bounds.

```python
def E2lossv2(theta,X,y):
    thetaVec = theta.T
    ypred    = theta[0] + X@thetaVec[1:]
    return np.sum(ypred[:]-y[:])**2/len(y)

loss = lambda theta: E2lossv2(theta,X_std,y_std)
theta0 = np.array([8.11502877e-05,8.26550700e-01,4.63616701e-03])
res = minimize(loss,theta0,tol=1e-12,method='BFGS')
theta_py = res.x[:]
Ytest_sc = theta_py[0] + Xtest_sc@theta_py[1:].T
Ytest    = y_stdscaler.inverse_transform(Ytest_sc.reshape(-1,1))
print("The optimal parameters found are \n", theta_py)
print("The prediction from ", Xtest, "is ", Ytest)
```

```
The optimal parameters found are
 [-7.44660311e-09  8.26550700e-01  4.63616701e-03]
The prediction from  [[1650    3]] is  [[294676.60596926]]
```