CS189

1. (a) By myself. consulting piazza.

(b) I certify that all solutions are entirely in my words and that I have not looked at another student's solutions. I have credited all external sources in this write up.

Yao Cao

HW08 Yao Cui 25933599

2. (a) $\|w^{t+1} - w^*\|_2^2 = \|w^t - \alpha_t G_t - w^*\|_2^2$

$= \|w^t - w^* - \alpha_t G_t\|_2^2$

$= (w^t - w^* - \alpha_t G_t)^T (w^t - w^* - \alpha_t G_t)$

$= (w^t - w^*)^T (w^t - w^*) + \alpha_t^2 G_t^T G_t - 2\alpha_t G_t^T (w^t - w^*)$

$= \|w^t - w^*\|_2^2 - 2\alpha_t \langle G_t, w^t - w^* \rangle + \alpha_t^2 \|G_t\|_2^2$

(b) $E[\langle G(w^t), w^t - w^* \rangle] = E_{i_1, \cdots i_{t-1}}[E_{i_t}[\langle G(w^t), w^t - w^* \rangle | i_1, \cdots, i_{t-1}]]$

$= E_{i_1, \cdots i_{t-1}}[\langle E_{i_t}[\nabla f_{i_t}(w^t) | i_1 \cdots i_{t-1}], w^t - w^* \rangle]$

$= E_{i_1, \cdots i_{t-1}}[\langle \nabla f(w^t), w^t - w^* \rangle]$

$= E[\langle \nabla f(w^t), w^t - w^* \rangle]$

- $w^t - w^*$ depends on $i_1, \cdots i_{t-1}$
- $G(w^t)$ only depends on i_t for fixed w^t
- unbiasedness of SG

$\Delta_{t+1} = E\|w^{t+1} - w^*\|_2^2 = E\|w^t - w^*\|_2^2 - 2\alpha_t E[\langle G_t, w^t - w^* \rangle] + \alpha_t^2 E\|G_t\|_2^2$

$\leq \Delta_t + \alpha_t^2 (M_g^2 \|w^t - w^*\|_2^2 + B^2) - 2\alpha_t E[\langle \nabla f(w^t), w^t - w^* \rangle]$

$= (1 + \alpha_t^2 M_g^2) \Delta_t + \alpha_t^2 B^2 - 2\alpha_t E[\langle \nabla f(w^t), w^t - w^* \rangle]$

(c) for OLS, $\nabla f(w^*) = 0$

$$\nabla f(w) = \nabla\left[\frac{1}{2}\|Xw-y\|_2^2\right]$$

$$= \nabla\left[\frac{1}{2}(Xw-y)^T(Xw-y)\right]$$

$$= \nabla\left[\frac{1}{2}(w^TX^TXw + y^Ty - 2w^TX^Ty\right]$$

$$= \frac{1}{2}(2X^TXw - 2X^Ty)$$

$$= X^TXw - X^Ty$$

$$\begin{aligned}\langle \nabla f(w^t), w^t-w^*\rangle &= \langle \nabla f(w^t) - \nabla f(w^*), w^t-w^*\rangle \\ &= \langle X^TXw^t - X^Ty - X^TXw^* + X^Ty, w^t-w^*\rangle \\ &= \langle X^TX(w^t-w^*), w^t-w^*\rangle \\ &= (w^t-w^*)^TX^TX(w^t-w^*) \\ &\geq \lambda_{min}(X^TX)\|w^t-w^*\|_2^2 \\ &= m\|w^t-w^*\|_2^2\end{aligned}$$

$$\Delta_{t+1} \leq (1+\alpha_t^2M_g^2)\Delta_t - 2\alpha_t E\langle \nabla f(w^t), w^t-w^*\rangle + \alpha_t^2B^2$$

$$\leq (1+\alpha_t^2M_g^2)\Delta_t - 2\alpha_t m\Delta_t + \alpha_t^2B^2$$

$$= (1+\alpha_t^2M_g^2 - 2\alpha_t m)\Delta_t + \alpha_t^2B^2$$

(d) $G_t(w^t) = \nabla f_{it}(w^t)$

$= X_{it}^T X_{it} w^t - X_{it}^T y_{it}$

$= X_{it}^T X_{it} w^t - X_{it}^T X_{it} w^*$

$= X_{it}^T X_{it} (w^t - w^*)$

$\| G_t(w^t) \|_2^2 = \| X_{it}^T X_{it} (w^t - w^*) \|_2^2$

$\leq \lambda_{max}(X_{it}^T X_{it} X_{it}^T X_{it}) \| w^t - w^* \|_2^2$

$E \| G_t(w^t) \|_2^2 \leq E[\lambda_{max}(X_{it}^T X_{it} X_{it}^T X_{it})] \| w^t - w^* \|_2^2 \Rightarrow B = 0$

$\leq \max_i \| x_i \|_2^4 \| w^t - w^* \|_2^2$

$\Delta_t \leq (1 + \alpha^2 M_g^2 - 2\alpha m) \Delta_{t-1}$

$\leq (1 + \alpha^2 M_g^2 - 2\alpha m)^2 \Delta_{t-2}$

$\leq (1 + \alpha^2 M_g^2 - 2\alpha m)^t \Delta_0$

$\min_\alpha (1 + \alpha^2 M_g^2 - 2\alpha m) = \min_\alpha (\alpha^2 M_g^2 - 2\alpha m + \frac{m^2}{M_g^2} - \frac{m^2}{M_g^2} + 1)$

$= \min_\alpha \left[(\alpha M_g - \frac{m}{M_g})^2 + (1 - \frac{m^2}{M_g^2}) \right]$

$\alpha^* = \frac{m}{M_g^2}$

$\Delta_t \leq (1 - \frac{m^2}{M_g^2})^t \Delta_0$

(e) $\Delta_t \le r\Delta_{t-1} + \alpha^2 B^2$

$\le r(r\Delta_{t-2} + \alpha^2 B^2) + \alpha^2 B^2$

$= r^2\Delta_{t-2} + r\alpha^2 B^2 + \alpha^2 B^2$

$\le r^2(r\Delta_{t-3} + \alpha^2 B^2) + (r+1)\alpha^2 B^2$

$= r^3\Delta_{t-3} + (r^2 + r + 1)\alpha^2 B^2$

$\le r^t\Delta_0 + \alpha^2 B^2 \sum_{i=0}^{t-1} r^i$

$r = 1 + \alpha^2 M_g^2 - 2\alpha m$

$\le r^t\Delta_0 + \alpha^2 B^2 \left(\frac{1}{1-r}\right)$

$= r^t\Delta_0 + \alpha^2 B^2 \left(\frac{1}{-\alpha^2 M_g^2 + 2\alpha m}\right)$

$= r^t\Delta_0 + \frac{\alpha B^2}{2m - \alpha M_g^2}$

$t \to \infty$

$\Delta_t \le \frac{\alpha B^2}{2m - \alpha M_g^2}$

$r = 1 + \alpha^2 M_g^2 - 2\alpha m < 1$

$\Rightarrow \alpha(2m - \alpha M_g^2) > 0$

$\Rightarrow 2m - \alpha M_g^2 > 0$

$\Rightarrow \frac{\alpha B^2}{2m - \alpha M_g^2} > 0$

$\frac{\alpha B^2}{2m - \alpha M_g^2}$ is a finite number once α, B, M_g are known.

So there is no guarantee $w^t \to w^*$ when $t \to \infty$

In [66]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import random
```

Part A: one solution

Assuming that I want to find the w that minimizes $\frac{1}{2n}||Xw - y||_2^2$. In this part, X is full rank, and $y \in range(X)$

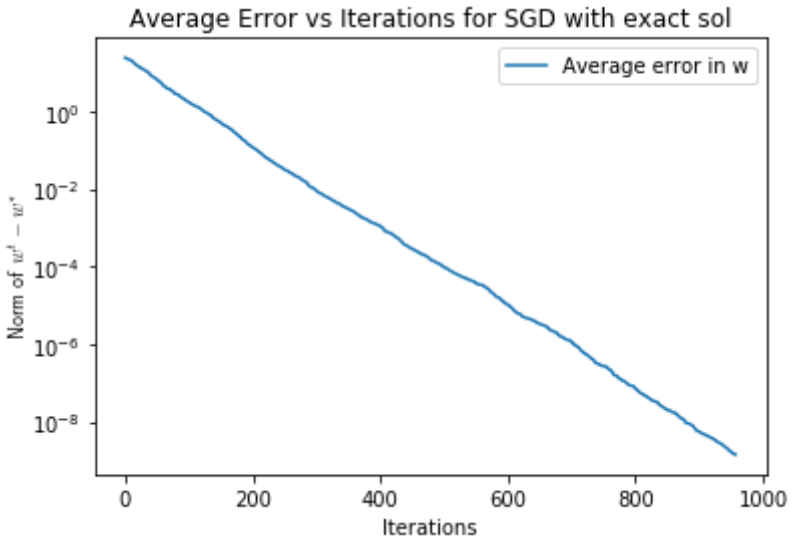
In [67]:
```
X = np.random.normal(scale = 20, size=(100,10))
print(np.linalg.matrix_rank(X)) # confirm that the matrix is full rank
# Theoretical optimal solution
w = np.random.normal(scale = 10, size = (10,1))
y = X.dot(w)
```

```
10
```

In [68]:
```
def sgd(X, y, w_actual, threshold, max_iterations, step_size, gd=False):
    if isinstance(step_size, float):
        step_size_func = lambda i: step_size
    else:
        step_size_func = step_size

    # run 10 gradient descent at the same time, for averaging purpose
    # w_guesses stands for the current iterates (for each run)
    w_guesses = [np.zeros((X.shape[1], 1)) for _ in range(10)]
    n = X.shape[0]
    error = []
    it = 0
    above_threshold = True
    previous_w = np.array(w_guesses)

    while it < max_iterations and above_threshold:
        it += 1
        curr_error = 0
        for j in range(len(w_guesses)):
            if gd:
                # Your code, implement the gradient for GD
                sample_gradient = X.T @ X @ w_guesses[j] - X.T @ y
            else:
                # Your code, implement the gradient for SGD
                index = random.sample(range(X.shape[0]), 1)[0]

                sample_gradient = X[index:index+1, :].T @ X[index:index+1, :
                            X[index:index+1, :].T @ y[index:index+1, :]

            # Your code: implement the gradient update
            # learning rate at this step is given by step_size_func(it)
            w_guesses[j] = w_guesses[j] - step_size_func(it) * sample_gradie

            curr_error += np.linalg.norm(w_guesses[j]-w_actual)
        error.append(curr_error/10)

        diff = np.array(previous_w) - np.array(w_guesses)
        diff = np.mean(np.linalg.norm(diff, axis=1))
        above_threshold = (diff > threshold)
        previous_w = np.array(w_guesses)
    return w_guesses, error
```

In [69]:
```
its = 5000
w_guesses, error = sgd(X, y, w, 1e-10, its, 0.0001)
```

In [70]:
```
iterations = [i for i in range(len(error))]
#plt.semilogy(iterations, error, label = "Average error in w")
plt.semilogy(iterations, error, label = "Average error in w")
plt.xlabel("Iterations")
plt.ylabel("Norm of $w^t - w^*$",  usetex=True)
plt.title("Average Error vs Iterations for SGD with exact sol")
plt.legend()
plt.show()
```



In [71]:
```
print("Required iterations: ", len(error))
average_error = np.mean([np.linalg.norm(w-w_guess) for w_guess in w_guesses]
print("Final average error: ", average_error)
print("When there is exact solution, with constant learning rate, SGD has li
```

```
Required iterations:  958
Final average error:  1.43534203733e-09
```

Part B: No solutions, constant step size

In [72]:
```
y2 = y + np.random.normal(scale=5, size = y.shape)
w=np.linalg.inv(X.T @ X) @ X.T @ y2
```

In [73]:
```
its = 5000
w_guesses2, error2 = sgd(X, y2, w, 1e-5, its, 0.0001)
w_guesses3, error3 = sgd(X, y2, w, 1e-5, its, 0.00001)
w_guesses4, error4 = sgd(X, y2, w, 1e-5, its, 0.000001)
```

In [74]:
```
w_guess_gd, error_gd = sgd(X, y2, w, 1e-5, its, 0.00001, True)
```

In [75]:
```
plt.semilogy([i for i in range(len(error2))], error2, label="SGD, lr = 0.000
plt.semilogy([i for i in range(len(error3))], error3, label="SGD, lr = 0.000
plt.semilogy([i for i in range(len(error4))], error4, label="SGD, lr = 0.000
plt.semilogy([i for i in range(len(error_gd))], error_gd, label="GD, lr = 0.
plt.xlabel("Iterations")
plt.ylabel("Norm of $w^t - w^*$",  usetex=True)
plt.title("Total Error vs Iterations for SGD without exact sol")
plt.legend()
plt.show()
```

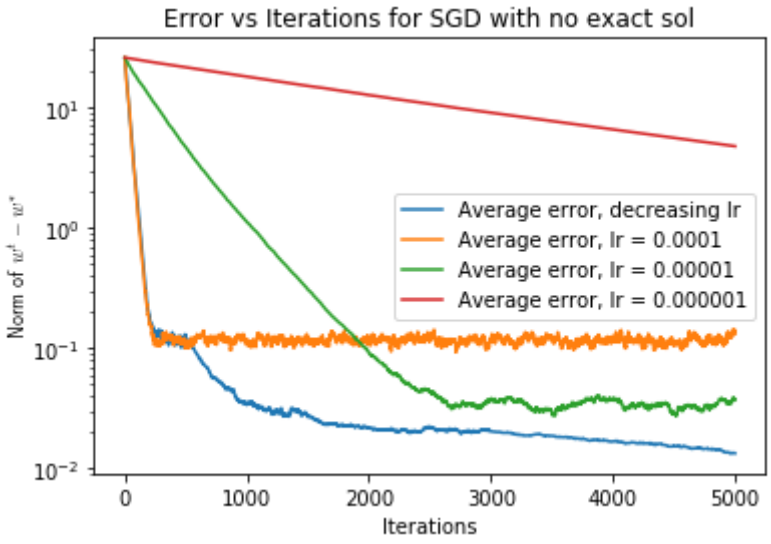
In [83]:
```
print("Required iterations, lr = 0.0001: ", len(error2))
average_error = np.mean([np.linalg.norm(w-w_guess) for w_guess in w_guesses2
print("Final average error: ", average_error)

print("Required iterations, lr = 0.00001: ", len(error3))
average_error = np.mean([np.linalg.norm(w-w_guess) for w_guess in w_guesses3
print("Final average error: ", average_error)

print("Required iterations, lr = 0.000001: ", len(error4))
average_error = np.mean([np.linalg.norm(w-w_guess) for w_guess in w_guesses4
print("Final average error: ", average_error)

print("Required iterations, GD: ", len(error_gd))
average_error = np.mean([np.linalg.norm(w-w_guess) for w_guess in w_guess_gd
print("Final average error: ", average_error)

print("when there is no exact solutions, with constant learning rate, there
SGD can get and the bound depends on the constant learning rate. The smaller
decreases but the bound also becomes smaller. GD can always converge with an
```

```
Required iterations, lr = 0.0001:  5000
Final average error:  0.133683684061
Required iterations, lr = 0.00001:  5000
Final average error:  0.037174266597
Required iterations, lr = 0.000001:  5000
Final average error:  4.77185536155
Required iterations, GD:  47
Final average error:  2.97441149944e-05
when there is no exact solutions, with constant learning rate, there is a
lways a bound on the smallest errorSGD can get and the bound depends on t
he constant learning rate. The smaller the learning rate, the slower the
errordecreases but the bound also becomes smaller. GD can always converge
with an appropriate constant step size.
```

Part C: No solutions, decreasing step size

In [77]:
```
its = 5000
def step_size(step):
    if step < 500:
        return 1e-4
    if step < 1500:
        return 1e-5
    if step < 3000:
        return 3e-6
    return 1e-6

w_guesses_variable, error_variable = sgd(X, y2, w, 1e-10, its, step_size, Fa
```

In [78]:
```
plt.semilogy([i for i in range(len(error_variable))], error_variable, label=
plt.semilogy([i for i in range(len(error2))], error2, label="Average error,
plt.semilogy([i for i in range(len(error3))], error3, label="Average error,
plt.semilogy([i for i in range(len(error4))], error4, label="Average error,

plt.xlabel("Iterations")
plt.ylabel("Norm of $w^t - w^*$",  usetex=True)
plt.title("Error vs Iterations for SGD with no exact sol")
plt.legend()
plt.show()
```



In [84]:
```
print("Required iterations, variable lr: ", len(error_variable))
average_error = np.mean([np.linalg.norm(w-w_guess) for w_guess in w_guesses_
print("Average error with decreasing lr:", average_error)

print("when there is no exact solutions, with a proper decaying learning rat
and small error bound at the same time")
```

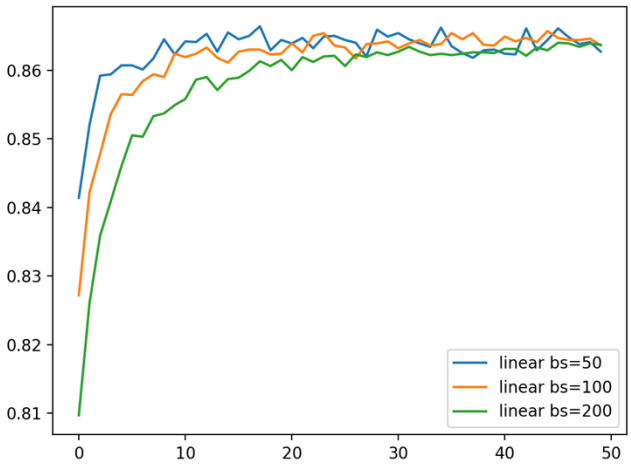
```
Required iterations, variable lr:  5000
Average error with decreasing lr: 0.0134313632012
when there is no exact solutions, with a proper decaying learning rate, w
e can achieve high learning efficiencyand small error bound at the same t
ime
```

In [ ]:
```

```

3a

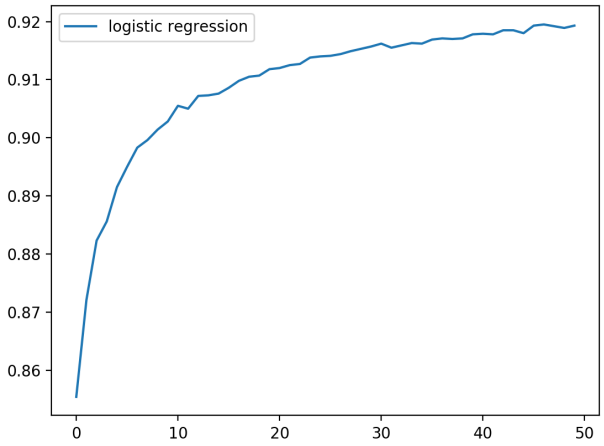
batch size: 50, 100, 200:
train_linear finishes in 44.193s
train_linear finishes in 32.491s
train_linear finishes in 24.578s

Using small batch size converges in fewer epochs but each epoch takes more time.



validation accuracy v.s. epoch

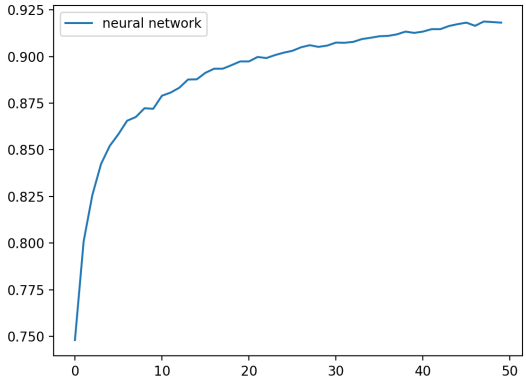
3b



validation accuracy v.s. epoch

```
def train_logistic(learning_rate=0.01, training_epochs=50, batch_size=100):
        x = tf.placeholder(tf.float32, [None, 784])
        y = tf.placeholder(tf.float32, [None, 10])

        W = tf.Variable(tf.zeros([784, 10]))
        b = tf.Variable(tf.zeros([10]))

        # YOUR CODE HERE
        z = tf.matmul(x, W) + b
        z_max = tf.reduce_max(z, axis = 1, keepdims=True)
        z_exp = tf.exp(tf.subtract(z, z_max))
        pred = tf.divide(z_exp, tf.reduce_sum(z_exp, axis = 1, keepdims=True))

        loss = -tf.reduce_mean(tf.log(tf.reduce_sum(tf.multiply(pred, y), axis = 1)))
        ##################

        optimizer =
tf.train.GradientDescentOptimizer(learning_rate).minimize(loss)
        return optimize(x, y, pred, loss, optimizer, training_epochs, batch_size)
```

3c



validation accuracy v.s. epoch

```
def train_nn(learning_rate=0.01, training_epochs=50, batch_size=100,
n_hidden=64):
        x = tf.placeholder(tf.float32, [None, 784])
        y = tf.placeholder(tf.float32, [None, 10])

        W1 = tf.Variable(tf.random_normal([784, n_hidden]))
        W2 = tf.Variable(tf.random_normal([n_hidden, 10]))
        b1 = tf.Variable(tf.random_normal([n_hidden]))
        b2 = tf.Variable(tf.random_normal([10]))

        # YOUR CODE HEREs
        z = tf.matmul(tf.tanh(tf.matmul(x, W1) + b1), W2) + b2
        z_max = tf.reduce_max(z, axis = 1, keepdims=True)
        z_exp = tf.exp(tf.subtract(z, z_max))
        pred = tf.divide(z_exp, tf.reduce_sum(z_exp, axis = 1, keepdims=True))

        loss = -tf.reduce_mean(tf.log(tf.reduce_sum(tf.multiply(pred, y), axis = 1)))
        ##################

        optimizer =
tf.train.GradientDescentOptimizer(learning_rate).minimize(loss)
        return optimize(x, y, pred, loss, optimizer, training_epochs, batch_size)
```

3. (d) $y_i - z_y = w^T(x_i - z_x)$

$y_i = w^T x_i - w^T z_x + z_y$

$Y_i \mid x_i \sim N(w^T x_i, 1 + \|w\|_2^2)$

$$\prod_{i=1}^{n} P(Y_i \mid x_i) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi(1+\|w\|_2^2)}} \exp\left(-\frac{(y_i - w^T x_i)^2}{2(1+\|w\|_2^2)}\right)$$

$$\log \prod_{i=1}^{n} P(Y_i \mid x_i) = C - \frac{n}{2}\log(1+\|w\|_2^2) - \sum_{i=1}^{n} \frac{(y_i - w^T x_i)^2}{2(1+\|w\|_2^2)}$$

3f

The solution is sensitive to the hyperparameters, as below:

epochs = 300

Adam:

	Learning rate = 0.1	Learning rate = 0.01	Learning rate = 0.001
Batch size = 100	0.443025371	0.083657307	**0.062726066**
Batch size = 500	0.099020762	0.095370744	4.739486885
Batch size = 1000	0.067676768	0.070304008	47.509384821

SGD:

	Learning rate = 0.1	Learning rate = 0.01	Learning rate = 0.001
Batch size = 100	0.085627795	**0.057028181**	0.066219059
Batch size = 500	0.070977511	0.068275265	20.465322206
Batch size = 1000	0.087335413	0.061001299	84.321256260

```
# YOUR CODE HERE
   y_diff = y-tf.matmul(x, w)
   w_term = 1+tf.norm(w)**2
   cost = 1.0/2 * tf.log(w_term) + 1.0/2/(w_term) * tf.reduce_mean(y_diff**2)
   ################
```

In [18]:
```
import numpy as np
import scipy as sp
import scipy.stats as st
import scipy.interpolate
import scipy.linalg as la
import pylab as pl
import math
from scipy import stats
from scipy.stats import multivariate_normal, probplot
from sklearn import preprocessing
import statsmodels.api as sm
import pickle

import scipy as SP
import scipy.optimize as opt
import pylab
import matplotlib.pyplot as plt
plt.style.use("fivethirtyeight")

# import helper functions from starter code
from gwas import *

%matplotlib inline
```

GWAS problem - Introduction

In [ ]:
```

```

Overall goal: This real world problem is one in computational biology which uses many of the techniques and concepts you have been introduced to, all together, in particular, linear regression, PCA, non-iid noise, diagonalizing multivariate Gaussian covariance matrices, and bias-variance trade-off. We will also tangentially introduce you to concepts of statistical testing. \textbf{This homework problem is effectively a demo in that we will ask you to execute code and answer questions about what you observe. You are not required to code anything at all.}

Setup and problem statement: Given a set of people for whom genetics (DNA) has been measured, and also a corresponding trait for each person, such as blood pressure, or ”is-a-smoker”, one can use data-driven methods to deduce which genetic effects are likely responsible for the trait. We have collected blood from n individuals who either smoke ($y_i = 1$) or do not smoke ($y_i = 0$). Their blood samples have been sequenced at m positions along the genome, yielding the feature matrix $X \in \mathbb{R}^{n \times m}$, composed of the genetic variants (which take on values 0, 1, or 2). Specifically, $X_{i,j}$ is a numeric encoding of the DNA for the ith person at genetic feature j. We want to deduce which of the m genetic features are associated with smoking, by considering one at a time.

In the data we give you, it will turn out that there is no true signal in the data; however, we will see that without careful modeling, we would erroneously conclude that the data was full of signal. Left unmodeled, such structure creates misleading results, yielding signal where none exists.

Overall modelling approach: The basic idea will be to "test" one genetic feature at a time and assign a score (a p-value) indicating our level of belief that it is associated with the trait. We will start with a simple linear regression model, and then build increasingly more correct linear predictive models. The three models we will use are (1) linear regression, (2) linear regression with PCA features, (3) linear regression with all genetic variants as features. The first model is naive, because it assumes the individuals are iid. The fundamental modelling challenges with these kinds of analyses is that the individuals in the study are not typically not iid, owing to differences in race and family-relatedness (e.g., sisters, brothers, patients, grandparents in the data set), which violate the iid assumption. Left unmodelled, such structure creates misleading results, yielding signal where none exists. Luckily, as we shall see, one can visualize these modelling issues via quantile-quantile plots, which will soon be briefly introduced.

Quick introduction to Hypothesis Testing

To understand whether genetic marker j is informative in predicting the the target, we will set up a hypothesis test. A Hypothesis Test allows us to examine two different "views" of the world, the Null and the Alternative, and conclude which one is more likely based on the data we observe. A Hypothesis Test has the following main components:

- **Null Hypothesis:** The genetic marker is not significant in predicting the target. Any relation you observe between the genetic marker and the target is merely due to chance.
- **Alternative Hypothesis:** The genetic marker contains important information about the target.
- **p-value:** The final result of a hypothesis test is a p-value. A p-value is a number between $[0, 1]$ which you should interpret as follows: If the p-value is really small, that is evidence against the Null and in favor of the Alternative hypothesis. If the p-value is large then this is evidence towards the Null.

We will perform a single Hypothesis Test for each of the genetic markers that we have in our dataset. This process, will give us back m p-values. These m values will make up the empirical distribution of our p-values.

How to test each variant Herein we provide a minimal exposition to allow you to do the homework. To estimate how implicated each genetic feature is we will use a score, in the form of a p-value. One can think of the p-value as a proxy for how informative the genetic feature is for prediction of the trait (e.g. "is smoker"). More precisely, to get the p-value for the j^{th} genetic feature, we first include it in the model (for a given model class, such as linear regression) and compute the maximum likelihood, LL_j (this is our alternative hypothesis). Then we repeat this process, but having removed the genetic feature of interest from the model, yielding LL_{-j} (this is our null hypothesis). To be clear, the null hypothesis will have none of the m genetic variants that are being tested. You can refer to the jupyter notebook for a brief explanation of hypothesis testing. The p-value is then a simple monotonic decreasing function of the difference in these two likelihoods, `diff-ll` $= LL_j - LL_{-j}$-- one that we will give you. P-values lie in $[0, 1]$ and the smaller the p-value, the larger the difference in the likelihoods, and the more evidence that the genetic marker is associated with the trait (assuming the model is correct).

Part a

To diagnose if something is amiss in our genetic analyses, we will make use of the following: (1) we assume that if any genetic signal is present, it is restricted to a small fraction of the genetic features, (2) p-values corresponding to no signal in the data are distributed as $p \sim \text{Unif}[0, 1]$. Combining these two assumptions, we will assume that p-values arising from a valid model should largely be drawn from $\text{Unif}[0, 1]$, and also that large deviations suggest that our model is incorrect. Quantile-quantile plots let us visualize if we have deviations or not. Quantile-quantile plots are a way to compare two probability distributions by comparing their quantile values (e.g. how does the smallest p-value in each distribution compare, and then the second smallest, etc.). In the quantile-quantile plot, you will see m points, one for each genetic marker. The x-coordinate of each point corresponds to the theoretical quantile we would expect to see if the distribution was in fact a $Unif[0, 1]$ and the y-coordinate corresponds to the observed value of the quantile. An example is shown in Figure 1, where the line on the diagonal results from an analysis where the model is correct, and hence the theoretical and empirical p-value quantiles match, while the other line, which deviates from the diagonal, indicates that we have likely made a modelling error. If there are genetic signals in the data, these would simply emerge as a handful of outlier points from the diagonal (not shown).

Before we dive into developing our models, we need to be able to understand whether the p-values we get back from our hypothesis tests look like m random draws from a $Unif[0, 1]$.

Use the `qqplot` function to make a qq-plot for each of the 3 distributions provided below and explain your findings. What should we observe in our qq-plot if our empirical distribution looks more and more similar to a Unif[0, 1]? Note that we use two kinds of qq-plots: one in p-value space and one negative log p-value space. The former is for intuition, while the latter is for higher resolution. The green lines in the negative log p-value qq-plots indicate the error bars.

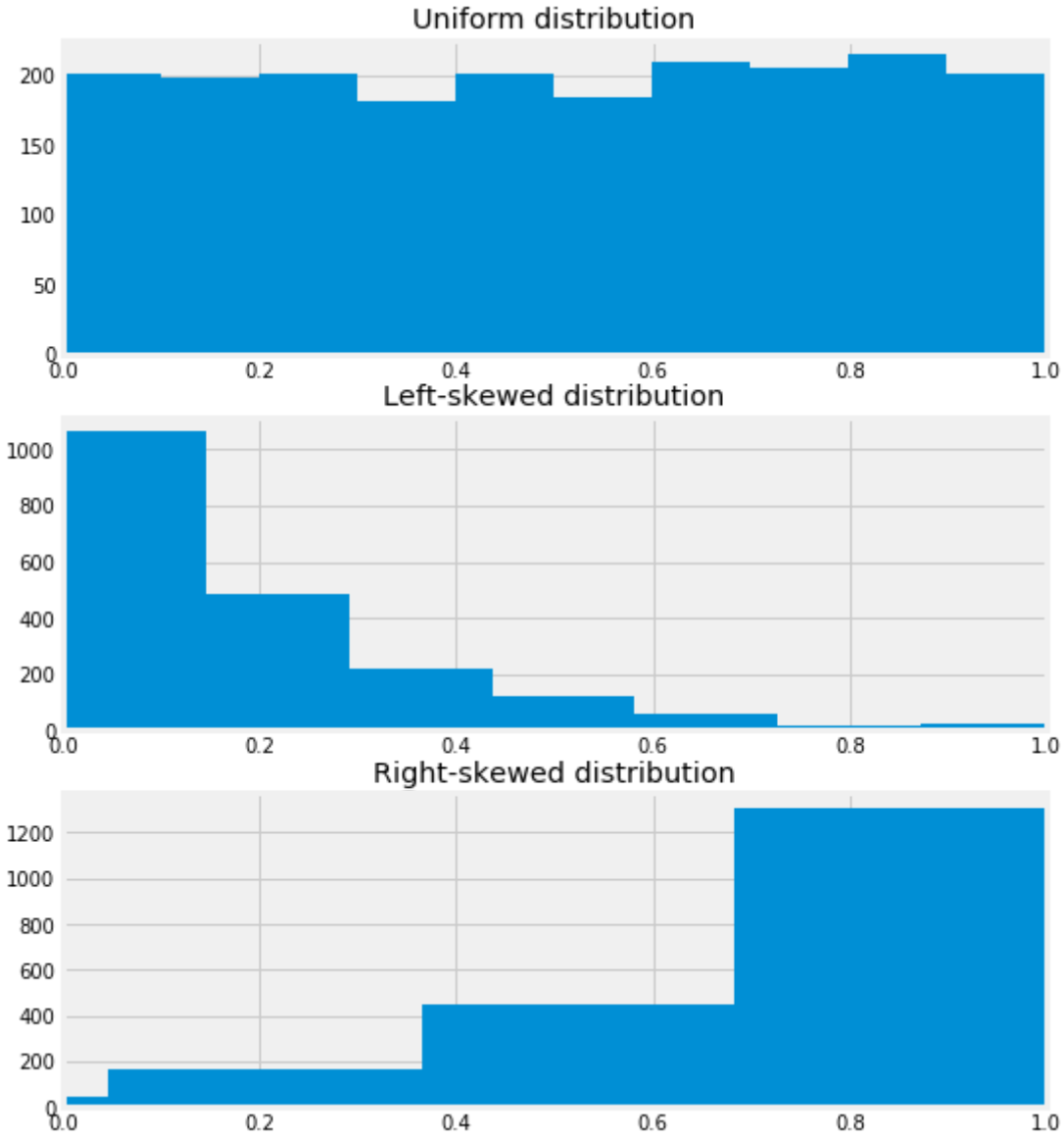
Answer to Part a:

If the emperical distribution is close to uniform distribution, the qq-line will line very close to x=y, or in the log scale plot, should fall inside the gray area. We can see that out of the three distributions considered here, only the one that is drawn from uniform meet the above requirement. The left skewed and right skewed distributions are far from uniform, clearly visible from the qq-plot.

In [2]:
```
# generate 3 distributions
n_points =  2000
unif = np.random.uniform(0, 1, n_points)
skewed_left = np.random.exponential(scale=0.2, size=n_points)
skewed_right = 1-np.random.exponential(scale=0.3, size=n_points)
```

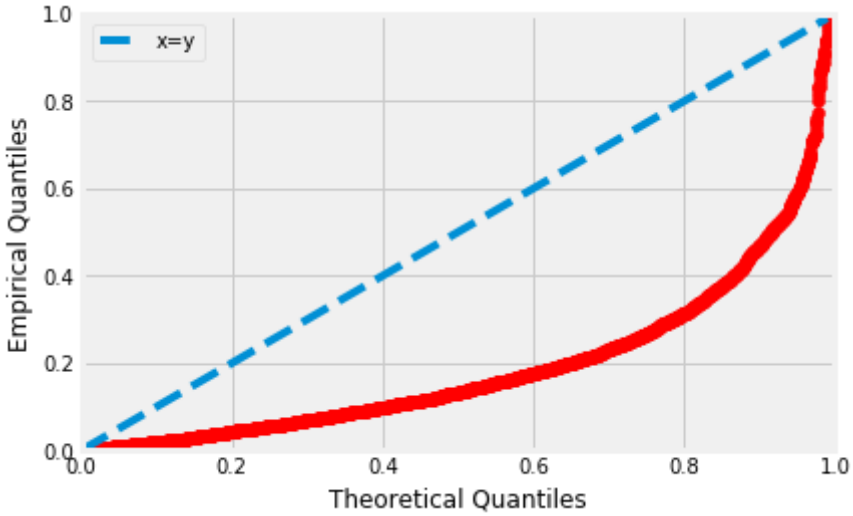
In [3]:
```
fig, axes = plt.subplots(nrows=3, ncols=1)
fig.set_figheight(10)
fig.set_figwidth(8)

# first subplot
axes[0].hist(unif)
axes[0].set_title('Uniform distribution')
axes[0].set_xlim(0, 1)

# second subplot
axes[1].hist(skewed_left)
axes[1].set_title('Left-skewed distribution')
axes[1].set_xlim(0, 1)

# third subplot
axes[2].hist(skewed_right)
axes[2].set_title('Right-skewed distribution')
axes[2].set_xlim(0, 1)
```

Out[3]: (0, 1)

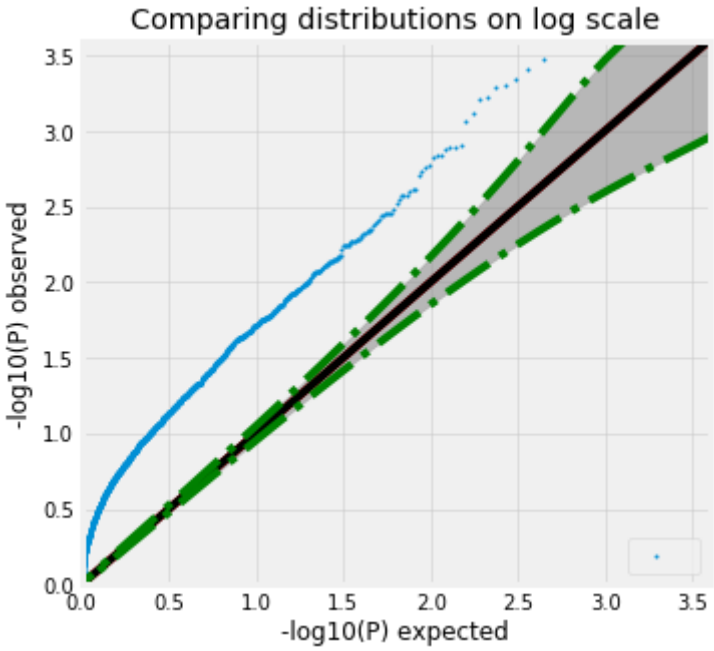


In [51]:
```
def qqplot(empirical_dist, legend = ""):
    """
    Generates two qq-plots one in the original scale of the data and
    one using a negative log transformation to make the difference
    between the expected and actual behavior more visible.

    If the observed line is outside the grey region denoted by the error bar
    then it is quite unlikely that our data came from a Unif[0, 1] distribut
    """
    x, y = probplot(empirical_dist, dist=stats.distributions.uniform(), fit=
    plt.scatter(x, y, c='r', alpha=1, )
    plt.xlabel("")
    plt.ylabel("")
    plt.xlim(0, 1)
    plt.ylim(0, 1)
    plt.xlabel("Theoretical Quantiles")
    plt.ylabel("Empirical Quantiles")
    plt.plot(np.arange(0, 1, 0.01), np.arange(0, 1, 0.01), "--", label='x=y'
    plt.legend()
    plt.show()
    fastlmm_qqplot(empirical_dist, legend=legend, xlim=(0,3.6), ylim=(0,3.6)
```

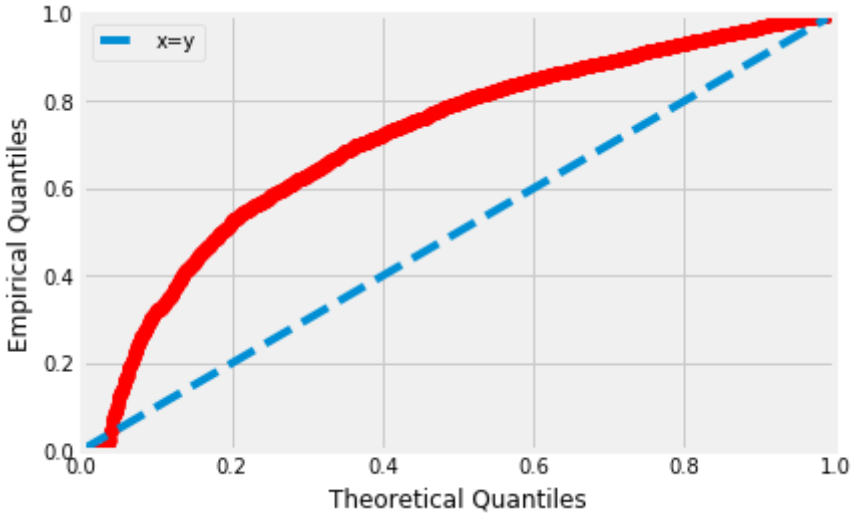
In [52]:
```
# SOLUTION CELL
qqplot(skewed_left)
```



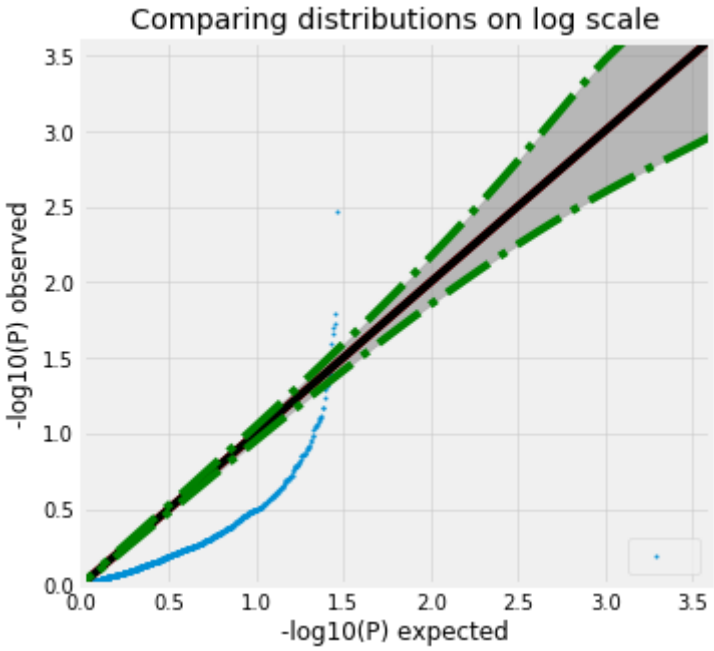

```
/Users/janeyu/anaconda/lib/python3.6/site-packages/numpy/lib/scimath.py:1
20: RuntimeWarning: invalid value encountered in less
  if any(isreal(x) & (x < 0)):
```



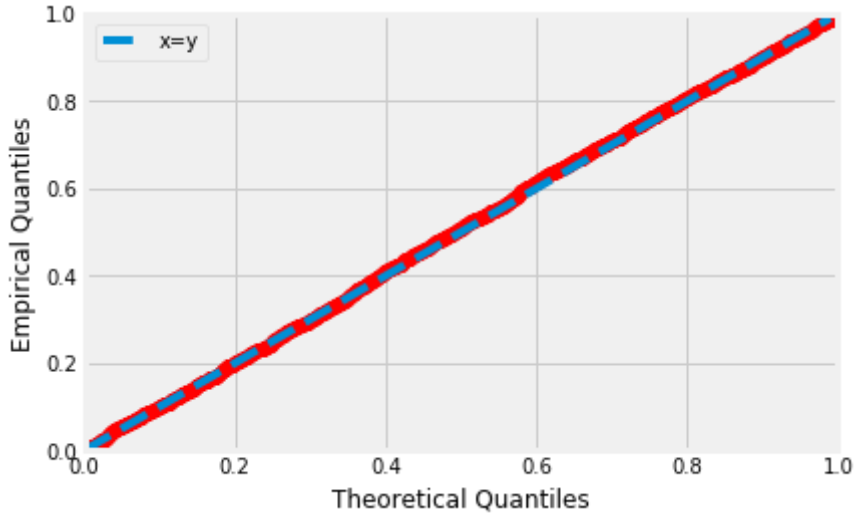
In [53]:
```
# SOLUTION CELL
qqplot(skewed_right)
```



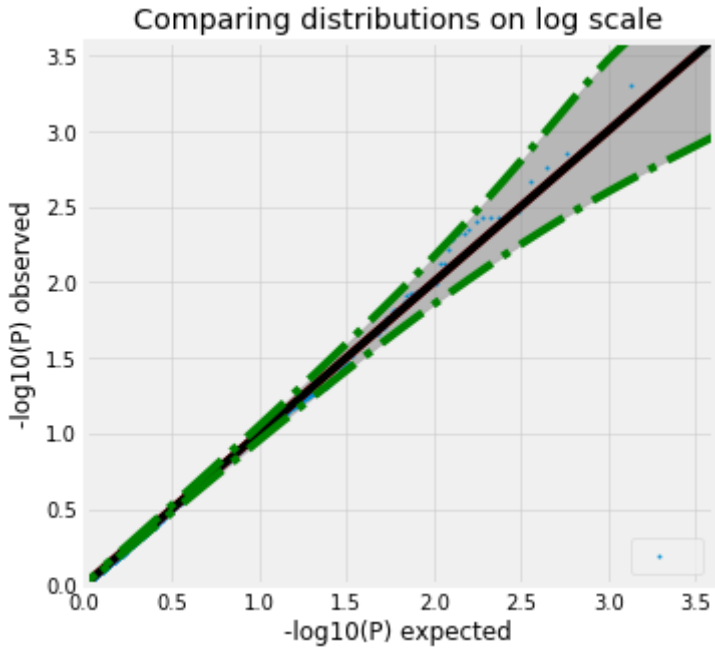

```
/Users/janeyu/anaconda/lib/python3.6/site-packages/numpy/lib/scimath.py:1
20: RuntimeWarning: invalid value encountered in less
  if any(isreal(x) & (x < 0)):
```



In [54]:
```
# SOLUTION CELL
qqplot(unif)
```




```
/Users/janeyu/anaconda/lib/python3.6/site-packages/numpy/lib/scimath.py:1
20: RuntimeWarning: invalid value encountered in less
  if any(isreal(x) & (x < 0)):
```



Part b

We will use linear models in the genetic marker we are testing. In particular, when testing the j^{th} genetic feature, we have that the trait, $\vec{y}$, is a linear function of the genetic variant, i.e. $\vec{y} = \overrightarrow{x_j} w_1 + \overrightarrow{w_0} + \vec{\epsilon}$where each entry ϵ_i of ϵ is random noise distributed as $N(0, \sigma^2)$, $w_1 \in \mathbb{R}^1$, $\overrightarrow{w_0} \in \mathbb{R}^{n \times 1}$ is a constant vector, and $\overrightarrow{x_j}$ is the jth column of X, representing data for the jth genetic variant. To simplify matters, we will add a column of ones to the right end of $\overrightarrow{x_j}$ and rewrite the regression as $\vec{y} = [\overrightarrow{x_j}, \vec{1}]\vec{w} + \vec{\epsilon}$where $[\overrightarrow{x_j}, \vec{1}]$ is the jth column of X with a vector of ones appended

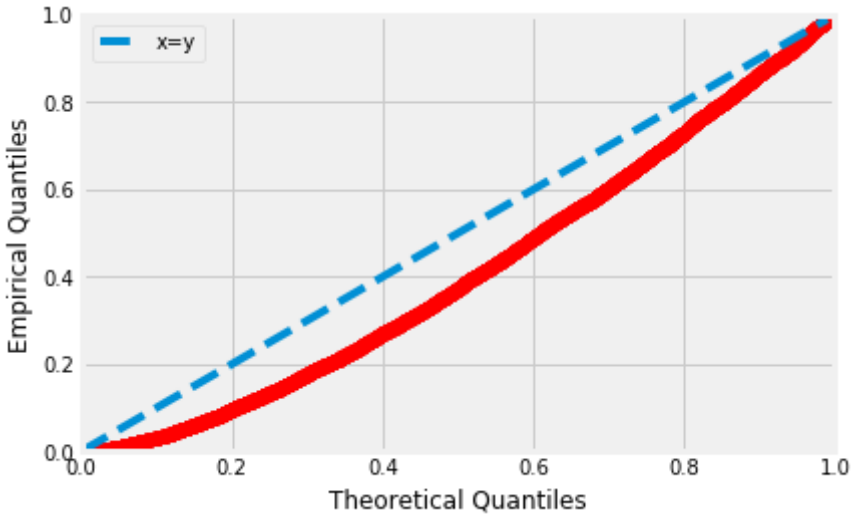
to the end and $\vec{w} \in \mathbb{R}^{2\times 1}$. The model without any genetic information, $\vec{y} = \vec{1}w + \vec{\epsilon}$ is referred to as the **null model** in the parlance of statistical testing. The **alternative model**, which includes the information we are testing (one genetic marker) is $\vec{y} = [\overrightarrow{x_j}, \vec{1}]\vec{w} + \vec{\epsilon}$ where $\overrightarrow{x_j}$ is the *j*th column of X, i.e. the data using only the *j*th genetic variant as a feature. **Plot the quantile-quantile plot of p-values using linear regression as just described, a so-called naive approach, by running the function** $naive\_model$. **From the plot, what do you conclude about the suitability of linear regression for this problem?**

Answer:

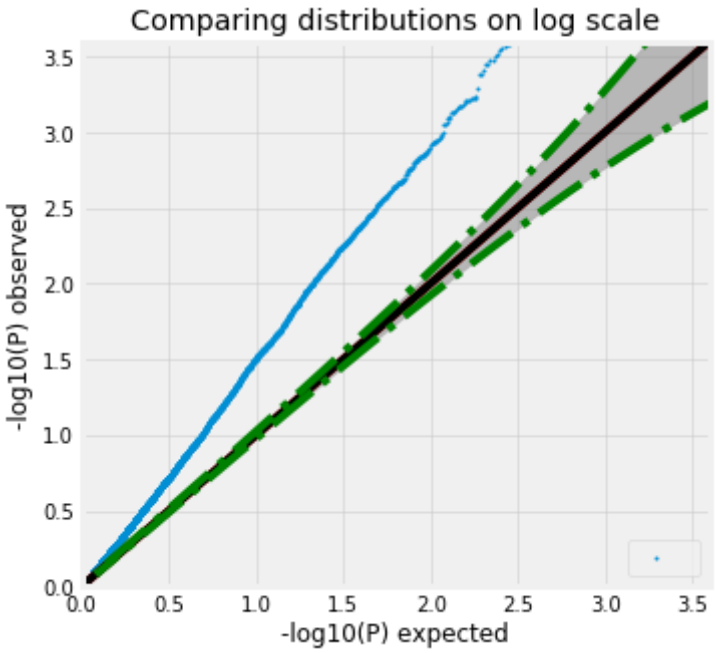
It seems using this model, the p-values for the genetic markers are not from an uniform distribution, which is against the assumption that most genetic markers have no effects on the trait y. So this model is not suitable for our problem.

In [8]:
```
# import all the data
X = load_X()
y=load_y()

#Normalize the columns
X = preprocessing.normalize(X, norm='l2', axis = 0)
y = y.reshape(-1, 1)
```

In [55]:
```
# SOLUTION CELL
# Note that the function naive_model returns the empirical distribution of p
# Green lines indicate error bars.
naive_pvals = naive_model(X,y)
plt.show()
qqplot(naive_pvals)
```



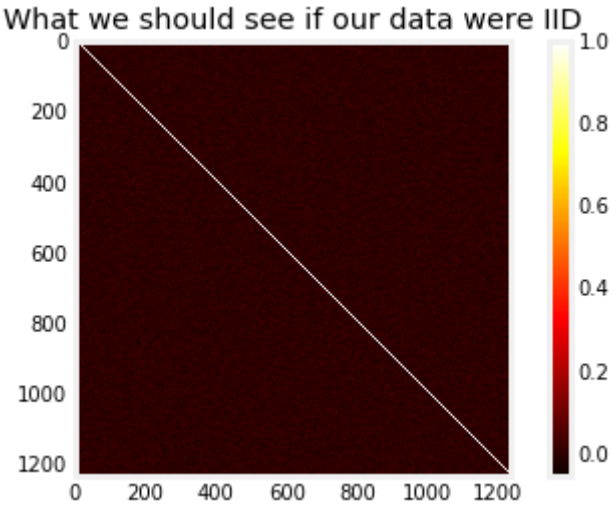

```
/Users/janeyu/anaconda/lib/python3.6/site-packages/numpy/lib/scimath.py:120: RuntimeWarning: invalid value encountered in less
  if any(isreal(x) & (x < 0)):
```



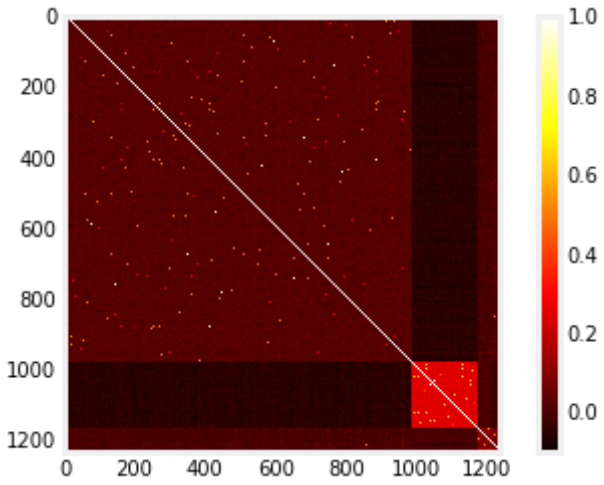
Based on the previous plots, we see that the model tells us that there are more genetic markers that are informative in predicting trait y than we would expect. Let's examine to what extent this is merely an artifact of the noise in our dataset not being iid using a correlation matrix. A correlation matrix visualizes the correlations between all datapoints in our dataset. If we can observe large correlations in the off-diagonal entries, then this presents strong evidence that our noise might not be iid.

In [10]:
```
# If our data were iid we would expect to see a correlation matrix to look s
X_r = np.random.randn(X.shape[0], X.shape[1])

corr_matrix = np.corrcoef(X_r)
plt.grid(False)
plt.imshow(corr_matrix, "hot")
plt.colorbar()
plt.title("What we should see if our data were IID")
```

Out[10]: Text(0.5,1,'What we should see if our data were IID')



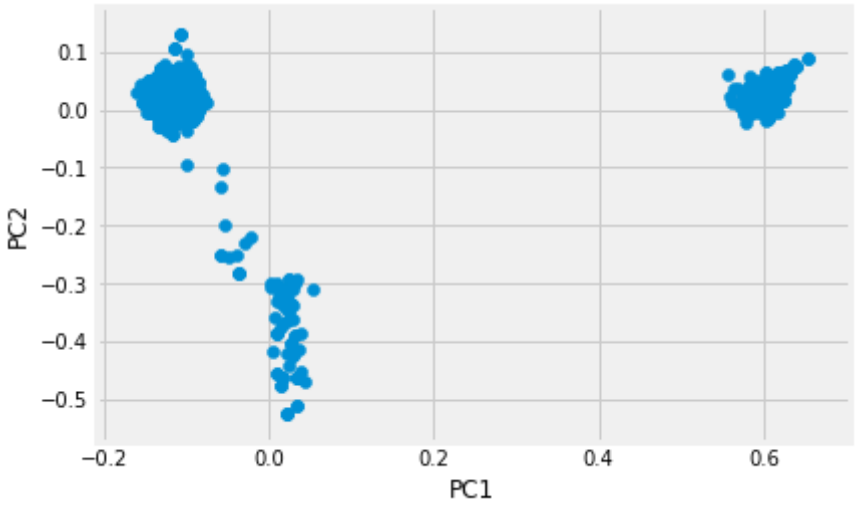
In [11]:
```
# Now let's see what we observe in our plot

from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import pandas as pd

def get_projection(X,k):
    _, _, V = np.linalg.svd(X)
    V = V.T
    return V[:, :k]

def corr_matrix_data(X):
    # demean the data
    X_demeaned = X - np.mean(X, 0)

    # get the projection matrix
    proj_matrix = get_projection(X_demeaned, 3)

    # get projected data
    X_proj = X_demeaned @ proj_matrix

    # do some k-means clustering to identify which points are in which clust
    km = KMeans(n_clusters = 3)
    clusters = km.fit_predict(X_proj)

    # sort data based on identified clusters
    t = pd.DataFrame(X)
    t['cluster'] = clusters
    t = t.sort_values("cluster")
    t = t.drop("cluster", 1)
    plt.imshow(np.corrcoef(t.as_matrix()), "hot")
    plt.colorbar()
    plt.grid(False)

corr_matrix_data(X)
```



By reorganizing the correlation matrix a bit, we can see that our data very much contains approximately 3 clusters, so in fact our iid noise assumptions certainly do not hold. At this point it might be a good idea to start considering projecting our data in a lower dimensional space, so that we can learn more about this dependency.

In [12]:
```
# Let's project our data in 2 dimensions
X_demeaned = X - np.mean(X, 0)
proj_matrix_2d = get_projection(X_demeaned, 2)
X_proj2d = X_demeaned @ proj_matrix_2d
plt.scatter(X_proj2d[:, 0], X_proj2d[:, 1])
plt.xlabel("PC1")
plt.ylabel("PC2")
```

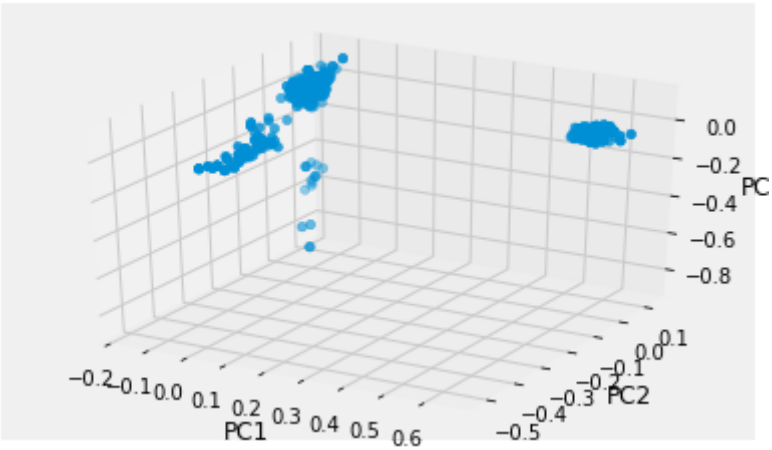
Out[12]:
```
Text(0,0.5,'PC2')
```



OR

In [13]:
```
# Now let's project on 3 dimensions
from mpl_toolkits.mplot3d import Axes3D

proj_matrix_3d = get_projection(X_demeaned, 3)
X_proj3d = X_demeaned @ proj_matrix_3d

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

xs = X_proj3d[:, 0]
ys = X_proj3d[:, 1]
zs = X_proj3d[:, 2]

ax.set_xlabel('PC1')
ax.set_ylabel('PC2')
ax.set_zlabel('PC3')

ax.scatter(xs, ys, zs)
```

Out[13]: <mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x12d5c09b0>



In the visualizations above, see that there is certainly some underlying structure in our data. In fact, the clusters we observe are related to the ethnicities of people in our dataset!

Part c

From the quantile-quantile plot in the previous part, it appears that the model is picking up on more association than theoretically expected. The reason for this is owing to the assumption of iid noise being correct. In particular, this data set contains individuals from different racial backgrounds, and also has clusters of individuals from extended families (e.g. grandparents, parents, siblings). This means that their genetics are not iid, and hence linear regression yields spurious results--all the genetic features seem to be implicated in the trait. Thus we need to modify our noise assumptions by somehow accounting for the hidden structure in the data set. The main idea is that when testing one genetic feature, all the other genetic features, jointly, are a proxy to the racial and family background. If we could include them in our model, we could correct the problem. **Ideally we would like to use all the genetic features in the linear regression model, however this is not a good idea. Why not?** Hint: There are roughly 1300 individuals and 7500 genetic variants. A written, English answer is sufficient.

Answer:

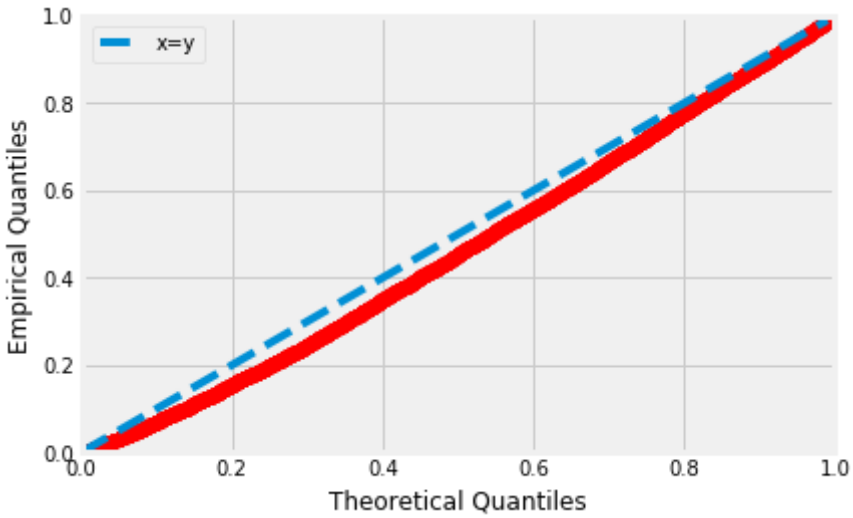
More features than number of samples, cannot solve OLS.

So instead of using all genetic features, we will try using PCA to reduce the number of genetic features. As we saw in class, PCA on a genetic similarity matrix can capture geography, which correlates to race, quite well. So instead of adding all the genetic features, we will instead use only three features (One needs to choose this number, but we have done so for you.), X_{proj}, which are the X projected onto the top 3 principal components of X. Consequently, the updated null model is $\vec{y} = X_{proj}\overrightarrow{w_{proj}} + \vec{\epsilon}$ where $\overrightarrow{w_{proj}} \in \mathbb{R}^{3\times 1}$, while the alternative model is $\vec{y} = [\overrightarrow{x_j}, X_{proj}, \vec{1}]\vec{w} + \vec{\epsilon}$ where $\vec{w} \in \mathbb{R}^{5\times 1}$ for genetic variant j. **Plot the quantile-quantile plot from obtaining p-values with this PCA linear regression approach by running the function** $pca\_corrected\_model$. **How does this plot compare to the first plot? What does this tell you about this model compared to the previous model?**

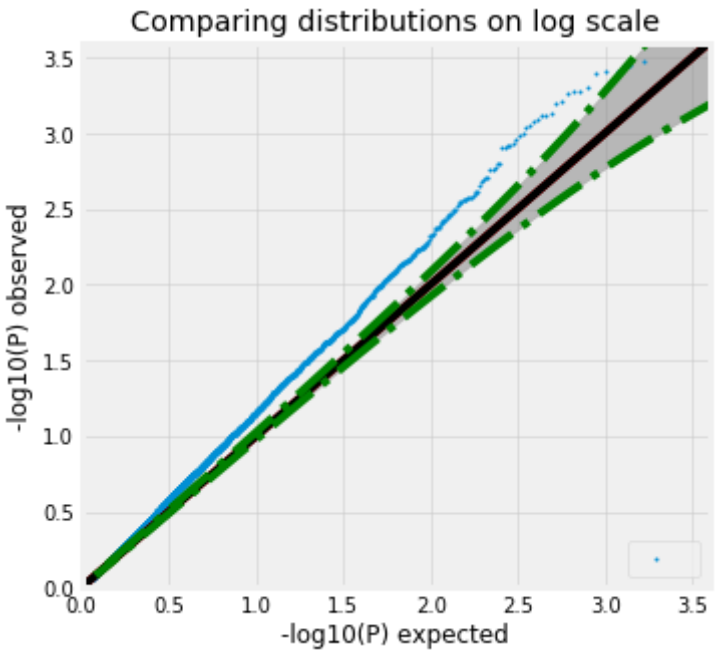
Answer:

Adding PCA features to the null hypothesis, we have removed the most important non-iid noises in the dataset, now p-values distribution are much closer to the uniform distribution than previous model. This model is clearly better than the previous one.

In [56]:
```
"""
remember that the function pca_corrected_model returns the
empirical distribution of the p-values for the pca-corrected model
"""


pca_model_pvals = pca_corrected_model(X, y)
qqplot(pca_model_pvals)
```

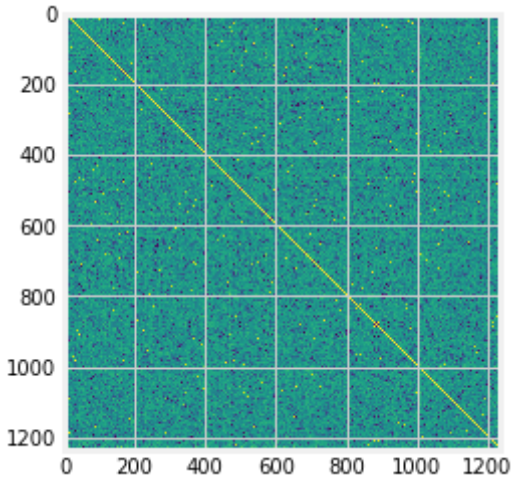



```
/Users/janeyu/anaconda/lib/python3.6/site-packages/numpy/lib/scimath.py:1
20: RuntimeWarning: invalid value encountered in less
  if any(isreal(x) & (x < 0)):
```



In [15]:
```
# Now, let's see if there is any patterns in our correlation matrix after we
# 3 largest directions of variance identified by our PCA
X_new = X - X_proj3d  @ proj_matrix_3d.T
# change the scale a bit so that patterns are more clearly visible
corr_matrix = abs(np.corrcoef(X_new))
plt.imshow(np.log(corr_matrix+0.0001))
```

Out[15]: <matplotlib.image.AxesImage at 0x117cf9d30>



We observe that there are still some patterns that appear in our correlation matrix. That seems like evidence that we haven't captured the full extent of the relationship between our datapoints so far. Can we do better?

Part d

PCA got us part of the way there. However, PCA truncates the eigenspectrum; if the tail-end of that spectrum is important, as it is for family-relatedness, then it will not fully correct for our problem. So we want a method which (a) is well-behaved in terms of number of parameters that need to be estimated, and (b) includes all of the information we need. So rather than adding the projections as features, we use an modelling approach called linear mixed models which effectively adjust the iid noise in the gaussian by the pairwise genetic similarity of all the individuals. That is, we set Σ in $\vec{y} \sim N(y|[\overrightarrow{x_j}, 1]\vec{w}, I\sigma^2 + XX^\top\sigma_k^2)$

Specifically, $\vec{y} = [\overrightarrow{x_j}, 1]\vec{w} + \vec{z} + \vec{\epsilon}$where $\vec{z} \sim N(0, \sigma_k^2 K)$where $K = XX^\top$, σ_k, $\vec{w} \in \mathbb{R}^{m\times 1}$, and σ are parameters we want to estimate. Notice that $\vec{y} \sim N([\overrightarrow{x_j}, 1]\vec{w}, \sigma^2 I + \sigma_k^2 K)$ Evaluation of the likelihood is thus on the order of $O(n^3)$ from the required matrix inverse and determinant of $\sigma^2 I + \sigma_k^2 K$. To test m genetic variants, the time complexity becomes $O(mn^3)$, which is extremely slow for large datasets. **Given the eigen-decomposition $K = UDU^\top$, how can we make this faster if we have to test thousands of genetic feature? Would this be computationally more efficient if you only have one genetic feature to test?**

Answer:

We can left multiply U.T and right multiply U to make $\sigma^2 I + \sigma_k^2 K$ a diagonal matrix, and then inverse can be easy to perform. At the same time, we should transform [xj, 1] and y by U and make them U[xj, 1], Uy.

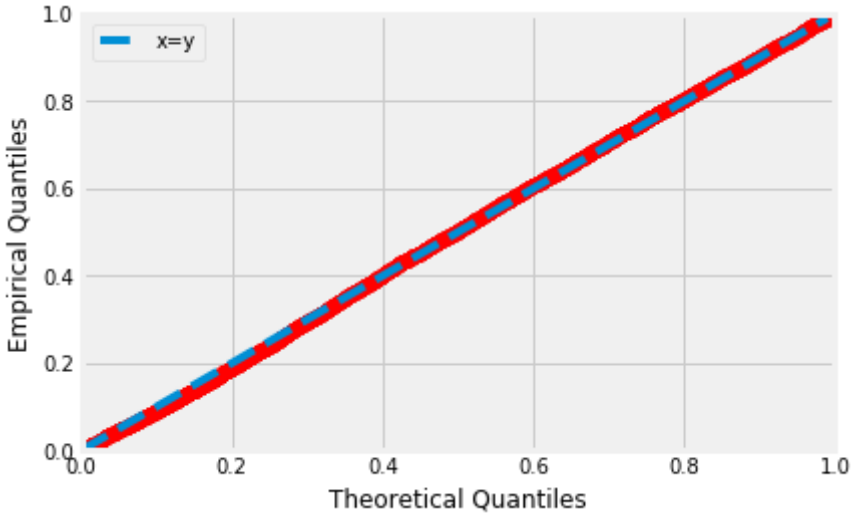
Finally, make the quantile-quantile plot for this last model by running the function `lmm`. What can we conclude about this model relative to the other two models?

HINT: Since the manipulations needed for $\sigma^2 I + \sigma_k^2 K$ is the bottleneck here, we would like a transformation which makes this covariance of the multi-variate gaussian be a diagonal matrix.

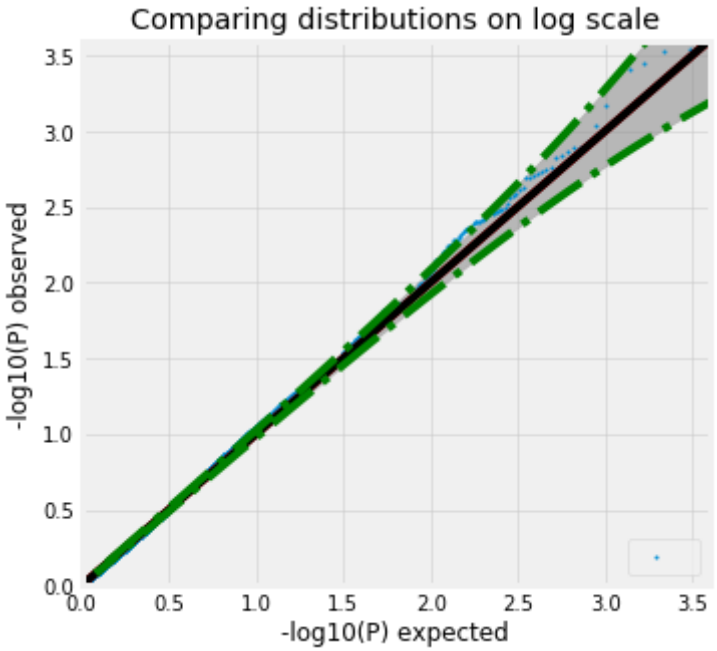
Answer

Now clearly, the p-values obey uniform distribution, after we add the non-iid noise to our model. This model is clearly better than previous models.

In [57]:
```
# SOLUTION CELL
lmm_pvals = lmm()
qqplot(lmm_pvals)
```




```
/Users/janeyu/anaconda/lib/python3.6/site-packages/numpy/lib/scimath.py:1
20: RuntimeWarning: invalid value encountered in less
  if any(isreal(x) & (x < 0)):
```



In [ ]:

In [ ]: