PROBLEM HW 3

(1.1) $\begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}\begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} a+2b \\ 2a+b \end{bmatrix}$

$\begin{bmatrix} a+2b \\ 2a+b \end{bmatrix} \cdot \begin{bmatrix} a \\ b \end{bmatrix} = a^2 + 4ab + b^2 \geq 0$

this statement is incorrect it can be negative. an example of a positive semidefinite matrix is $\begin{bmatrix} 1 \\ -2 \end{bmatrix}$.

(1.2) (a) looking at each partial

$\frac{\partial}{\partial \theta_{y=1}} = \frac{n_1}{\theta_{y=1}} + \frac{n_1 - n}{1-\theta_{y=1}}$, $\frac{\partial}{\partial \theta_{j|0}} = \sum_{j=1}^{d}\left[r_0^{(j)}/\theta_{j|0} + \left(\frac{r_0^{(i)} + n_1 - n}{1-\theta_{j|0}}\right)\right]$,

and $\frac{\partial}{\partial \theta_{j|1}} = \sum\left(r_1^{(i)}/\theta_{j|1} + \frac{r_1^{(i)} - n_1}{1-\theta_{j|1}}\right)$.

As observed, after one partial derivative with respect to a parameter, θ_p, only θ_p remains, all other terms drop as constants. ∴ driving with respect to another parameter, θ_q, would drop to zero, $\frac{\partial^2}{\partial \theta_p \theta_q} = 0$. QED

(b) $-f(\theta) \Rightarrow$

$\frac{\partial}{\partial \theta_{y=1}} = \frac{-n_1}{\theta_{y=1}} + \frac{n - n_1}{1-\theta_{y=1}}$ $\quad \frac{\partial}{\partial \theta_{j|1}} = \sum\left[\frac{-r_1^{(i)}}{\theta_{j|1}} + \frac{n_1 - r_1^{(i)}}{1-\theta_{j|1}}\right]$

$\frac{\partial^2}{\partial \theta_{y=1}^2} = n_1/\theta_{y=1}^2 + \frac{n-n_1}{(1-\theta_{y=1})^2}$ $\quad \frac{\partial^2}{\partial \theta_{j|1}^2} = \sum\left[\frac{r_1^{(i)}}{\theta_{j|1}^2} + \frac{n_1 - r_1^{(i)}}{(1-\theta_{j|1})^2}\right]$

$\frac{\partial}{\partial \theta_{j|0}} = \sum\left[-r_0^{(j)}/\theta_{j|0} + \frac{n - n_1 - r_0^{(j)}}{1-\theta_{j|0}}\right]$

$\frac{\partial^2}{\partial \theta_{j|0}^2} = \sum\left[r_0^{(j)}/\theta_{j|0} + \frac{n - n_1 - r_0^{(i)}}{(1-\theta_{j|0})^2}\right]$

because the Hessian is a diagonal matrix with elements equal to $\frac{\partial^2}{\partial \theta^2}$, thus we observe that

$x^T H x = \begin{bmatrix} x^{(0)} & x^{(1)} & x^{(2)} \end{bmatrix}\begin{bmatrix} \ddots & & 0 \\ & & \\ 0 & & \ddots \end{bmatrix}\begin{bmatrix} x^{(0)} \\ x^{(1)} \\ x^{(2)} \end{bmatrix}$

$= (x^{(0)})^2 \frac{\partial^2}{\partial \theta_{y=1}^2} + (x^{(1)})^2 \frac{\partial^2}{\partial \theta_{j|0}^2} + (x^{(2)})^2 \frac{\partial^2}{\partial \theta_{j|1}^2}$

because each second partial is ≥ 0, H is positive semidefinite.

(1.3) $D = \{(x_1, y_1), \ldots (x_n, y_n)\}$ where $x_i \in \mathbb{R}^d$, $y_i \in \mathbb{R}$

$$L_D(\tilde{\theta}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \tilde{\theta}^T \tilde{x}_i)^2$$

(a) $\frac{\partial}{\partial \tilde{\theta}} L_D(\tilde{\theta}) = \frac{\partial}{\partial \tilde{\theta}} \frac{1}{n} \sum_{i=1}^{n} (y_i - \tilde{\theta}^T \tilde{x}_i)^2$

$= \sum_{i=1}^{n} -2 x_i (y_i - \tilde{\theta}^T \tilde{x}_i)$

(b) $\frac{\partial^2}{\partial \tilde{\theta}} L_D(\tilde{\theta}) = \sum_{i=1}^{n} -2 \tilde{x}_i (-\tilde{x}_i^T) = \frac{2}{n} \sum \tilde{x}_i \tilde{x}^T$

$\frac{\partial}{\partial \theta_0} L_D(\theta_0) = \frac{\partial}{\partial \theta_0} \frac{1}{n} \sum \theta_0^2 = 2\theta_0 \quad \frac{\partial^2}{\partial \theta_0^2} = 2$

$\frac{\partial}{\partial \theta} L_D(\theta) = \frac{\partial}{\partial \theta} \frac{1}{n} \sum (y_i - \theta^T x_i)^2 = \frac{-2}{n} \sum x_i + x_i (y_i - \theta^T x_i)$

$\frac{\partial^2}{\partial \theta^2} = \frac{+2}{n} \sum x_i x_i^T$

HESSIAN $\Rightarrow \begin{bmatrix} \frac{\partial^2}{\partial \theta_0^2} & \cdots \\ \vdots & \frac{\partial^2}{\partial \theta^2} \cdots \\ \cdots & \frac{\partial^2}{\partial \theta} \end{bmatrix} = \begin{bmatrix} 2 & & \\ & \frac{2}{n} \sum_{i=1}^{n} x_i x_i^T & \\ & & \frac{2}{n} \sum_{i=1}^{n} \tilde{x}_i \tilde{x}_i^T \end{bmatrix}$

Programming Assignment 3: Sentiment analysis

In this programming assignment, you will implement logistic regression to predict the sentiment of reviews that come from `imdb.com` , `amazon.com` , and `yelp.com` . Make sure the notebook is in the same folder that contains `full_set.txt` .

1. Load and preprocess data

In [1]:
```
%matplotlib inline
import string
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
matplotlib.rc('xtick', labelsize=14)
matplotlib.rc('ytick', labelsize=14)
```

The data set consists of 3000 sentences, each labeled '1' (if it came from a positive review) or '0' (if it came from a negative review). To be consistent with our notation from lecture, we will change the negative review label to '-1'.

In [2]:
```
## Read in the data set.
with open("full_set.txt") as f:
    content = f.readlines()

## Remove leading and trailing white space
content = [x.strip() for x in content]

## Separate the sentences from the labels
sentences = [x.split("\t")[0] for x in content]
labels = [x.split("\t")[1] for x in content]

## Transform the labels from '0 v.s. 1' to '-1 v.s. 1'
y = np.array(labels, dtype='int8')
y = 2*y - 1

for i in range(5):
    print ('Label: ',y[i],'; ', sentences[i])
```

```
Label:  -1 ;  So there is no way for me to plug it in here in the US unless I
go by a converter.
Label:  1 ;  Good case, Excellent value.
Label:  1 ;  Great for the jawbone.
Label:  -1 ;  Tied to charger for conversations lasting more than 45 minutes.
MAJOR PROBLEMS!!
Label:  1 ;  The mic is great.
```

Preprocessing the text data

To transform this prediction problem into one amenable to linear classification, we will first need to preprocess the text data. We will do four transformations:

1. Remove punctuation and numbers.
2. Transform all words to lower-case.
3. Remove *stop words*.
4. Convert the sentences into vectors, using a bag-of-words representation.

We begin with first two steps.

In [3]:
```
## full_remove takes a string x and a list of characters removal_list
## returns x with all the characters in removal_list replaced by ' '
def full_remove(x, removal_list):
    for w in removal_list:
        x = x.replace(w, ' ')
    return x

## Remove digits
digits = [str(x) for x in range(10)]
digit_less = [full_remove(x, digits) for x in sentences]

## Remove punctuation
punc_less = [full_remove(x, list(string.punctuation)) for x in digit_less]

## Make everything lower-case
sents_lower = [x.lower() for x in punc_less]
```

Stop words

Stop words are words that are filtered out because they are believed to contain no useful information for the task at hand. These usually include articles such as 'a' and 'the', pronouns such as 'i' and 'they', and prepositions such 'to' and 'from'. We have put together a very small list of stop words, but these are by no means comprehensive. Feel free to use something different; for instance, larger lists can easily be found on the web.

In [4]:
```
## Define our stop words
stop_set = set(['the', 'a', 'an', 'i', 'he', 'she', 'they', 'to', 'of', 'it',
'from'])

## Remove stop words
sents_split = [x.split() for x in sents_lower]
sents_processed = [" ".join(list(filter(lambda a: a not in stop_set, x))) for
x in sents_split]
```

What do the sentences look like so far?

In [5]:
```
sents_processed[0:10]
```

Out[5]:
```
['so there is no way for me plug in here in us unless go by converter',
 'good case excellent value',
 'great for jawbone',
 'tied charger for conversations lasting more than minutes major problems',
 'mic is great',
 'have jiggle plug get line up right get decent volume',
 'if you have several dozen or several hundred contacts then imagine fun send
ing each them one by one',
 'if you are razr owner you must have this',
 'needless say wasted my money',
 'what waste money and time']
```

Bag of words

In order to use linear classifiers on our data set, we need to transform our textual data into numeric data. The classical way to do this is known as the *bag of words* representation. In this representation, each word is thought of as corresponding to a number in `{1, 2, ..., d}` where `d` is the size of our vocabulary. And each sentence is represented as a d-dimensional vector x, where x_i is the number of times that word i occurs in the sentence.

To do this transformation, we will make use of the `CountVectorizer` class in `scikit-learn` (Note that this is the only time you can call an external function from scikit-learn). We will cap the number of features at 4500, meaning a word will make it into our vocabulary only if it is one of the 4500 most common words in the corpus. This is often a useful step as it can weed out spelling mistakes and words which occur too infrequently to be useful.

Task P1: Once you get the bag-of-words representation, append a '1' to the beginning of each vector to allow our linear classifier to learn a bias term. What is the size of the resulting data_mat matrix?

In [6]:
```
from sklearn.feature_extraction.text import CountVectorizer

## Transform to bag of words representation.
vectorizer = CountVectorizer(analyzer = "word", tokenizer = None, preprocessor
= None, stop_words = None, max_features = 4500)
data_features = vectorizer.fit_transform(sents_processed)
print ('The original size: ',data_features.shape)
```

```
The original size:  (3000, 4500)
```

In [7]:
```
## STUDENT: YOUR CODE STARTS HERE
# Task: Append '1' to the beginning of each vector.
# Hint: You can use data_features.toarray() to transform data_features into a
 numpy array
# The output should be a numpy array named data_mat

extra_one = np.ones((3000,1), dtype=float)
data_mat = np.concatenate((extra_one, data_features.toarray()), axis=1)

## STUDENT: CODE ENDS
print ('The updated size: ',data_mat.shape)
```

```
The updated size:  (3000, 4501)
```

Training / test split

Finally, we split the data into a training set of 2500 sentences and a test set of 500 sentences (of which 250 are positive and 250 negative).

In [8]:
```
## Split the data into testing and training sets
np.random.seed(0)
test_inds = np.append(np.random.choice((np.where(y==-1))[0], 250, replace=Fals
e), np.random.choice((np.where(y==1))[0], 250, replace=False))
train_inds = list(set(range(len(labels))) - set(test_inds))

train_data = data_mat[train_inds,]
train_labels = y[train_inds]

test_data = data_mat[test_inds,]
test_labels = y[test_inds]

print("train data: ", train_data.shape)
print("test data: ", test_data.shape)
```

```
train data:  (2500, 4501)
test data:  (500, 4501)
```

2. Fitting a logistic regression model to the training data

In this section, we will implement our own logistic regression solver using gradient descent. As we have seen in the class, to learn the parameters of logistic regression, we need to perform the following optimization:

$$\tilde{\theta}_t = \underset{\tilde{\theta}}{\operatorname{argmin}}\ L_{\mathcal{D}}(\tilde{\theta}) = \underset{\tilde{\theta}}{\operatorname{argmin}} \sum_{i=1}^{n} \ln\left(1 + e^{y_i\ \tilde{\theta}_t^T \tilde{x}_i}\right)$$

where $y_i \in \{-1, +1\}$ is the label, $\tilde{\theta}$ is the vector of coefficients:

$$\tilde{\theta} = [\,\theta_0 \quad \theta_1 \quad \ldots \quad \theta_d\,]^T,$$

and $\tilde{x}$ is the "augmented" feature vector (of $d+1$ dimensions), where we stick a 1 in the front of the original features:

$$\tilde{x} = [\,1 \quad x_1 \quad \ldots \quad x_d\,]^T.$$

There is no nice, closed-form solution like with least-squares linear regression (http://en.wikipedia.org/wiki/Moore%E2%80%93Penrose_pseudoinverse) so we will use gradient descent (http://en.wikipedia.org/wiki/Gradient_descent) instead. Specifically we will use batch gradient descent which calculates the gradient from all data points in the data set. Luckily, the loss function $L_{\mathcal{D}}(\tilde{\theta})$ we want to minimize is convex (http://en.wikipedia.org/wiki/Convex_optimization) so there is only one minimum. Thus the minimum we arrive at is the global minimum.

Gradient descent is a general method and requires twice differentiability for smoothness (http://en.wikipedia.org/wiki/Smooth_function). It updates the parameters using a first-order approximation of the error surface.

$$\tilde{\theta}_{t+1} = \tilde{\theta}_t + \nabla L_{\mathcal{D}}(\tilde{\theta}_t)$$

Task P2: Derive the gradient of the loss $L_{\mathcal{D}}(\tilde{\theta})$ with respect to $\tilde{\theta}$, namely $\nabla L_{\mathcal{D}}(\tilde{\theta}_t)$. The answer should depend on data points (x_i, y_i) for $i = 1, \ldots, n$, and the model parameter $\tilde{\theta}$. Make sure you get the sign correct. Also implement the function `weight_derivative`. Print the output of the code.

In [9]:
```
def weight_derivative(weights, feature_matrix, labels):
    # Input:
    # weights: weight vector w, a numpy vector of dimension d
    # feature_matrix: numpy array of size n by d, where n is the number of dat
a points, and d is the feature dimension
    # labels: true labels y, a numpy vector of dimension d, each with value -1
or +1
    # Output:
    # Derivative of the regression cost function with respect to the weight w,
a numpy array of dimension d

    ## STUDENT: Start of code ###
    derivative_sum = 0.
    for i in range(len(labels)):
        outside = feature_matrix[i] * labels[i]
        inside = np.exp(-1 * labels[i] * np.dot(weights, feature_matrix[i]) )
        derivative_sum += outside * (1 / ( 1 + inside))


    return derivative_sum
    # End of code ###
```

In [10]:
```
# STUDENT: PRINT THE OUTPUT AND COPY IT TO THE SOLUTION FILE
my_weights = np.ones(data_mat.shape[1]) # a weight of all 1s
derivative = weight_derivative(my_weights,train_data,train_labels)

print (derivative[:10])
```

```
[ 1.23407200e+03  9.99999959e-01 -4.24835426e-18 -6.14417460e-06
  1.99987630e+00  2.99985907e+00 -1.79862100e-02  3.49776173e+01
  1.99996625e+00  1.00000000e+00]
```

Now, we can just use the same gradient descent algorithm that we wrote in assignment 2 to solve it.

In [11]:
```
def gradient_descent(feature_matrix, labels, initial_weights, step_size, toler
ance):
    # Gradient descent algorithm for logistic regression problem

    # Input:
    # feature_matrix: numpy array of size n by d, where n is the number of dat
a points, and d is the feature dimension
    # labels: true labels y, a numpy vector of dimension n
    # initial_weights: initial weight vector to start with, a numpy vector of
 dimension d
    # step_size: step size of update
    # tolerance: tolerace epsilon for stopping condition
    # Output:
    # Weights obtained after convergence

    converged = False
    weights = np.array(initial_weights) # current iterate
    i = 0
    while not converged:
        # impelementation of what the gradient descent algorithm does in every
iteration
        # Refer back to the update rule listed above: update the weight
        i += 1
        derivative = weight_derivative(weights, feature_matrix, labels)

        weights -= (step_size * derivative)

        # Compute the gradient magnitude:

        gradient_magnitude = np.sqrt(np.sum(derivative**2))

        # Check the stopping condition to decide whether you want to stop the
 iterations
        # print ("grad mag :", gradient_magnitude)
        #print ("tolerance:", tolerance)

        if gradient_magnitude < tolerance:
            converged = True

        print ("Iteration: ",i,"gradient_magnitude: ", gradient_magnitude) # f
or us to check about convergence

    return(weights)
```

Task P3: Specify the initial_weights, step_size, and tolerance for the function `gradient_descent` . Copy the outputs of the code to the solution file.

In [12]:
```
#Initialize the weights, step size and tolerance
# Start of code
#STUDENT: Specify the initial_weights, step_size, and tolerance
initial_weights =  np.zeros(4501)
step_size = 0.005
tolerance = 5
# end of code

# Use the regression_gradient_descent function to calculate the gradient decen
t and store it in the variable 'final_weights'
final_weights = gradient_descent(train_data,train_labels, initial_weights, ste
p_size, tolerance)

# end of code
print ("Here are the final weights after convergence:")
print (final_weights)
```

```
Iteration:  1 gradient_magnitude:  209.2175900826697
Iteration:  2 gradient_magnitude:  212.64421867541415
Iteration:  3 gradient_magnitude:  584.1017850068945
Iteration:  4 gradient_magnitude:  1256.785468117014
Iteration:  5 gradient_magnitude:  1536.9148922237887
Iteration:  6 gradient_magnitude:  1409.1836051491587
Iteration:  7 gradient_magnitude:  1458.5670556063646
Iteration:  8 gradient_magnitude:  1377.5557284445008
Iteration:  9 gradient_magnitude:  1300.7306897048818
Iteration:  10 gradient_magnitude:  1275.3901995059225
Iteration:  11 gradient_magnitude:  1210.8689930956264
Iteration:  12 gradient_magnitude:  1183.3122667955197
Iteration:  13 gradient_magnitude:  1143.7827656559937
Iteration:  14 gradient_magnitude:  1123.2213579344218
Iteration:  15 gradient_magnitude:  1097.5395865103742
Iteration:  16 gradient_magnitude:  1081.8619393447846
Iteration:  17 gradient_magnitude:  1061.382551300502
Iteration:  18 gradient_magnitude:  1047.2783473847555
Iteration:  19 gradient_magnitude:  1028.33031960631
Iteration:  20 gradient_magnitude:  1014.9038320157325
Iteration:  21 gradient_magnitude:  996.8935564366525
Iteration:  22 gradient_magnitude:  984.1076780231133
Iteration:  23 gradient_magnitude:  967.0239088547238
Iteration:  24 gradient_magnitude:  954.7326610353883
Iteration:  25 gradient_magnitude:  938.4105340162379
Iteration:  26 gradient_magnitude:  926.4127965970251
Iteration:  27 gradient_magnitude:  910.6622678459509
Iteration:  28 gradient_magnitude:  898.7848057155093
Iteration:  29 gradient_magnitude:  883.460324055539
Iteration:  30 gradient_magnitude:  871.5694147684499
Iteration:  31 gradient_magnitude:  856.567704891846
Iteration:  32 gradient_magnitude:  844.5740166434672
Iteration:  33 gradient_magnitude:  829.8284196486136
Iteration:  34 gradient_magnitude:  817.6958386292453
Iteration:  35 gradient_magnitude:  803.1757952524371
Iteration:  36 gradient_magnitude:  790.9223179043909
Iteration:  37 gradient_magnitude:  776.630511513173
Iteration:  38 gradient_magnitude:  764.3140257167513
Iteration:  39 gradient_magnitude:  750.2757801180821
Iteration:  40 gradient_magnitude:  737.9689018841347
Iteration:  41 gradient_magnitude:  724.2151793922668
Iteration:  42 gradient_magnitude:  711.9793910578471
Iteration:  43 gradient_magnitude:  698.5296323784795
Iteration:  44 gradient_magnitude:  686.3980477603345
Iteration:  45 gradient_magnitude:  673.2493328386929
Iteration:  46 gradient_magnitude:  661.2221773920269
Iteration:  47 gradient_magnitude:  648.3476589986032
Iteration:  48 gradient_magnitude:  636.39876527527
Iteration:  49 gradient_magnitude:  623.7539787363194
Iteration:  50 gradient_magnitude:  611.8430140899121
Iteration:  51 gradient_magnitude:  599.3758986510311
Iteration:  52 gradient_magnitude:  587.4606109236814
Iteration:  53 gradient_magnitude:  575.1208277085286
Iteration:  54 gradient_magnitude:  563.1656094899296
Iteration:  55 gradient_magnitude:  550.9104947902235
Iteration:  56 gradient_magnitude:  538.8903583561545
Iteration:  57 gradient_magnitude:  526.6871727888521
```

```
Iteration:  58 gradient_magnitude:  514.5882505547553
Iteration:  59 gradient_magnitude:  502.4139463958696
Iteration:  60 gradient_magnitude:  490.232290830791
Iteration:  61 gradient_magnitude:  478.07237403375234
Iteration:  62 gradient_magnitude:  465.81249656443856
Iteration:  63 gradient_magnitude:  453.66010817673754
Iteration:  64 gradient_magnitude:  441.33404720918537
Iteration:  65 gradient_magnitude:  429.189750161746
Iteration:  66 gradient_magnitude:  416.81690556319415
Iteration:  67 gradient_magnitude:  404.6891726768411
Iteration:  68 gradient_magnitude:  392.2967629528769
Iteration:  69 gradient_magnitude:  380.20285727512345
Iteration:  70 gradient_magnitude:  367.8266013077073
Iteration:  71 gradient_magnitude:  355.7933744864988
Iteration:  72 gradient_magnitude:  343.4778696356579
Iteration:  73 gradient_magnitude:  331.5419771529183
Iteration:  74 gradient_magnitude:  319.3402843144899
Iteration:  75 gradient_magnitude:  307.547451069085
Iteration:  76 gradient_magnitude:  295.5196197702691
Iteration:  77 gradient_magnitude:  283.9228620287215
Iteration:  78 gradient_magnitude:  272.1334663315304
Iteration:  79 gradient_magnitude:  260.79049122631613
Iteration:  80 gradient_magnitude:  249.30556587089515
Iteration:  81 gradient_magnitude:  238.27580186415088
Iteration:  82 gradient_magnitude:  227.15973799616583
Iteration:  83 gradient_magnitude:  216.50151181862836
Iteration:  84 gradient_magnitude:  205.81443658149323
Iteration:  85 gradient_magnitude:  195.58269942865454
Iteration:  86 gradient_magnitude:  185.37866330248985
Iteration:  87 gradient_magnitude:  175.6234522413175
Iteration:  88 gradient_magnitude:  165.949485027442
Iteration:  89 gradient_magnitude:  156.71508388863742
Iteration:  90 gradient_magnitude:  147.6109607604349
Iteration:  91 gradient_magnitude:  138.93557487588703
Iteration:  92 gradient_magnitude:  130.43402011517728
Iteration:  93 gradient_magnitude:  122.34972918882
Iteration:  94 gradient_magnitude:  114.47678260333875
Iteration:  95 gradient_magnitude:  107.00956998066466
Iteration:  96 gradient_magnitude:  99.7849149498983
Iteration:  97 gradient_magnitude:  92.95465087529884
Iteration:  98 gradient_magnitude:  86.39180487249278
Iteration:  99 gradient_magnitude:  80.21214450534154
Iteration:  100 gradient_magnitude:  74.3184991614723
Iteration:  101 gradient_magnitude:  68.79670932423879
Iteration:  102 gradient_magnitude:  63.57344575338471
Iteration:  103 gradient_magnitude:  58.710172824901846
Iteration:  104 gradient_magnitude:  54.15204784162417
Iteration:  105 gradient_magnitude:  49.94096659989784
Iteration:  106 gradient_magnitude:  46.03585808105781
Iteration:  107 gradient_magnitude:  42.463000360195224
Iteration:  108 gradient_magnitude:  39.190941857163686
Iteration:  109 gradient_magnitude:  36.23335805792003
Iteration:  110 gradient_magnitude:  33.5647082585785
Iteration:  111 gradient_magnitude:  31.18816887291432
Iteration:  112 gradient_magnitude:  29.080848312511723
Iteration:  113 gradient_magnitude:  27.236912176292073
Iteration:  114 gradient_magnitude:  25.63364343732374
```

```
Iteration:  115 gradient_magnitude:  24.257755625097204
Iteration:  116 gradient_magnitude:  23.08567152954206
Iteration:  117 gradient_magnitude:  22.099042437343662
Iteration:  118 gradient_magnitude:  21.274205496622386
Iteration:  119 gradient_magnitude:  20.59102839853628
Iteration:  120 gradient_magnitude:  20.027728082421643
Iteration:  121 gradient_magnitude:  19.56561037098103
Iteration:  122 gradient_magnitude:  19.186604935644556
Iteration:  123 gradient_magnitude:  18.87551562261523
Iteration:  124 gradient_magnitude:  18.61875917197336
Iteration:  125 gradient_magnitude:  18.40522000962763
Iteration:  126 gradient_magnitude:  18.22550672671518
Iteration:  127 gradient_magnitude:  18.072132886222924
Iteration:  128 gradient_magnitude:  17.939031838719774
Iteration:  129 gradient_magnitude:  17.821478528163816
Iteration:  130 gradient_magnitude:  17.715764573370677
Iteration:  131 gradient_magnitude:  17.619060008863656
Iteration:  132 gradient_magnitude:  17.529199605740764
Iteration:  133 gradient_magnitude:  17.444560587446784
Iteration:  134 gradient_magnitude:  17.363926788294005
Iteration:  135 gradient_magnitude:  17.286400953639944
Iteration:  136 gradient_magnitude:  17.211321593940664
Iteration:  137 gradient_magnitude:  17.13820676372649
Iteration:  138 gradient_magnitude:  17.06670508791991
Iteration:  139 gradient_magnitude:  16.996562377607162
Iteration:  140 gradient_magnitude:  16.927593858762798
Iteration:  141 gradient_magnitude:  16.859665518711925
Iteration:  142 gradient_magnitude:  16.792678929177484
Iteration:  143 gradient_magnitude:  16.7265613666857
Iteration:  144 gradient_magnitude:  16.661257804776092
Iteration:  145 gradient_magnitude:  16.59672593834993
Iteration:  146 gradient_magnitude:  16.53293209198693
Iteration:  147 gradient_magnitude:  16.469848834954345
Iteration:  148 gradient_magnitude:  16.40745294637519
Iteration:  149 gradient_magnitude:  16.34572432735134
Iteration:  150 gradient_magnitude:  16.284645008023983
Iteration:  151 gradient_magnitude:  16.22419867511495
Iteration:  152 gradient_magnitude:  16.16437019030453
Iteration:  153 gradient_magnitude:  16.10514539367519
Iteration:  154 gradient_magnitude:  16.046510866880503
Iteration:  155 gradient_magnitude:  15.988453852754946
Iteration:  156 gradient_magnitude:  15.930962133994601
Iteration:  157 gradient_magnitude:  15.87402399820974
Iteration:  158 gradient_magnitude:  15.817628171145916
Iteration:  159 gradient_magnitude:  15.761763797976469
Iteration:  160 gradient_magnitude:  15.706420402823937
Iteration:  161 gradient_magnitude:  15.651587875259974
Iteration:  162 gradient_magnitude:  15.59725644310102
Iteration:  163 gradient_magnitude:  15.543416660465736
Iteration:  164 gradient_magnitude:  15.4900593877519
Iteration:  165 gradient_magnitude:  15.437175780320262
Iteration:  166 gradient_magnitude:  15.384757272702528
Iteration:  167 gradient_magnitude:  15.33279556783401
Iteration:  168 gradient_magnitude:  15.281282623986028
Iteration:  169 gradient_magnitude:  15.230210644651683
Iteration:  170 gradient_magnitude:  15.179572067381049
Iteration:  171 gradient_magnitude:  15.129359554404584
```

```
Iteration:  172 gradient_magnitude:  15.079565982890676
Iteration:  173 gradient_magnitude:  15.030184436335034
Iteration:  174 gradient_magnitude:  14.9812081959398
Iteration:  175 gradient_magnitude:  14.932630732752292
Iteration:  176 gradient_magnitude:  14.884445699964864
Iteration:  177 gradient_magnitude:  14.836646925759693
Iteration:  178 gradient_magnitude:  14.789228406386957
Iteration:  179 gradient_magnitude:  14.742184299660112
Iteration:  180 gradient_magnitude:  14.695508918705679
Iteration:  181 gradient_magnitude:  14.649196726050144
Iteration:  182 gradient_magnitude:  14.603242327957602
Iteration:  183 gradient_magnitude:  14.557640469051073
Iteration:  184 gradient_magnitude:  14.512386027169724
Iteration:  185 gradient_magnitude:  14.467474008471374
Iteration:  186 gradient_magnitude:  14.422899542752038
Iteration:  187 gradient_magnitude:  14.378657878981288
Iteration:  188 gradient_magnitude:  14.334744381035113
Iteration:  189 gradient_magnitude:  14.291154523620724
Iteration:  190 gradient_magnitude:  14.247883888380223
Iteration:  191 gradient_magnitude:  14.204928160166121
Iteration:  192 gradient_magnitude:  14.162283123478563
Iteration:  193 gradient_magnitude:  14.119944659057047
Iteration:  194 gradient_magnitude:  14.077908740618357
Iteration:  195 gradient_magnitude:  14.0361714317338
Iteration:  196 gradient_magnitude:  13.994728882838668
Iteration:  197 gradient_magnitude:  13.953577328367645
Iteration:  198 gradient_magnitude:  13.9127130840099
Iteration:  199 gradient_magnitude:  13.872132544078209
Iteration:  200 gradient_magnitude:  13.831832178986582
Iteration:  201 gradient_magnitude:  13.791808532831256
Iteration:  202 gradient_magnitude:  13.752058221070152
Iteration:  203 gradient_magnitude:  13.712577928296206
Iteration:  204 gradient_magnitude:  13.67336440610013
Iteration:  205 gradient_magnitude:  13.63441447101851
Iteration:  206 gradient_magnitude:  13.595725002563249
Iteration:  207 gradient_magnitude:  13.557292941328651
Iteration:  208 gradient_magnitude:  13.519115287172589
Iteration:  209 gradient_magnitude:  13.4811890974684
Iteration:  210 gradient_magnitude:  13.443511485424283
Iteration:  211 gradient_magnitude:  13.406079618467201
Iteration:  212 gradient_magnitude:  13.36889071668839
Iteration:  213 gradient_magnitude:  13.331942051347713
Iteration:  214 gradient_magnitude:  13.295230943434289
Iteration:  215 gradient_magnitude:  13.258754762280883
Iteration:  216 gradient_magnitude:  13.222510924229741
Iteration:  217 gradient_magnitude:  13.186496891347577
Iteration:  218 gradient_magnitude:  13.150710170187615
Iteration:  219 gradient_magnitude:  13.115148310596648
Iteration:  220 gradient_magnitude:  13.079808904565159
Iteration:  221 gradient_magnitude:  13.044689585118686
Iteration:  222 gradient_magnitude:  13.009788025248662
Iteration:  223 gradient_magnitude:  12.975101936881057
Iteration:  224 gradient_magnitude:  12.940629069881224
Iteration:  225 gradient_magnitude:  12.90636721109345
Iteration:  226 gradient_magnitude:  12.872314183413737
Iteration:  227 gradient_magnitude:  12.83846784489444
Iteration:  228 gradient_magnitude:  12.804826087879452
```

```
Iteration:  229 gradient_magnitude:  12.771386838168674
Iteration:  230 gradient_magnitude:  12.738148054210564
Iteration:  231 gradient_magnitude:  12.705107726321613
Iteration:  232 gradient_magnitude:  12.672263875931698
Iteration:  233 gradient_magnitude:  12.6396145548542
Iteration:  234 gradient_magnitude:  12.60715784457995
Iteration:  235 gradient_magnitude:  12.574891855594025
Iteration:  236 gradient_magnitude:  12.542814726714477
Iteration:  237 gradient_magnitude:  12.510924624452157
Iteration:  238 gradient_magnitude:  12.479219742390756
Iteration:  239 gradient_magnitude:  12.447698300586337
Iteration:  240 gradient_magnitude:  12.41635854498553
Iteration:  241 gradient_magnitude:  12.385198746861706
Iteration:  242 gradient_magnitude:  12.354217202268433
Iteration:  243 gradient_magnitude:  12.323412231509545
Iteration:  244 gradient_magnitude:  12.292782178625183
Iteration:  245 gradient_magnitude:  12.262325410893217
Iteration:  246 gradient_magnitude:  12.23204031834546
Iteration:  247 gradient_magnitude:  12.201925313298126
Iteration:  248 gradient_magnitude:  12.17197882989598
Iteration:  249 gradient_magnitude:  12.142199323669699
Iteration:  250 gradient_magnitude:  12.112585271105942
Iteration:  251 gradient_magnitude:  12.083135169229662
Iteration:  252 gradient_magnitude:  12.053847535198212
Iteration:  253 gradient_magnitude:  12.024720905906841
Iteration:  254 gradient_magnitude:  11.995753837605122
Iteration:  255 gradient_magnitude:  11.966944905523967
Iteration:  256 gradient_magnitude:  11.938292703512834
Iteration:  257 gradient_magnitude:  11.909795843686748
Iteration:  258 gradient_magnitude:  11.881452956082828
Iteration:  259 gradient_magnitude:  11.853262688325957
Iteration:  260 gradient_magnitude:  11.82522370530328
Iteration:  261 gradient_magnitude:  11.79733468884725
Iteration:  262 gradient_magnitude:  11.769594337426893
Iteration:  263 gradient_magnitude:  11.74200136584705
Iteration:  264 gradient_magnitude:  11.714554504955279
Iteration:  265 gradient_magnitude:  11.687252501356223
Iteration:  266 gradient_magnitude:  11.660094117133122
Iteration:  267 gradient_magnitude:  11.633078129576296
Iteration:  268 gradient_magnitude:  11.60620333091832
Iteration:  269 gradient_magnitude:  11.579468528075711
Iteration:  270 gradient_magnitude:  11.55287254239689
Iteration:  271 gradient_magnitude:  11.526414209416217
Iteration:  272 gradient_magnitude:  11.500092378613926
Iteration:  273 gradient_magnitude:  11.473905913181746
Iteration:  274 gradient_magnitude:  11.447853689794043
Iteration:  275 gradient_magnitude:  11.421934598384304
Iteration:  276 gradient_magnitude:  11.39614754192681
Iteration:  277 gradient_magnitude:  11.370491436223311
Iteration:  278 gradient_magnitude:  11.344965209694568
Iteration:  279 gradient_magnitude:  11.319567803176618
Iteration:  280 gradient_magnitude:  11.294298169721607
Iteration:  281 gradient_magnitude:  11.269155274403055
Iteration:  282 gradient_magnitude:  11.244138094125423
Iteration:  283 gradient_magnitude:  11.219245617437867
Iteration:  284 gradient_magnitude:  11.19447684435202
Iteration:  285 gradient_magnitude:  11.169830786163732
```

```
Iteration:  286 gradient_magnitude:  11.145306465278626
Iteration:  287 gradient_magnitude:  11.120902915041349
Iteration:  288 gradient_magnitude:  11.096619179568437
Iteration:  289 gradient_magnitude:  11.0724543135847
Iteration:  290 gradient_magnitude:  11.048407382262969
Iteration:  291 gradient_magnitude:  11.024477461067203
Iteration:  292 gradient_magnitude:  11.000663635598764
Iteration:  293 gradient_magnitude:  10.976965001445864
Iteration:  294 gradient_magnitude:  10.953380664036015
Iteration:  295 gradient_magnitude:  10.929909738491489
Iteration:  296 gradient_magnitude:  10.906551349487607
Iteration:  297 gradient_magnitude:  10.88330463111387
Iteration:  298 gradient_magnitude:  10.86016872673781
Iteration:  299 gradient_magnitude:  10.837142788871489
Iteration:  300 gradient_magnitude:  10.814225979040614
Iteration:  301 gradient_magnitude:  10.79141746765614
Iteration:  302 gradient_magnitude:  10.768716433888367
Iteration:  303 gradient_magnitude:  10.746122065543398
Iteration:  304 gradient_magnitude:  10.723633558941959
Iteration:  305 gradient_magnitude:  10.701250118800488
Iteration:  306 gradient_magnitude:  10.678970958114428
Iteration:  307 gradient_magnitude:  10.656795298043704
Iteration:  308 gradient_magnitude:  10.63472236780031
Iteration:  309 gradient_magnitude:  10.612751404537931
Iteration:  310 gradient_magnitude:  10.59088165324362
Iteration:  311 gradient_magnitude:  10.569112366631389
Iteration:  312 gradient_magnitude:  10.547442805037745
Iteration:  313 gradient_magnitude:  10.525872236319092
Iteration:  314 gradient_magnitude:  10.504399935750945
Iteration:  315 gradient_magnitude:  10.483025185928943
Iteration:  316 gradient_magnitude:  10.461747276671595
Iteration:  317 gradient_magnitude:  10.440565504924727
Iteration:  318 gradient_magnitude:  10.419479174667593
Iteration:  319 gradient_magnitude:  10.398487596820623
Iteration:  320 gradient_magnitude:  10.37759008915473
Iteration:  321 gradient_magnitude:  10.3567859762022
Iteration:  322 gradient_magnitude:  10.336074589169092
Iteration:  323 gradient_magnitude:  10.315455265849101
Iteration:  324 gradient_magnitude:  10.294927350538904
Iteration:  325 gradient_magnitude:  10.274490193954913
Iteration:  326 gradient_magnitude:  10.254143153151404
Iteration:  327 gradient_magnitude:  10.233885591440037
Iteration:  328 gradient_magnitude:  10.21371687831067
Iteration:  329 gradient_magnitude:  10.193636389353502
Iteration:  330 gradient_magnitude:  10.17364350618248
Iteration:  331 gradient_magnitude:  10.15373761635995
Iteration:  332 gradient_magnitude:  10.133918113322544
Iteration:  333 gradient_magnitude:  10.114184396308245
Iteration:  334 gradient_magnitude:  10.094535870284636
Iteration:  335 gradient_magnitude:  10.074971945878291
Iteration:  336 gradient_magnitude:  10.055492039305296
Iteration:  337 gradient_magnitude:  10.036095572302862
Iteration:  338 gradient_magnitude:  10.016781972062025
Iteration:  339 gradient_magnitude:  9.997550671161404
Iteration:  340 gradient_magnitude:  9.97840110750198
Iteration:  341 gradient_magnitude:  9.95933272424292
Iteration:  342 gradient_magnitude:  9.940344969738366
```

```
Iteration:  343 gradient_magnitude:  9.921437297475215
Iteration:  344 gradient_magnitude:  9.902609166011866
Iteration:  345 gradient_magnitude:  9.883860038917874
Iteration:  346 gradient_magnitude:  9.865189384714565
Iteration:  347 gradient_magnitude:  9.846596676816517
Iteration:  348 gradient_magnitude:  9.82808139347395
Iteration:  349 gradient_magnitude:  9.809643017715976
Iteration:  350 gradient_magnitude:  9.791281037294695
Iteration:  351 gradient_magnitude:  9.772994944630144
Iteration:  352 gradient_magnitude:  9.754784236756041
Iteration:  353 gradient_magnitude:  9.736648415266366
Iteration:  354 gradient_magnitude:  9.718586986262697
Iteration:  355 gradient_magnitude:  9.700599460302342
Iteration:  356 gradient_magnitude:  9.682685352347237
Iteration:  357 gradient_magnitude:  9.664844181713573
Iteration:  358 gradient_magnitude:  9.647075472022166
Iteration:  359 gradient_magnitude:  9.629378751149547
Iteration:  360 gradient_magnitude:  9.611753551179753
Iteration:  361 gradient_magnitude:  9.594199408356827
Iteration:  362 gradient_magnitude:  9.576715863037975
Iteration:  363 gradient_magnitude:  9.55930245964742
Iteration:  364 gradient_magnitude:  9.54195874663088
Iteration:  365 gradient_magnitude:  9.52468427641074
Iteration:  366 gradient_magnitude:  9.507478605341793
Iteration:  367 gradient_magnitude:  9.490341293667676
Iteration:  368 gradient_magnitude:  9.473271905477851
Iteration:  369 gradient_magnitude:  9.456270008665243
Iteration:  370 gradient_magnitude:  9.439335174884425
Iteration:  371 gradient_magnitude:  9.422466979510414
Iteration:  372 gradient_magnitude:  9.405665001598024
Iteration:  373 gradient_magnitude:  9.388928823841775
Iteration:  374 gradient_magnitude:  9.372258032536367
Iteration:  375 gradient_magnitude:  9.355652217537683
Iteration:  376 gradient_magnitude:  9.339110972224331
Iteration:  377 gradient_magnitude:  9.322633893459706
Iteration:  378 gradient_magnitude:  9.306220581554566
Iteration:  379 gradient_magnitude:  9.289870640230115
Iteration:  380 gradient_magnitude:  9.273583676581586
Iteration:  381 gradient_magnitude:  9.257359301042301
Iteration:  382 gradient_magnitude:  9.24119712734823
Iteration:  383 gradient_magnitude:  9.225096772502999
Iteration:  384 gradient_magnitude:  9.209057856743387
Iteration:  385 gradient_magnitude:  9.19308000350526
Iteration:  386 gradient_magnitude:  9.177162839389965
Iteration:  387 gradient_magnitude:  9.161305994131162
Iteration:  388 gradient_magnitude:  9.14550910056209
Iteration:  389 gradient_magnitude:  9.129771794583265
Iteration:  390 gradient_magnitude:  9.114093715130588
Iteration:  391 gradient_magnitude:  9.098474504143876
Iteration:  392 gradient_magnitude:  9.082913806535803
Iteration:  393 gradient_magnitude:  9.067411270161225
Iteration:  394 gradient_magnitude:  9.051966545786911
Iteration:  395 gradient_magnitude:  9.036579287061661
Iteration:  396 gradient_magnitude:  9.021249150486804
Iteration:  397 gradient_magnitude:  9.005975795387064
Iteration:  398 gradient_magnitude:  8.990758883881803
Iteration:  399 gradient_magnitude:  8.975598080856622
```

```
Iteration:  400 gradient_magnitude:  8.960493053935329
Iteration:  401 gradient_magnitude:  8.945443473452235
Iteration:  402 gradient_magnitude:  8.930449012424834
Iteration:  403 gradient_magnitude:  8.915509346526783
Iteration:  404 gradient_magnitude:  8.90062415406124
Iteration:  405 gradient_magnitude:  8.885793115934531
Iteration:  406 gradient_magnitude:  8.871015915630137
Iteration:  407 gradient_magnitude:  8.856292239182997
Iteration:  408 gradient_magnitude:  8.841621775154136
Iteration:  409 gradient_magnitude:  8.827004214605594
Iteration:  410 gradient_magnitude:  8.812439251075663
Iteration:  411 gradient_magnitude:  8.797926580554423
Iteration:  412 gradient_magnitude:  8.783465901459575
Iteration:  413 gradient_magnitude:  8.769056914612571
Iteration:  414 gradient_magnitude:  8.754699323215013
Iteration:  415 gradient_magnitude:  8.740392832825352
Iteration:  416 gradient_magnitude:  8.726137151335863
Iteration:  417 gradient_magnitude:  8.71193198894988
Iteration:  418 gradient_magnitude:  8.697777058159316
Iteration:  419 gradient_magnitude:  8.68367207372244
Iteration:  420 gradient_magnitude:  8.669616752641918
Iteration:  421 gradient_magnitude:  8.655610814143113
Iteration:  422 gradient_magnitude:  8.641653979652634
Iteration:  423 gradient_magnitude:  8.627745972777145
Iteration:  424 gradient_magnitude:  8.613886519282403
Iteration:  425 gradient_magnitude:  8.600075347072556
Iteration:  426 gradient_magnitude:  8.586312186169673
Iteration:  427 gradient_magnitude:  8.572596768693508
Iteration:  428 gradient_magnitude:  8.558928828841497
Iteration:  429 gradient_magnitude:  8.54530810286898
Iteration:  430 gradient_magnitude:  8.531734329069666
Iteration:  431 gradient_magnitude:  8.518207247756289
Iteration:  432 gradient_magnitude:  8.504726601241517
Iteration:  433 gradient_magnitude:  8.49129213381904
Iteration:  434 gradient_magnitude:  8.477903591744912
Iteration:  435 gradient_magnitude:  8.46456072321907
Iteration:  436 gradient_magnitude:  8.451263278367062
Iteration:  437 gradient_magnitude:  8.438011009222008
Iteration:  438 gradient_magnitude:  8.42480366970672
Iteration:  439 gradient_magnitude:  8.411641015616052
Iteration:  440 gradient_magnitude:  8.398522804599434
Iteration:  441 gradient_magnitude:  8.385448796143592
Iteration:  442 gradient_magnitude:  8.372418751555461
Iteration:  443 gradient_magnitude:  8.3594324339453
Iteration:  444 gradient_magnitude:  8.346489608209964
Iteration:  445 gradient_magnitude:  8.333590041016386
Iteration:  446 gradient_magnitude:  8.32073350078521
Iteration:  447 gradient_magnitude:  8.307919757674632
Iteration:  448 gradient_magnitude:  8.295148583564384
Iteration:  449 gradient_magnitude:  8.282419752039914
Iteration:  450 gradient_magnitude:  8.269733038376726
Iteration:  451 gradient_magnitude:  8.257088219524885
Iteration:  452 gradient_magnitude:  8.244485074093692
Iteration:  453 gradient_magnitude:  8.23192338233652
Iteration:  454 gradient_magnitude:  8.21940292613581
Iteration:  455 gradient_magnitude:  8.206923488988233
Iteration:  456 gradient_magnitude:  8.19448485598999
```

```
Iteration:  457 gradient_magnitude:  8.18208681382229
Iteration:  458 gradient_magnitude:  8.169729150736963
Iteration:  459 gradient_magnitude:  8.15741165654223
Iteration:  460 gradient_magnitude:  8.145134122588626
Iteration:  461 gradient_magnitude:  8.132896341755051
Iteration:  462 gradient_magnitude:  8.120698108434997
Iteration:  463 gradient_magnitude:  8.108539218522887
Iteration:  464 gradient_magnitude:  8.096419469400573
Iteration:  465 gradient_magnitude:  8.084338659923965
Iteration:  466 gradient_magnitude:  8.0722965904098
Iteration:  467 gradient_magnitude:  8.060293062622554
Iteration:  468 gradient_magnitude:  8.048327879761468
Iteration:  469 gradient_magnitude:  8.036400846447732
Iteration:  470 gradient_magnitude:  8.024511768711777
Iteration:  471 gradient_magnitude:  8.012660453980716
Iteration:  472 gradient_magnitude:  8.000846711065886
Iteration:  473 gradient_magnitude:  7.989070350150549
Iteration:  474 gradient_magnitude:  7.977331182777691
Iteration:  475 gradient_magnitude:  7.965629021837951
Iteration:  476 gradient_magnitude:  7.953963681557677
Iteration:  477 gradient_magnitude:  7.942334977487101
Iteration:  478 gradient_magnitude:  7.9307427264886226
Iteration:  479 gradient_magnitude:  7.9191867467252175
Iteration:  480 gradient_magnitude:  7.907666857648964
Iteration:  481 gradient_magnitude:  7.896182879989675
Iteration:  482 gradient_magnitude:  7.884734635743653
Iteration:  483 gradient_magnitude:  7.8733219481625465
Iteration:  484 gradient_magnitude:  7.861944641742322
Iteration:  485 gradient_magnitude:  7.850602542212346
Iteration:  486 gradient_magnitude:  7.839295476524576
Iteration:  487 gradient_magnitude:  7.828023272842843
Iteration:  488 gradient_magnitude:  7.816785760532266
Iteration:  489 gradient_magnitude:  7.805582770148743
Iteration:  490 gradient_magnitude:  7.794414133428565
Iteration:  491 gradient_magnitude:  7.78327968327812
Iteration:  492 gradient_magnitude:  7.772179253763701
Iteration:  493 gradient_magnitude:  7.761112680101418
Iteration:  494 gradient_magnitude:  7.750079798647193
Iteration:  495 gradient_magnitude:  7.739080446886877
Iteration:  496 gradient_magnitude:  7.72811446342644
Iteration:  497 gradient_magnitude:  7.717181687982266
Iteration:  498 gradient_magnitude:  7.706281961371539
Iteration:  499 gradient_magnitude:  7.695415125502732
Iteration:  500 gradient_magnitude:  7.684581023366167
Iteration:  501 gradient_magnitude:  7.6737794990246915
Iteration:  502 gradient_magnitude:  7.663010397604421
Iteration:  503 gradient_magnitude:  7.652273565285586
Iteration:  504 gradient_magnitude:  7.641568849293466
Iteration:  505 gradient_magnitude:  7.630896097889398
Iteration:  506 gradient_magnitude:  7.620255160361888
Iteration:  507 gradient_magnitude:  7.609645887017794
Iteration:  508 gradient_magnitude:  7.599068129173603
Iteration:  509 gradient_magnitude:  7.588521739146786
Iteration:  510 gradient_magnitude:  7.578006570247235
Iteration:  511 gradient_magnitude:  7.567522476768785
Iteration:  512 gradient_magnitude:  7.557069313980816
Iteration:  513 gradient_magnitude:  7.546646938119935
```

```
Iteration:  514 gradient_magnitude:  7.536255206381732
Iteration:  515 gradient_magnitude:  7.525893976912624
Iteration:  516 gradient_magnitude:  7.5155631088017705
Iteration:  517 gradient_magnitude:  7.505262462073063
Iteration:  518 gradient_magnitude:  7.494991897677196
Iteration:  519 gradient_magnitude:  7.484751277483817
Iteration:  520 gradient_magnitude:  7.474540464273733
Iteration:  521 gradient_magnitude:  7.464359321731215
Iteration:  522 gradient_magnitude:  7.454207714436354
Iteration:  523 gradient_magnitude:  7.444085507857505
Iteration:  524 gradient_magnitude:  7.433992568343792
Iteration:  525 gradient_magnitude:  7.423928763117691
Iteration:  526 gradient_magnitude:  7.413893960267676
Iteration:  527 gradient_magnitude:  7.4038880287409405
Iteration:  528 gradient_magnitude:  7.393910838336184
Iteration:  529 gradient_magnitude:  7.383962259696468
Iteration:  530 gradient_magnitude:  7.374042164302144
Iteration:  531 gradient_magnitude:  7.3641504244638325
Iteration:  532 gradient_magnitude:  7.3542869133154865
Iteration:  533 gradient_magnitude:  7.344451504807513
Iteration:  534 gradient_magnitude:  7.334644073699957
Iteration:  535 gradient_magnitude:  7.324864495555749
Iteration:  536 gradient_magnitude:  7.315112646734019
Iteration:  537 gradient_magnitude:  7.3053884043834785
Iteration:  538 gradient_magnitude:  7.295691646435851
Iteration:  539 gradient_magnitude:  7.286022251599377
Iteration:  540 gradient_magnitude:  7.276380099352374
Iteration:  541 gradient_magnitude:  7.266765069936857
Iteration:  542 gradient_magnitude:  7.257177044352218
Iteration:  543 gradient_magnitude:  7.247615904348975
Iteration:  544 gradient_magnitude:  7.238081532422558
Iteration:  545 gradient_magnitude:  7.228573811807179
Iteration:  546 gradient_magnitude:  7.219092626469741
Iteration:  547 gradient_magnitude:  7.209637861103809
Iteration:  548 gradient_magnitude:  7.200209401123645
Iteration:  549 gradient_magnitude:  7.190807132658282
Iteration:  550 gradient_magnitude:  7.181430942545677
Iteration:  551 gradient_magnitude:  7.172080718326892
Iteration:  552 gradient_magnitude:  7.162756348240359
Iteration:  553 gradient_magnitude:  7.1534577212161645
Iteration:  554 gradient_magnitude:  7.144184726870425
Iteration:  555 gradient_magnitude:  7.134937255499679
Iteration:  556 gradient_magnitude:  7.125715198075363
Iteration:  557 gradient_magnitude:  7.116518446238315
Iteration:  558 gradient_magnitude:  7.107346892293343
Iteration:  559 gradient_magnitude:  7.098200429203841
Iteration:  560 gradient_magnitude:  7.089078950586453
Iteration:  561 gradient_magnitude:  7.079982350705791
Iteration:  562 gradient_magnitude:  7.070910524469198
Iteration:  563 gradient_magnitude:  7.061863367421565
Iteration:  564 gradient_magnitude:  7.052840775740188
Iteration:  565 gradient_magnitude:  7.043842646229688
Iteration:  566 gradient_magnitude:  7.034868876316957
Iteration:  567 gradient_magnitude:  7.025919364046174
Iteration:  568 gradient_magnitude:  7.016994008073853
Iteration:  569 gradient_magnitude:  7.0080927076639385
Iteration:  570 gradient_magnitude:  6.9992153626829525
```

```
Iteration:  571 gradient_magnitude:  6.990361873595185
Iteration:  572 gradient_magnitude:  6.981532141457922
Iteration:  573 gradient_magnitude:  6.972726067916732
Iteration:  574 gradient_magnitude:  6.963943555200784
Iteration:  575 gradient_magnitude:  6.955184506118213
Iteration:  576 gradient_magnitude:  6.946448824051534
Iteration:  577 gradient_magnitude:  6.93773641295309
Iteration:  578 gradient_magnitude:  6.929047177340547
Iteration:  579 gradient_magnitude:  6.9203810222924345
Iteration:  580 gradient_magnitude:  6.911737853443723
Iteration:  581 gradient_magnitude:  6.903117576981437
Iteration:  582 gradient_magnitude:  6.8945200996403235
Iteration:  583 gradient_magnitude:  6.885945328698551
Iteration:  584 gradient_magnitude:  6.877393171973444
Iteration:  585 gradient_magnitude:  6.868863537817273
Iteration:  586 gradient_magnitude:  6.860356335113067
Iteration:  587 gradient_magnitude:  6.851871473270477
Iteration:  588 gradient_magnitude:  6.843408862221667
Iteration:  589 gradient_magnitude:  6.834968412417255
Iteration:  590 gradient_magnitude:  6.826550034822285
Iteration:  591 gradient_magnitude:  6.818153640912231
Iteration:  592 gradient_magnitude:  6.809779142669058
Iteration:  593 gradient_magnitude:  6.80142645257729
Iteration:  594 gradient_magnitude:  6.793095483620143
Iteration:  595 gradient_magnitude:  6.784786149275674
Iteration:  596 gradient_magnitude:  6.776498363512979
Iteration:  597 gradient_magnitude:  6.768232040788414
Iteration:  598 gradient_magnitude:  6.759987096041864
Iteration:  599 gradient_magnitude:  6.751763444693035
Iteration:  600 gradient_magnitude:  6.743561002637785
Iteration:  601 gradient_magnitude:  6.735379686244495
Iteration:  602 gradient_magnitude:  6.727219412350463
Iteration:  603 gradient_magnitude:  6.719080098258336
Iteration:  604 gradient_magnitude:  6.710961661732576
Iteration:  605 gradient_magnitude:  6.7028640209959605
Iteration:  606 gradient_magnitude:  6.694787094726107
Iteration:  607 gradient_magnitude:  6.686730802052041
Iteration:  608 gradient_magnitude:  6.678695062550784
Iteration:  609 gradient_magnitude:  6.670679796243984
Iteration:  610 gradient_magnitude:  6.662684923594569
Iteration:  611 gradient_magnitude:  6.654710365503435
Iteration:  612 gradient_magnitude:  6.64675604330616
Iteration:  613 gradient_magnitude:  6.638821878769757
Iteration:  614 gradient_magnitude:  6.630907794089449
Iteration:  615 gradient_magnitude:  6.6230137118854735
Iteration:  616 gradient_magnitude:  6.615139555199928
Iteration:  617 gradient_magnitude:  6.607285247493622
Iteration:  618 gradient_magnitude:  6.599450712642984
Iteration:  619 gradient_magnitude:  6.591635874936973
Iteration:  620 gradient_magnitude:  6.583840659074038
Iteration:  621 gradient_magnitude:  6.576064990159096
Iteration:  622 gradient_magnitude:  6.5683087937005284
Iteration:  623 gradient_magnitude:  6.560571995607229
Iteration:  624 gradient_magnitude:  6.552854522185652
Iteration:  625 gradient_magnitude:  6.545156300136908
Iteration:  626 gradient_magnitude:  6.537477256553872
Iteration:  627 gradient_magnitude:  6.5298173189183295
```

```
Iteration:  628 gradient_magnitude:  6.522176415098139
Iteration:  629 gradient_magnitude:  6.51455447334442
Iteration:  630 gradient_magnitude:  6.506951422288781
Iteration:  631 gradient_magnitude:  6.499367190940554
Iteration:  632 gradient_magnitude:  6.491801708684062
Iteration:  633 gradient_magnitude:  6.484254905275917
Iteration:  634 gradient_magnitude:  6.476726710842333
Iteration:  635 gradient_magnitude:  6.469217055876466
Iteration:  636 gradient_magnitude:  6.461725871235782
Iteration:  637 gradient_magnitude:  6.454253088139445
Iteration:  638 gradient_magnitude:  6.4467986381657285
Iteration:  639 gradient_magnitude:  6.439362453249452
Iteration:  640 gradient_magnitude:  6.4319444656794404
Iteration:  641 gradient_magnitude:  6.424544608096005
Iteration:  642 gradient_magnitude:  6.417162813488447
Iteration:  643 gradient_magnitude:  6.409799015192584
Iteration:  644 gradient_magnitude:  6.402453146888301
Iteration:  645 gradient_magnitude:  6.395125142597118
Iteration:  646 gradient_magnitude:  6.387814936679786
Iteration:  647 gradient_magnitude:  6.380522463833894
Iteration:  648 gradient_magnitude:  6.373247659091514
Iteration:  649 gradient_magnitude:  6.365990457816851
Iteration:  650 gradient_magnitude:  6.3587507957039175
Iteration:  651 gradient_magnitude:  6.351528608774239
Iteration:  652 gradient_magnitude:  6.344323833374565
Iteration:  653 gradient_magnitude:  6.3371364061746105
Iteration:  654 gradient_magnitude:  6.329966264164815
Iteration:  655 gradient_magnitude:  6.32281334465412
Iteration:  656 gradient_magnitude:  6.315677585267766
Iteration:  657 gradient_magnitude:  6.308558923945109
Iteration:  658 gradient_magnitude:  6.301457298937461
Iteration:  659 gradient_magnitude:  6.294372648805941
Iteration:  660 gradient_magnitude:  6.2873049124193505
Iteration:  661 gradient_magnitude:  6.2802540289520685
Iteration:  662 gradient_magnitude:  6.273219937881964
Iteration:  663 gradient_magnitude:  6.2662025789883185
Iteration:  664 gradient_magnitude:  6.259201892349783
Iteration:  665 gradient_magnitude:  6.2522178183423405
Iteration:  666 gradient_magnitude:  6.245250297637284
Iteration:  667 gradient_magnitude:  6.238299271199229
Iteration:  668 gradient_magnitude:  6.23136468028412
Iteration:  669 gradient_magnitude:  6.224446466437275
Iteration:  670 gradient_magnitude:  6.217544571491435
Iteration:  671 gradient_magnitude:  6.210658937564833
Iteration:  672 gradient_magnitude:  6.203789507059283
Iteration:  673 gradient_magnitude:  6.1969362226582785
Iteration:  674 gradient_magnitude:  6.190099027325116
Iteration:  675 gradient_magnitude:  6.18327786430103
Iteration:  676 gradient_magnitude:  6.176472677103343
Iteration:  677 gradient_magnitude:  6.16968340952363
Iteration:  678 gradient_magnitude:  6.1629100056259105
Iteration:  679 gradient_magnitude:  6.156152409744838
Iteration:  680 gradient_magnitude:  6.149410566483918
Iteration:  681 gradient_magnitude:  6.142684420713739
Iteration:  682 gradient_magnitude:  6.135973917570216
Iteration:  683 gradient_magnitude:  6.129279002452846
Iteration:  684 gradient_magnitude:  6.122599621022989
```

```
Iteration:  685 gradient_magnitude:  6.11593571920215
Iteration:  686 gradient_magnitude:  6.109287243170287
Iteration:  687 gradient_magnitude:  6.1026541393641285
Iteration:  688 gradient_magnitude:  6.096036354475504
Iteration:  689 gradient_magnitude:  6.08943383544969
Iteration:  690 gradient_magnitude:  6.082846529483772
Iteration:  691 gradient_magnitude:  6.076274384025018
Iteration:  692 gradient_magnitude:  6.06971734676926
Iteration:  693 gradient_magnitude:  6.063175365659306
Iteration:  694 gradient_magnitude:  6.056648388883344
Iteration:  695 gradient_magnitude:  6.050136364873377
Iteration:  696 gradient_magnitude:  6.043639242303661
Iteration:  697 gradient_magnitude:  6.037156970089158
Iteration:  698 gradient_magnitude:  6.030689497384004
Iteration:  699 gradient_magnitude:  6.024236773579986
Iteration:  700 gradient_magnitude:  6.01779874830504
Iteration:  701 gradient_magnitude:  6.0113753714217495
Iteration:  702 gradient_magnitude:  6.0049665930258636
Iteration:  703 gradient_magnitude:  5.998572363444828
Iteration:  704 gradient_magnitude:  5.9921926332363284
Iteration:  705 gradient_magnitude:  5.985827353186837
Iteration:  706 gradient_magnitude:  5.979476474310184
Iteration:  707 gradient_magnitude:  5.973139947846131
Iteration:  708 gradient_magnitude:  5.966817725258964
Iteration:  709 gradient_magnitude:  5.960509758236086
Iteration:  710 gradient_magnitude:  5.954215998686639
Iteration:  711 gradient_magnitude:  5.947936398740115
Iteration:  712 gradient_magnitude:  5.941670910745001
Iteration:  713 gradient_magnitude:  5.935419487267419
Iteration:  714 gradient_magnitude:  5.929182081089781
Iteration:  715 gradient_magnitude:  5.9229586452094605
Iteration:  716 gradient_magnitude:  5.9167491328374675
Iteration:  717 gradient_magnitude:  5.910553497397136
Iteration:  718 gradient_magnitude:  5.904371692522828
Iteration:  719 gradient_magnitude:  5.898203672058637
Iteration:  720 gradient_magnitude:  5.892049390057114
Iteration:  721 gradient_magnitude:  5.885908800777992
Iteration:  722 gradient_magnitude:  5.879781858686931
Iteration:  723 gradient_magnitude:  5.873668518454267
Iteration:  724 gradient_magnitude:  5.867568734953771
Iteration:  725 gradient_magnitude:  5.861482463261423
Iteration:  726 gradient_magnitude:  5.855409658654186
Iteration:  727 gradient_magnitude:  5.849350276608802
Iteration:  728 gradient_magnitude:  5.843304272800591
Iteration:  729 gradient_magnitude:  5.837271603102253
Iteration:  730 gradient_magnitude:  5.831252223582696
Iteration:  731 gradient_magnitude:  5.825246090505855
Iteration:  732 gradient_magnitude:  5.819253160329536
Iteration:  733 gradient_magnitude:  5.813273389704257
Iteration:  734 gradient_magnitude:  5.807306735472104
Iteration:  735 gradient_magnitude:  5.801353154665597
Iteration:  736 gradient_magnitude:  5.795412604506562
Iteration:  737 gradient_magnitude:  5.78948504240501
Iteration:  738 gradient_magnitude:  5.783570425958033
Iteration:  739 gradient_magnitude:  5.777668712948698
Iteration:  740 gradient_magnitude:  5.771779861344955
Iteration:  741 gradient_magnitude:  5.765903829298555
```

```
Iteration:  742 gradient_magnitude:  5.760040575143977
Iteration:  743 gradient_magnitude:  5.754190057397353
Iteration:  744 gradient_magnitude:  5.748352234755416
Iteration:  745 gradient_magnitude:  5.7425270660944445
Iteration:  746 gradient_magnitude:  5.736714510469224
Iteration:  747 gradient_magnitude:  5.730914527112007
Iteration:  748 gradient_magnitude:  5.725127075431492
Iteration:  749 gradient_magnitude:  5.719352115011798
Iteration:  750 gradient_magnitude:  5.7135896056114595
Iteration:  751 gradient_magnitude:  5.707839507162418
Iteration:  752 gradient_magnitude:  5.702101779769027
Iteration:  753 gradient_magnitude:  5.696376383707068
Iteration:  754 gradient_magnitude:  5.690663279422762
Iteration:  755 gradient_magnitude:  5.684962427531802
Iteration:  756 gradient_magnitude:  5.679273788818382
Iteration:  757 gradient_magnitude:  5.6735973242342475
Iteration:  758 gradient_magnitude:  5.667932994897729
Iteration:  759 gradient_magnitude:  5.66228076209281
Iteration:  760 gradient_magnitude:  5.6566405872681855
Iteration:  761 gradient_magnitude:  5.651012432036327
Iteration:  762 gradient_magnitude:  5.645396258172568
Iteration:  763 gradient_magnitude:  5.6397920276141775
Iteration:  764 gradient_magnitude:  5.634199702459459
Iteration:  765 gradient_magnitude:  5.628619244966842
Iteration:  766 gradient_magnitude:  5.623050617553986
Iteration:  767 gradient_magnitude:  5.617493782796892
Iteration:  768 gradient_magnitude:  5.61194870342902
Iteration:  769 gradient_magnitude:  5.60641534234041
Iteration:  770 gradient_magnitude:  5.600893662576814
Iteration:  771 gradient_magnitude:  5.595383627338829
Iteration:  772 gradient_magnitude:  5.589885199981046
Iteration:  773 gradient_magnitude:  5.584398344011189
Iteration:  774 gradient_magnitude:  5.578923023089279
Iteration:  775 gradient_magnitude:  5.57345920102679
Iteration:  776 gradient_magnitude:  5.568006841785817
Iteration:  777 gradient_magnitude:  5.56256590947825
Iteration:  778 gradient_magnitude:  5.557136368364956
Iteration:  779 gradient_magnitude:  5.551718182854957
Iteration:  780 gradient_magnitude:  5.546311317504628
Iteration:  781 gradient_magnitude:  5.540915737016889
Iteration:  782 gradient_magnitude:  5.535531406240412
Iteration:  783 gradient_magnitude:  5.530158290168826
Iteration:  784 gradient_magnitude:  5.524796353939933
Iteration:  785 gradient_magnitude:  5.519445562834926
Iteration:  786 gradient_magnitude:  5.514105882277618
Iteration:  787 gradient_magnitude:  5.508777277833666
Iteration:  788 gradient_magnitude:  5.5034597152098135
Iteration:  789 gradient_magnitude:  5.49815316025313
Iteration:  790 gradient_magnitude:  5.492857578950258
Iteration:  791 gradient_magnitude:  5.487572937426665
Iteration:  792 gradient_magnitude:  5.482299201945902
Iteration:  793 gradient_magnitude:  5.477036338908865
Iteration:  794 gradient_magnitude:  5.471784314853066
Iteration:  795 gradient_magnitude:  5.4665430964519
Iteration:  796 gradient_magnitude:  5.461312650513934
Iteration:  797 gradient_magnitude:  5.456092943982177
Iteration:  798 gradient_magnitude:  5.450883943933379
```

```
Iteration:  799 gradient_magnitude:  5.445685617577321
Iteration:  800 gradient_magnitude:  5.440497932256111
Iteration:  801 gradient_magnitude:  5.435320855443489
Iteration:  802 gradient_magnitude:  5.430154354744134
Iteration:  803 gradient_magnitude:  5.424998397892982
Iteration:  804 gradient_magnitude:  5.419852952754531
Iteration:  805 gradient_magnitude:  5.414717987322179
Iteration:  806 gradient_magnitude:  5.409593469717538
Iteration:  807 gradient_magnitude:  5.4044793681897705
Iteration:  808 gradient_magnitude:  5.399375651114929
Iteration:  809 gradient_magnitude:  5.3942822869952884
Iteration:  810 gradient_magnitude:  5.389199244458697
Iteration:  811 gradient_magnitude:  5.384126492257928
Iteration:  812 gradient_magnitude:  5.379063999270024
Iteration:  813 gradient_magnitude:  5.374011734495666
Iteration:  814 gradient_magnitude:  5.368969667058529
Iteration:  815 gradient_magnitude:  5.363937766204653
Iteration:  816 gradient_magnitude:  5.358916001301811
Iteration:  817 gradient_magnitude:  5.353904341838887
Iteration:  818 gradient_magnitude:  5.348902757425254
Iteration:  819 gradient_magnitude:  5.34391121779016
Iteration:  820 gradient_magnitude:  5.3389296927821155
Iteration:  821 gradient_magnitude:  5.33395815236828
Iteration:  822 gradient_magnitude:  5.328996566633868
Iteration:  823 gradient_magnitude:  5.32404490578154
Iteration:  824 gradient_magnitude:  5.319103140130815
Iteration:  825 gradient_magnitude:  5.314171240117469
Iteration:  826 gradient_magnitude:  5.309249176292956
Iteration:  827 gradient_magnitude:  5.304336919323818
Iteration:  828 gradient_magnitude:  5.299434439991106
Iteration:  829 gradient_magnitude:  5.294541709189808
Iteration:  830 gradient_magnitude:  5.289658697928269
Iteration:  831 gradient_magnitude:  5.284785377327627
Iteration:  832 gradient_magnitude:  5.279921718621248
Iteration:  833 gradient_magnitude:  5.275067693154159
Iteration:  834 gradient_magnitude:  5.270223272382498
Iteration:  835 gradient_magnitude:  5.2653884278729555
Iteration:  836 gradient_magnitude:  5.260563131302224
Iteration:  837 gradient_magnitude:  5.2557473544564495
Iteration:  838 gradient_magnitude:  5.250941069230692
Iteration:  839 gradient_magnitude:  5.246144247628386
Iteration:  840 gradient_magnitude:  5.241356861760796
Iteration:  841 gradient_magnitude:  5.236578883846493
Iteration:  842 gradient_magnitude:  5.231810286210816
Iteration:  843 gradient_magnitude:  5.2270510412853595
Iteration:  844 gradient_magnitude:  5.222301121607432
Iteration:  845 gradient_magnitude:  5.217560499819553
Iteration:  846 gradient_magnitude:  5.212829148668925
Iteration:  847 gradient_magnitude:  5.208107041006932
Iteration:  848 gradient_magnitude:  5.203394149788616
Iteration:  849 gradient_magnitude:  5.198690448072184
Iteration:  850 gradient_magnitude:  5.193995909018495
Iteration:  851 gradient_magnitude:  5.189310505890566
Iteration:  852 gradient_magnitude:  5.184634212053072
Iteration:  853 gradient_magnitude:  5.179967000971853
Iteration:  854 gradient_magnitude:  5.175308846213425
Iteration:  855 gradient_magnitude:  5.170659721444488
```

```
Iteration:  856 gradient_magnitude:  5.166019600431445
Iteration:  857 gradient_magnitude:  5.161388457039925
Iteration:  858 gradient_magnitude:  5.15676626523429
Iteration:  859 gradient_magnitude:  5.152152999077176
Iteration:  860 gradient_magnitude:  5.14754863272901
Iteration:  861 gradient_magnitude:  5.1429531404475455
Iteration:  862 gradient_magnitude:  5.138366496587391
Iteration:  863 gradient_magnitude:  5.133788675599552
Iteration:  864 gradient_magnitude:  5.129219652030965
Iteration:  865 gradient_magnitude:  5.124659400524042
Iteration:  866 gradient_magnitude:  5.120107895816213
Iteration:  867 gradient_magnitude:  5.115565112739477
Iteration:  868 gradient_magnitude:  5.111031026219948
Iteration:  869 gradient_magnitude:  5.106505611277411
Iteration:  870 gradient_magnitude:  5.101988843024876
Iteration:  871 gradient_magnitude:  5.097480696668141
Iteration:  872 gradient_magnitude:  5.092981147505342
Iteration:  873 gradient_magnitude:  5.088490170926529
Iteration:  874 gradient_magnitude:  5.084007742413227
Iteration:  875 gradient_magnitude:  5.079533837538002
Iteration:  876 gradient_magnitude:  5.07506843196404
Iteration:  877 gradient_magnitude:  5.070611501444713
Iteration:  878 gradient_magnitude:  5.066163021823163
Iteration:  879 gradient_magnitude:  5.0617229690318775
Iteration:  880 gradient_magnitude:  5.057291319092269
Iteration:  881 gradient_magnitude:  5.052868048114266
Iteration:  882 gradient_magnitude:  5.048453132295893
Iteration:  883 gradient_magnitude:  5.044046547922864
Iteration:  884 gradient_magnitude:  5.039648271368172
Iteration:  885 gradient_magnitude:  5.035258279091685
Iteration:  886 gradient_magnitude:  5.03087654763974
Iteration:  887 gradient_magnitude:  5.026503053644745
Iteration:  888 gradient_magnitude:  5.022137773824776
Iteration:  889 gradient_magnitude:  5.017780684983185
Iteration:  890 gradient_magnitude:  5.013431764008205
Iteration:  891 gradient_magnitude:  5.009090987872555
Iteration:  892 gradient_magnitude:  5.004758333633055
Iteration:  893 gradient_magnitude:  5.000433778430235
Iteration:  894 gradient_magnitude:  4.9961172994879535
Here are the final weights after convergence:
[ 0.2964133  -0.1068523   0.11042425 ...  0.         0.18507002
  0.26978528]
```

Task P4: Write the code to extract the y-intercept θ_0 and the rest of the parameters $\theta = [\,\theta_1 \quad \dots \quad \theta_d\,]^T$. Copy the code and print the outputs.

In [13]:
```
## STUDENT: CODE STARTS HERE
## Pull out the parameters (theta_0, theta) of the logistic regression model
theta0 = final_weights[0]
theta = final_weights[1:]
## STUDENT: CODE ENDS HERE
print ('y intercept: ',theta0)
print ('theta1 and theta2: ',theta[1],theta[2])
```

```
y intercept:  0.2964132952656125
theta1 and theta2:  0.11042424734280551 0.9606256296966943
```

Recall that the logistic/sigmoid function (http://en.wikipedia.org/wiki/Logistic_function) is given by

$$f(z) = \frac{1}{1 + e^{-z}}$$

Based on the logistic model, the posterior probability

$$P(y \mid \mathbf{x}) = f(y\ \tilde{\theta}^T \tilde{\mathbf{x}})$$

To make a prediction, we can simply choose the label $y \in \{-1, +1\}$ with the higher posterior probability.

Task P5: Write the code to make the prediction for a given data matrix. Report the training error and test error.

In [14]:
```
def model_predict(feature_matrix,weights):
# Prediction made by logistic regression

    # Input:
    # feature_matrix: numpy array of size n by d+1, where n is the number of d
ata points, and d+1 is the feature dimension
    #                 note we have included the dummy feature as the first col
umn of the feature_matrix
    # weights: weight vector to start with, a numpy vector of dimension d+1
    # Output:
    # labels: predicted labels, a numpy vector of dimension n

    ## STUDENT: YOUR CODE HERE
    predicted_labels = np.zeros(len(feature_matrix[0]))

    for i in range(len(feature_matrix)):
        inside = np.dot(weights, feature_matrix[i])
        predict_pos = 1 / (1 + np.exp(-1. * inside))
        predict_neg = 1 / (1 + np.exp(inside))
        predicted_labels[i] = 1 if predict_pos > predict_neg else 0

    return predicted_labels
    ## STUDENT: CODE ENDS
```

In [15]:
```
# STUDENT: copy the output of this section to the solution file

## Get predictions on training and test data
preds_train = model_predict(train_data,final_weights)
preds_test = model_predict(test_data,final_weights)

## Compute errors
errs_train = np.sum((preds_train > 0.0) != (train_labels > 0.0))
errs_test = np.sum((preds_test > 0.0) != (test_labels > 0.0))

print ("Training error: ", float(errs_train)/len(train_labels))
print ("Test error: ", float(errs_test)/len(test_labels))
```

```
Training error:  0.0004
Test error:  0.002

<ipython-input-15-92528e4315e1>:8: DeprecationWarning: elementwise comparison
failed; this will raise an error in the future.
  errs_train = np.sum((preds_train > 0.0) != (train_labels > 0.0))
<ipython-input-15-92528e4315e1>:9: DeprecationWarning: elementwise comparison
failed; this will raise an error in the future.
  errs_test = np.sum((preds_test > 0.0) != (test_labels > 0.0))
```

3. Analyzing the margin

As discussed in the lecture, the logistic regression model produces not just classifications but also conditional probability estimates.

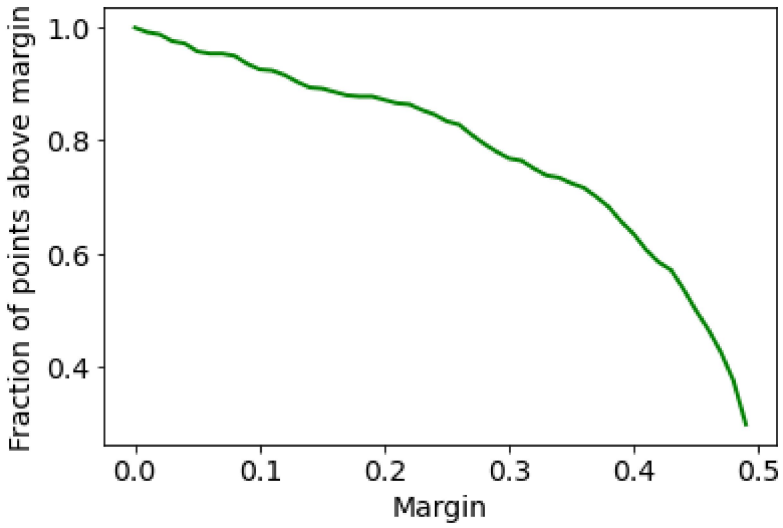
We will say that `x` has **margin** `gamma` if (according to the logistic regression model) `Pr(y=1|x) > (1/2)+gamma` or `Pr(y=1|x) < (1/2)-gamma`. For example, if `Pr(y=1|x)` is 0.7 according to the logistic regression model, then the margin is 0.2. If `Pr(y=1|x)` is 0.15 according to the logistic regression model, then the margin is 0.35.

Task P6: Implement the following function **margin_counts** that takes as input the learned weights $\tilde{\theta}$, the feature matrix (`feature_matrix`), and a value of `gamma`, and computes how many points in the data have margin at least `gamma`. Copy the code and the output plot (i.e., visualization of the test set's distribution of margin values) to the solution file.

In [16]:
```
def margin_counts(feature_matrix, weights, gamma):
## Return number of points for which Pr(y=1) lies in [0, 0.5 - gamma) or (0.5
 + gamma, 1]

    # Input:
    # feature_matrix: numpy array of size n by d+1, where n is the number of d
ata points, and d+1 is the feature dimension
    #                 note we have included the dummy feature as the first col
umn of the feature_matrix
    # weights: weight vector to start with, a numpy vector of dimension d+1
    # gamma: the margin value
    # Output:
    # number of points for which Pr(y=1) lies in [0, 0.5 - gamma) or (0.5 + ga
mma, 1]

    ## STUDENT: YOUR CODE HERE
    gamma_sum = 0
    for i in range(len(feature_matrix)):
        inside = np.dot(weights, feature_matrix[i])
        predict_pos = 1 / (1 + np.exp(-1. * inside))
        if (0 <= predict_pos < (0.5 - gamma)) or ((0.5 + gamma) < predict_pos
<= 1):
            gamma_sum += 1

    return gamma_sum

    ## STUDENT: CODE ENDS
```

We now visualize the test set's distribution of margin values.

In [17]:
```
gammas = np.arange(0,0.5,0.01)
f = np.vectorize(lambda g: margin_counts(test_data, final_weights,g))
plt.plot(gammas, f(gammas)/500.0, linewidth=2, color='green')
plt.xlabel('Margin', fontsize=14)
plt.ylabel('Fraction of points above margin', fontsize=14)
plt.show()
```



Next, we investigate a natural question: Are points `x` with larger margin more likely to be classified correctly?

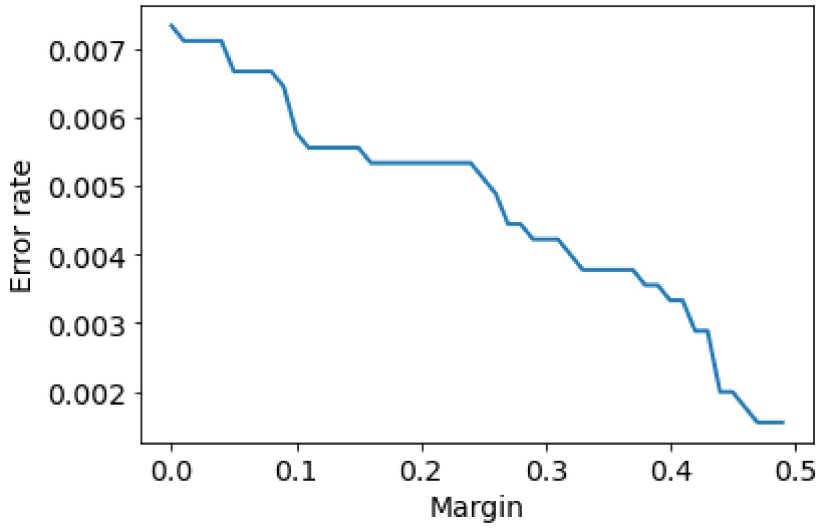
To address this, we define a function **margin_errors** that computes the fraction of points with margin at least `gamma` that are misclassified.

Task P7: Implement the function `margin_errors` that computes the fraction of points with margin at least `gamma` that are misclassified. Copy the code and the output plot (i.e., visualization of the relationship between margin and error rate) to the solution file. What do you observe from the plot?

In [18]:
```
def margin_errors(feature_matrix, labels, weights, gamma):
## Return error of predictions that lie in intervals [0, 0.5 - gamma) and (0.5
+ gamma, 1]

    # Input:
    # feature_matrix: numpy array of size n by d+1, where n is the number of d
ata points, and d+1 is the feature dimension
    #                 note we have included the dummy feature as the first col
umn of the feature_matrix
    # labels: true labels y, a numpy vector of dimension n
    # weights: weight vector to start with, a numpy vector of dimension d+1
    # gamma: the margin value
    # Output:
    # error of predictions that lie in intervals [0, 0.5 - gamma) and (0.5 + g
amma, 1]

    ## STUDENT: YOUR CODE HERE
    gamma_error = 0.
    predictions = model_predict(feature_matrix, weights)
    for i in range(len(feature_matrix)):
        inside = np.dot(weights, feature_matrix[i])
        predict_pos = 1 / (1 + np.exp(-1. * inside))
        if (0 <= predict_pos < (0.5 - gamma)) or ((0.5 + gamma) < predict_pos
<= 1):
            if predictions[i] == labels[i]:
                gamma_error += 1.

    return gamma_error / len(predictions)

    ## STUDENT: YOUR CODE ENDS
```

We now visualize the relationship between margin and error rate.

In [19]:
```
## Create grid of gamma values
gammas = np.arange(0, 0.5, 0.01)

## Compute margin_errors on test data for each value of g
f = np.vectorize(lambda g: margin_errors(test_data, test_labels,final_weights,
g))

## Plot the result
plt.plot(gammas, f(gammas), linewidth=2)
plt.ylabel('Error rate', fontsize=14)
plt.xlabel('Margin', fontsize=14)
plt.show()
```



4. Words with large influence

Finally, we attempt to partially **interpret** the logistic regression model.

Which words are most important in deciding whether a sentence is positive? As a first approximation to this, we simply take the words whose coefficients in θ have the largest positive values.

Likewise, we look at the words whose coefficients in θ have the most negative values, and we think of these as influential in negative predictions.

Task P8: Report the top 10 positive words (i.e., words with the largest positive coefficients of θ) and the top 10 negative words (i.e., words with the most negative coefficients of θ).

In [26]:
```
## Convert vocabulary into a list:
## This is a list where the i-th entry corresponds to the

vocab = np.array([z[0] for z in sorted(vectorizer.vocabulary_.items(), key=lam
bda x:x[1])])

## STUDENT: YOUR CODE HERE
sorted_words = np.argsort(final_weights)
pos_words = []
neg_words = []
for i in range(10):
    pos_words.append(vocab[sorted_words[i]])
    neg_words.append(vocab[sorted_words[-i]])

print("Top ten influential words:")
print("Positive Words")
for i in range(10):
    print(i + 1, ": ", pos_words[i])
print("Negative Words")
for i in range(10):
    print(i + 1, ": ", neg_words[i])

## STUDENT: CODE ENDS
```

```
Top ten influential words:
Positive Words
1 :  greater
2 :  loved
3 :  perfected
4 :  awful
5 :  delight
6 :  nicely
7 :  lovely
8 :  function
9 :  likes
10 :  beautifully
Negative Words
1 :  greater
2 :  disapppointment
3 :  poorly
4 :  wasted
5 :  worth
6 :  disappointment
7 :  arepas
8 :  sucks
9 :  badly
10 :  dog
```

5. (Bonus question) Classifiers that can abstain

Suppose you are building a classifier, and can tolerate an error rate of at most some value e. Unfortunately, every classifier you try has a higher error than this.

Therefore, you decide that the classifier is allowed to occasionally **abstain**: that is, to say *"don't know"*. When it actually makes a prediction, it must have error rate at most e. And subject to this constraint, it should abstain as infrequently as possible.

How would you build an abstaining classifier of this kind, starting from a logistic regression model? To get the bonus score, you need to show the following:

- A general description of the method
- Your code implementation
- A case study to show how you can use it in practice (including necessary plots)

In [21]:
```
## STUDENT: YOUR CODE HERE


## STUDENT: CODE ENDS
```