DS5220 - Spring 2019 Si Wu

Homework 1

1) a) joint probability distribution (non-matrix):

$$f(Y_1,Y_2) = \frac{1}{\sqrt{2\pi}\,\sigma_1\sigma_2\sqrt{1-\rho^2}} \exp\left\{-\frac{1}{2(1-\rho^2)}\left[\frac{(Y_1-\mu_1)^2}{\sigma_1^2} - 2\rho\frac{Y_1-\mu_1}{\sigma_1}\frac{Y_2-\mu_2}{\sigma_2} + \frac{(Y_2-\mu_2)^2}{\sigma_2^2}\right]\right\}$$

where μ_1 is the expected value of Y_1, μ_2 is the expected value of Y_2,
σ_1^2 is the variance of Y_1, σ_2^2 is the variance of Y_2
ρ is the coefficient of correlation for random variables Y_1 and Y_2.

joint probability distribution (matrix):

multivariate normal

$$\begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} \sim MVN\left(\begin{bmatrix}\mu_1 \\ \mu_2\end{bmatrix}, \begin{bmatrix}\sigma_1^2 & COV(Y_1,Y_2) \\ COV(Y_1,Y_2) & \sigma_2^2\end{bmatrix}\right)$$

$$\sim MVN\left(\begin{bmatrix}\mu_1 \\ \mu_2\end{bmatrix}, \begin{bmatrix}\sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2\end{bmatrix}\right)$$

where $COV(Y_1,Y_2) = E\{(Y_1-\mu_1)(Y_2-\mu_2)\}$.

covariance of Y_1, Y_2

and, $\rho = \frac{COV(Y_1,Y_2)}{\sigma_1\sigma_2} = \frac{E\{(Y_1-\mu_1)(Y_2-\mu_2)\}}{\sigma_1\sigma_2}$

so $COV(Y_1,Y_2) = \rho\sigma_1\sigma_2$.

b) $f(Y_1|Y_2) = \dfrac{f(Y_2, Y_1)}{f(Y_2)}$

$$= \frac{\frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} \exp\left\{-\frac{1}{2(1-\rho^2)}\left[\frac{(Y_2-\mu_2)^2}{\sigma_2^2} - 2\rho\frac{Y_2-\mu_2}{\sigma_2}\frac{Y_1-\mu_1}{\sigma_1} + \frac{(Y_1-\mu_1)^2}{\sigma_1^2}\right]\right\}}{\frac{1}{\sqrt{2\pi\sigma_2^2}} \exp\left[-\frac{(Y_2-\mu_2)^2}{2\sigma_2^2}\right]}.$$

$$= \frac{\sqrt{2\pi\sigma_2^2}}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} \exp\left\{-\frac{1}{2(1-\rho^2)}\left[\frac{(Y_2-\mu_2)^2}{\sigma_2^2} - 2\rho\frac{Y_2-\mu_2}{\sigma_2}\frac{Y_1-\mu_1}{\sigma_1} + \frac{(Y_1-\mu_1)^2}{\sigma_1^2}\right] + \frac{(Y_2-\mu_2)^2}{2\sigma_2^2}\right\}$$

$$= \frac{1}{\sqrt{2\pi}\,\sigma_1\sqrt{1-\rho^2}} \exp\left\{-\frac{1}{2(1-\rho^2)}\frac{(Y_2-\mu_2)^2}{\sigma_2^2} + \frac{2\rho}{2(1-\rho^2)}\frac{Y_2-\mu_2}{\sigma_2}\frac{Y_1-\mu_1}{\sigma_1} - \frac{1}{2(1-\rho^2)}\frac{(Y_1-\mu_1)^2}{\sigma_1^2} + \frac{(Y_2-\mu_2)^2}{2\sigma_2^2}\right\}$$

$$= \frac{1}{\sqrt{2\pi}\,\sigma_1\sqrt{1-\rho^2}} \exp\left\{\frac{(1-\rho^2)(Y_2-\mu_2)^2}{2(1-\rho^2)\sigma_2^2} - \frac{(Y_2-\mu_2)^2}{2(1-\rho^2)\sigma_2^2} + \frac{2\rho}{2(1-\rho^2)}\frac{Y_2-\mu_2}{\sigma_2}\frac{Y_1-\mu_1}{\sigma_1} - \frac{1}{2(1-\rho^2)}\frac{(Y_1-\mu_1)^2}{\sigma_1^2}\right\}$$

$$= \frac{1}{\sqrt{2\pi}\,\sigma_1\sqrt{1-\rho^2}} \exp\left\{\frac{(-\rho^2)(Y_2-\mu_2)^2}{2(1-\rho^2)\sigma_2^2} + \frac{2\rho}{2(1-\rho^2)}\frac{Y_2-\mu_2}{\sigma_2}\frac{Y_1-\mu_1}{\sigma_1} - \frac{1}{2(1-\rho^2)}\frac{(Y_1-\mu_1)^2}{\sigma_1^2}\right\}$$

$$= \frac{1}{\sqrt{2\pi}\,\sigma_1\sqrt{1-\rho^2}} \exp\left\{-\frac{1}{2(1-\rho^2)}\left[\frac{\rho^2(Y_2-\mu_2)^2}{\sigma_2^2} - 2\rho\frac{Y_2-\mu_2}{\sigma_2}\frac{Y_1-\mu_1}{\sigma_1} + \frac{(Y_1-\mu_1)^2}{\sigma_1^2}\right]\right\}$$

By going through the same procedure, we have

$$f(Y_2|Y_1) = \frac{1}{\sqrt{2\pi}\,\sigma_2\sqrt{1-\rho^2}} \exp\left\{-\frac{1}{2(1-\rho^2)}\left[\frac{\rho^2(Y_1-\mu_1)^2}{\sigma_1^2} - 2\rho\frac{Y_1-\mu_1}{\sigma_1}\frac{Y_2-\mu_2}{\sigma_2} + \frac{(Y_2-\mu_2)^2}{\sigma_2^2}\right]\right\}.$$

c) $\rho = \dfrac{\mathrm{COV}(Y_1, Y_2)}{\sigma_1\sigma_2} = \dfrac{E\{(Y_1-\mu_1)(Y_2-\mu_2)\}}{\sigma_1\sigma_2} = \dfrac{E\{(Y_2-\mu_2)(Y_1-\mu_1)\}}{\sigma_2\sigma_1} = \dfrac{\mathrm{COV}\{Y_2, Y_1\}}{\sigma_2\sigma_1}$

So, coefficient of correlation ρ does NOT depend on whether we want to predict Y_1 as a function of Y_2, or Y_2 as a function of Y_1

d) if $Y_1 = \theta_0 + \theta_1 Y_2 + \varepsilon$, $\varepsilon \overset{iid}{\sim} N(0, \sigma^2)$ (Y_1 fixed, and we are interested in how Y_2 varies)

However, from the data Y_{1i} and Y_{2i}, $i = 1, 2, \ldots N$, we have

$$\hat{\theta}_1 = \frac{\sum_{i=1}^{N}(Y_{2i}-\bar{Y}_2)(Y_{1i}-\bar{Y}_1)}{\sum_{i=1}^{N}(Y_{2i}-\bar{Y}_2)^2}$$

and because $\hat{\rho} = \dfrac{E\{(Y_1-\mu_1)(Y_2-\mu_2)\}}{\sigma_1\sigma_2} = \dfrac{\sum_{i=1}^{N}(Y_{1i}-\bar{Y}_1)(Y_{2i}-\bar{Y}_2)}{\sqrt{\sum_{i=1}^{N}(Y_{1i}-\bar{Y}_1)^2}\sqrt{\sum_{i=1}^{N}(Y_{2i}-\bar{Y}_2)^2}}$

So

$$\hat{\theta}_1 = \left(\frac{\sum_{i=1}^{N} (Y_{1i}-\bar{Y}_1)(Y_{2i}-\bar{Y}_2)}{\sqrt{\sum_{i=1}^{N}(Y_{1i}-\bar{Y}_1)^2}\sqrt{\sum_{i=1}^{N}(Y_{2i}-\bar{Y}_2)^2}} \right) \frac{\sqrt{\sum_{i=1}^{N}(Y_{1i}-\bar{Y}_1)^2}}{\sqrt{\sum_{i=1}^{N}(Y_{2i}-\bar{Y}_2)^2}}$$

$$= \hat{\rho} \frac{\sqrt{\sum_{i=1}^{N}(Y_{1i}-\bar{Y}_1)^2}}{\sqrt{\sum_{i=1}^{N}(Y_{2i}-\bar{Y}_2)^2}} = \hat{\rho} \frac{\hat{\sigma}_1^2 \cdot \frac{1}{N-1}}{\hat{\sigma}_2^2 \cdot \frac{1}{N-1}}$$

So this estimate from data Y_{1i}, Y_{2i}, $i = 1 \ldots N$ mirrors the quantities in the population

this mirrors this

$$E\{Y_1 | Y_2\} = \mu_1 - \mu_2 \rho \frac{\sigma_1}{\sigma_2} + \rho \frac{\sigma_1}{\sigma_2} Y_2$$

So the slope of linear regression is related, but not identical to correlation ρ.

2) a) i) it is convex

ii) A function $J(\theta): \mathbb{R}^p \to \mathbb{R}$ is convex if for any $\theta_1, \theta_2 \in \mathbb{R}^p$ and $\lambda \in [0,1]$, $J[\lambda\theta_1 + (1-\lambda)\theta_2] \leq \lambda J(\theta_1) + (1-\lambda) J(\theta_2)$

Let's say $J(x) = x^2$. $J[\lambda\theta_1 + (1-\lambda)\theta_2] = A$. $\lambda J(\theta_1) + (1-\lambda) J(\theta_2) = B$.

$$A = J[\lambda\theta_1 + (1-\lambda)\theta_2] = [\lambda\theta_1 + (1-\lambda)\theta_2]^2 = (\lambda\theta_1)^2 + 2\lambda\theta_1(1-\lambda)\theta_2 + [(1-\lambda)\theta_2]^2$$
$$= \lambda^2\theta_1^2 + 2\lambda(1-\lambda)\theta_1\theta_2 + (1-\lambda)^2\theta_2^2$$

$$B = \lambda J(\theta_1) + (1-\lambda) J(\theta_2) = \lambda\theta_1^2 + (1-\lambda)\theta_2^2$$

If we can prove $A - B \leq 0$,

we can prove $J[\lambda\theta_1 + (1-\lambda)\theta_2] \leq \lambda J(\theta_1) + (1-\lambda) J(\theta_2)$.

$$A - B = J[\lambda\theta_1 + (1-\lambda)\theta_2] - \lambda J(\theta_1) + (1-\lambda) J(\theta_2)$$
$$= \lambda^2\theta_1^2 + 2\lambda(1-\lambda)\theta_1\theta_2 + (1-\lambda)^2\theta_2^2 - \lambda\theta_1^2 - (1-\lambda)\theta_2^2$$
$$= (\lambda^2 - \lambda)\theta_1^2 + [(1-\lambda)^2 - (1-\lambda)]\theta_2^2 + 2\lambda(1-\lambda)\theta_1\theta_2$$
$$= \lambda(\lambda-1)\theta_1^2 + [1 - 2\lambda + \lambda^2 - 1 + \lambda]\theta_2^2 + 2\lambda(1-\lambda)\theta_1\theta_2$$
$$= \lambda(\lambda-1)\theta_1^2 + [-\lambda + \lambda^2]\theta_2^2 + 2\lambda(1-\lambda)\theta_1\theta_2$$
$$= \lambda(\lambda-1)\theta_1^2 + \lambda(\lambda-1)\theta_2^2 + 2\lambda(1-\lambda)\theta_1\theta_2$$
$$= \lambda(\lambda-1)[\theta_1^2 - 2\theta_1\theta_2 + \theta_2]^2$$
$$= \lambda(\lambda-1)(\theta_1 - \theta_2)^2$$

$(\theta_1 - \theta_2)^2$ is always > 0.

and when $\lambda = 0$ or 1, $A - B = 0$.

When $0 < \lambda < 1$, $A - B < 0$.

So, $A - B \leq 0$ at all times.

So, $J[\lambda\theta_1 + (1-\lambda)\theta_2] \leq \lambda J(\theta_1) + (1-\lambda) J(\theta_2)$ at all times.

So, this function $J(x) = x^2$ is convex.

And, we know that,

If $f_1, \ldots f_n$ are convex functions, $x \in \mathbb{R}^p$ and $\omega_1, \ldots \omega_n \geq 0$, then $f(x) = \omega_1 f_1(x) + \ldots + \omega_n f_n(x)$ is convex.

So, $L = \sum_{i=1}^{N} e_i^2$ is convex.

iii) It is useful in the context of linear regression because if it's convex, we know it has a global minimum point. And we use that point for optimization problems.

b) i) it is convex.

ii). Let's say $J(x) = |x|$.

$J[\lambda\theta_1 + (1-\lambda)\theta_2] = A$. $\quad \lambda J(\theta_1) + (1-\lambda) J(\theta_2) = B$.

$A = J[\lambda\theta_1 + (1-\lambda)\theta_2] = |\lambda\theta_1 + (1-\lambda)\theta_2|$

$B = \lambda J(\theta_1) + (1-\lambda) J(\theta_2) = \lambda|\theta_1| + (1-\lambda)|\theta_2|$

If we can prove $A \leq B$, we can prove $J[\lambda\theta_1 + (1-\lambda)\theta_2] \leq \lambda J(\theta_1) + (1-\lambda) J(\theta_2)$.

If $\theta_1 < 0$, $\theta_2 \geq 0$

or $\theta_1 \geq 0$, $\theta_2 < 0$

A = the <u>difference</u> between $|\lambda\theta_1|$ and $|(1-\lambda)\theta_2|$

B = the <u>sum</u> of $|\lambda\theta_1|$ and $|(1-\lambda)\theta_2|$

so $A < B$.

if $\theta_1 > 0$, $\theta_2 > 0$
or $\theta_1 < 0$ $\theta_2 < 0$
both A and B are the sum of $|\lambda\theta_1|$ and $|(1-\lambda)\theta_2|$
so $A = B$.

So, $A \leq B$ at all times. So, $J[\lambda\theta_1 + (1-\lambda)\theta_2] \leq \lambda J(\theta_1) + (1-\lambda) J(\theta_2)$ at all times.

So, this function $J(x) = |x|$ is convex.

And, we know that,
If $f_1, \dots f_n$ are convex functions, $X \in \mathbb{R}^p$ and $\omega_1, \dots \omega_n \geq 0$, then
$f(x) = \omega_1 f_1(x) + \dots + \omega_n f_n(x)$ is convex.
So, $L = \sum_{i=1}^{N} |e_i|$ is convex.

iii) It is useful in the context of linear regression because if it's convex, we know it has a global minimum point. And we use that point for optimization problems.

c). i). it is convex.

ii). if $|e| \le \delta$, $L = \sum_{i=1}^{N} l(e_i) = \sum_{i=1}^{N} \frac{1}{2} e^2$

and we know from part (a) that $L = \sum_{i=1}^{N} e_i^2$ is convex.

so $L = \sum_{i=1}^{N} \frac{1}{2} e^2$ is convex.

if $|e| > \delta$. $L = \sum_{i=1}^{N} l(e_i) = \sum_{i=1}^{N} \delta |e| - \frac{1}{2}\delta^2 = \delta \sum_{i=1}^{N} |e_i| - \frac{1}{2}\delta^2 N$

and we know from part (b) that $L = \sum_{i=1}^{N} |e_i|$ is convex

so $L = \sum_{i=1}^{N} \delta |e| - \frac{1}{2}\delta^2$ is convex.

so, $L = \sum_{i=1}^{N} l(e_i)$, where $l(e) = \begin{cases} \frac{1}{2} e^2 & \text{if } |e| \le \delta \\ \delta |e| - \frac{1}{2}\delta^2 & \text{if } |e| > \delta \end{cases}$ is convex.

3)a) $Y_i = \theta_0 + \theta_1 X_i + e_i, \quad i=1, \dots N$

$$E(Y_i) = E(\theta_0 + \theta_1 X_i + e_i)$$

$$= E(\theta_0 + \theta_1 X_i) + \underbrace{\cancel{E(e_i)}}_{=0}$$

$$e_i \overset{iid}{\sim} N(0, \sigma^2)$$

constant variance

so, $Var(e_i) = \sigma^2$.

So, $Y_i | X_i \sim N(\theta_0 + \theta_1 X_i, \sigma^2)$.

a normal distribution with mean value $\theta_0 + \theta_1 X_i$, and variance σ^2.

So, probability density function of y

$$f(Y_i | \theta_0, \theta_1, X_i, \sigma)$$

$$= \frac{1}{\sqrt{2\pi}\,\sigma} e^{-\frac{1}{2\sigma^2}[Y_i - (\theta_0 + \theta_1 X_i)]^2}$$

likelihood on training data

$$L(\theta_0 \; \theta_1 | X_i \; Y_i \; \sigma^2)$$

$$= \frac{1}{\sqrt{2\pi}\,\sigma} e^{-\frac{1}{2\sigma^2}[Y_i - (\theta_0 + \theta_1 X_i)]^2}$$

This is the same expression but views the observed Y_i, X_i and σ as fixed, and $\underset{2\times 1}{\theta}$ as unknown so $L(\theta_0, \theta_1 | X_1, \ldots X_N, Y_1 \ldots Y_N, \sigma^2)$

$$= \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi}\,\sigma} e^{-\frac{1}{2\sigma^2}[Y_i - (\theta_0 + \theta_1 X_i)]^2}$$

$$\therefore \ln L = \ln\left[\prod_{i=1}^{N} \frac{1}{\sqrt{2\pi}\,\sigma} e^{-\frac{1}{2\sigma^2}[Y_i - (\theta_0 + \theta_1 X_i)]^2}\right]$$

$$= \underbrace{\ln\frac{1}{\sqrt{2\pi}\,\sigma} + \ln\frac{1}{\sqrt{2\pi}\,\sigma} + \cdots \ln\frac{1}{\sqrt{2\pi}\,\sigma}}_{N} + \left(-\frac{1}{2\sigma^2}\right)[Y_1 - (\theta_0 + \theta_1 X_1)]^2 + \ldots + \left(-\frac{1}{2\sigma^2}\right)[Y_N - (\theta_0 + \theta_1 X_N)]^2$$

$$= N \ln\frac{1}{\sqrt{2\pi}\,\sigma} + \underline{\sum_{i=1}^{N}\left(-\frac{1}{2\sigma^2}\right)[Y_i - (\theta_0 + \theta_1 X_i)]^2}.$$

need to minimize this to find maximum likelihood solution.

Let's say $J_1(\theta) = N \ln\frac{1}{\sqrt{2\pi}\,\sigma} + \sum_{i=1}^{N}\left(-\frac{1}{2\sigma^2}\right)[Y_i - (\theta_0 + \theta_1 X_i)]^2$

to minimize $J_1(\theta)$,

$$\frac{\partial J_1(\theta)}{\partial \theta_0} = -\frac{1}{2\sigma^2}\sum_{i=1}^{N} 2[Y_i - \theta_0 - \theta_1 X_i](-1) \overset{\text{set to}}{=} 0$$

$$\frac{\partial J_1(\theta)}{\partial \theta_1} = -\frac{1}{2\sigma^2}\sum_{i=1}^{N} 2[Y_i - \theta_0 - \theta_1 X_i](-X_i) \overset{\text{set to}}{=} 0$$

So, $\sum_{i=1}^{N} y_i - \theta_0 - \theta_1 x_i = 0$

$$\sum_{i=1}^{N} y_i = \sum_{i=1}^{N} (\theta_0 + \theta_1 x_i)$$

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_N \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} + \underbrace{\begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_N \end{bmatrix}}_{\text{error.}}$$

$$Y_{N\times 1} = X_{N\times 2}\,\theta_{2\times 1} + e_{N\times 1}$$

if $J_2(\theta) = \sum_{i=1}^{N} (y_i - x_i^T \theta)^2$

minimizing squared loss function $J_2(\theta)$ (views the observed y_i, x_i as fixed, and $\underset{2\times 1}{\theta}$ as unknown)

$$J_2(\theta) = \sum_{i=1}^{N} (y_i - x_i^T\theta)^2 = \|\underset{N\times 1}{Y} - \underset{N\times 2}{X}\,\underset{2\times 1}{\theta}\|_2^2$$

$$\boxed{\begin{aligned} \frac{\partial A\theta}{\partial \theta} &= A^T \\ \frac{\partial \theta^T A \theta}{\partial \theta} &= 2A\theta \end{aligned}}$$

$$= (Y - X\theta)^T (Y - X\theta)$$
$$= Y^T Y - \theta^T X^T Y - Y^T X\theta + \theta^T X^T X \theta$$

Gradient in matrix notation

$$\nabla J(\theta) = -X^T Y - (Y^T X)^T + 2X^T X\theta$$
$$= -2X^T Y + 2X^T X\theta \overset{\text{set to}}{=} 0$$

so $-2X^TY = 2X^TX\theta$

so solution $\hat{\theta} = \frac{X^T \cdot Y}{X^TX} = (X^TX)^{-1}X^TY.$

b) gradient $\nabla J(\theta) = -2X^TY + 2X^TX\theta$. This was derived in part a).

c) batch gradient descent for linear regression

- parameters : α_{max}, $\tau \in [0.5, 0.9]$, tolerance, max Backtrack
- Initialize θ
- $\alpha \leftarrow \alpha_{max}$
- Repeat until convergence {
  - $\theta^{try} \leftarrow \theta$
  - obj $\leftarrow J(\theta)$
  - Repeat maxBacktrack times {

    – For $j = 1, \ldots P$ {

    start over if did not improve

    $\theta_j^{try} \leftarrow \theta_j + \alpha \underline{\frac{\partial}{\partial \theta_j} \sum_{i=1}^{N} (Y_i - X_i^T\theta)^2}$

    this is the gradient with respect to the parameter vector

    }

    – If $J(\theta_j^{try}) <$ obj – tolerance, then $\theta \leftarrow \theta^{try}$; break $\leftarrow$ improved

    else $\alpha \leftarrow \tau\alpha$

  }

- if maxBacktrack is reached, $\alpha \leftarrow 0$; Break

}.

d) stochastic gradient descent for linear regression

- Initialize θ
- Repeat until convergence {
    - shuffle the order of observations, 1, ... N
    - for i in 1 ... N {
        - for j in 1, ... p {

$$\theta_j \leftarrow \theta_j + \alpha \frac{d}{d\theta_j} (y_i - x_i^T \theta)^2$$

        }

    }

}.

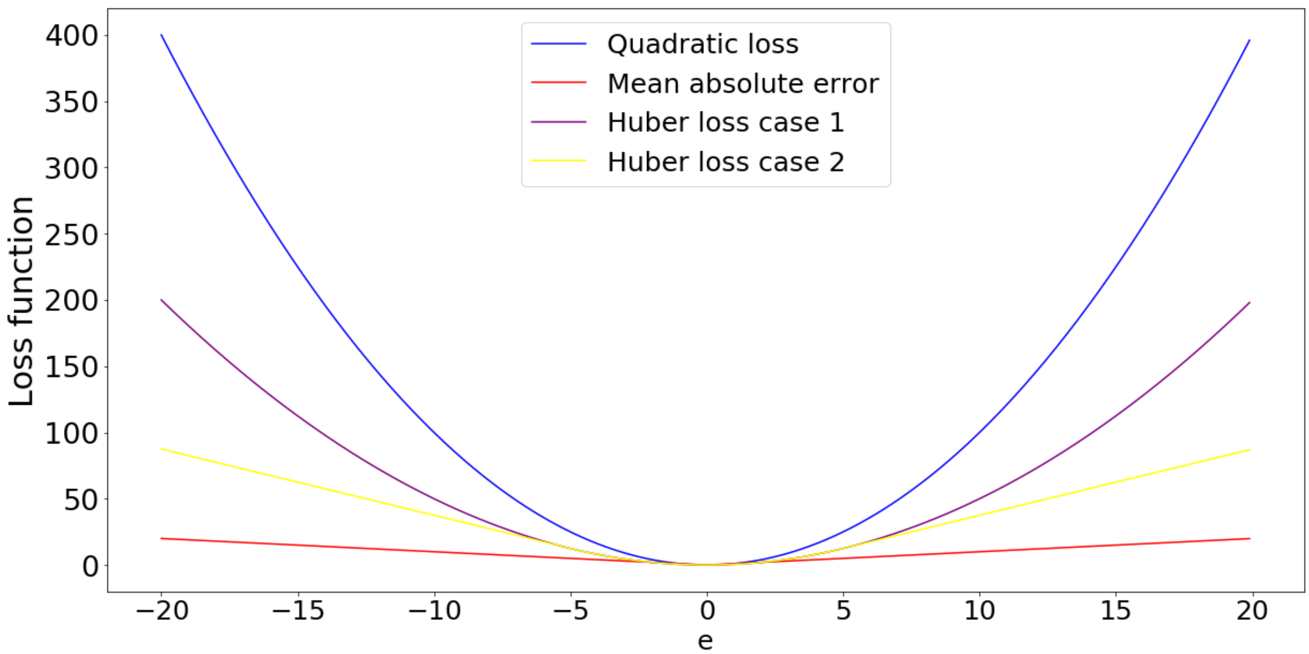
In [2]:

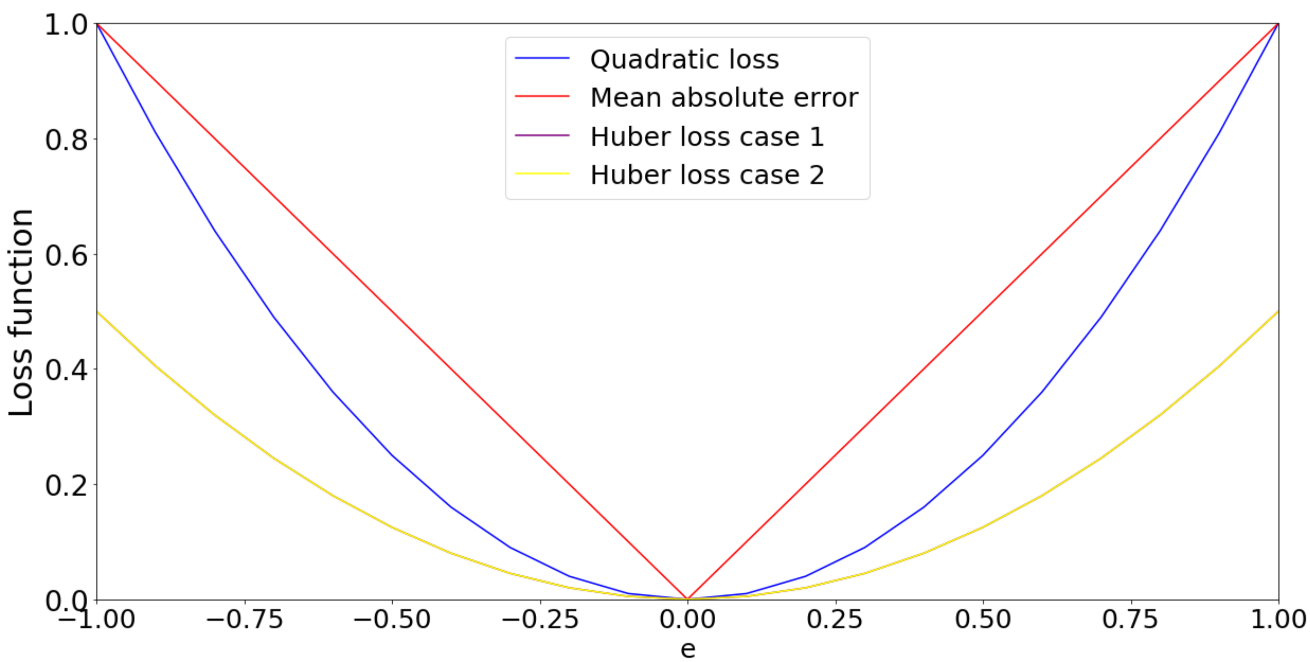
```
import matplotlib.pyplot as plt
import numpy as np
from numpy.linalg import inv
from matplotlib import pyplot
```

In [71]:

```
#Question 4a
e=[]
L_a=[]
L_b=[]
L_c1=[]
L_c2=[]
delta1=50
delta2=5

for i in range(-200,200):
    e.append(i/10)
    a=e[-1]**2
    b=abs(e[-1])
    if abs(e[-1]) <= delta1:
        c1=0.5*(e[-1]**2)
    else:
        c1=delta1*abs(e[-1])-.5*(delta1**2)
    if abs(e[-1]) <= delta2:
        c2=0.5*(e[-1]**2)
    else:
        c2=delta2*abs(e[-1])-.5*(delta2**2)
    L_a.append(a)
    L_b.append(b)
    L_c1.append(c1)
    L_c2.append(c2)

fig=plt.figure(figsize=(20,10))
plt.plot(e,L_a, color='blue')
plt.plot(e,L_b, color='red')
plt.plot(e,L_c1, color='purple')
plt.plot(e,L_c2, color='yellow')
plt.legend(["Quadratic loss", "Mean absolute error", "Huber loss case 1", "Huber
loss case 2"],fontsize=25)
plt.ylabel('Loss function', fontsize = 30)
plt.xlabel('e', fontsize=25)
plt.yticks(fontsize =27)
plt.xticks(fontsize =25)
plt.show()

fig=plt.figure(figsize=(20,10))
plt.plot(e,L_a, color='blue')
plt.plot(e,L_b, color='red')
plt.plot(e,L_c1, color='purple')
```

```
plt.plot(e,L_c1, color='purple')
plt.plot(e,L_c2, color='yellow')
plt.legend(["Quadratic loss", "Mean absolute error", "Huber loss case 1", "Huber loss case 2"],fontsize=25)
plt.ylabel('Loss function', fontsize = 30)
plt.xlabel('e', fontsize=25)
plt.yticks(fontsize =27)
plt.xticks(fontsize =25)
plt.xlim([-1,1])
plt.ylim([0,1])
plt.show()
```





We can see from the two graphs above that the curve for quadratic loss function has the largest gradient for large e, while the curve for mean absolute error has the smallest gradient for large e. Quadratic loss is effective for large e because it is highly sensitive in this region. For small e (i.e. $|e| < 1$), mean absolute error is more sensitive. Huber loss is sort of a "happy medium" between the two.

In [194]:

```
#question 4b
def gradient_descent(x, y, initial_theta, loss_func, alpha=0.01, precision=0.001
):
    loss = []
    theta = initial_theta
    all_thetas1 = [] # to store all thetas
    predictions = [] # to store all predictions
    number_of_steps = 0
    previous_loss = 0
    prediction = np.dot(x,theta) #dot product
    error = prediction - y
    current_loss = loss_func(error)
    predictions.append(prediction)
    loss.append(current_loss)
    all_thetas1.append(theta)
    number_of_steps+=1
    while abs(current_loss - previous_loss) > precision: #if the difference betw
een current and previous values of loss function is bigger than the precision we
set
        previous_loss = current_loss #we update the value of the loss function t
o be the current value of loss function
        gradient = np.dot(x.T,error) #new gradient
        theta = theta - alpha * gradient  #update theta
        all_thetas1.append(theta)

        prediction = np.dot(x,theta)
        error = prediction - y
        current_loss = loss_func(error)
        loss.append(current_loss)

    return all_thetas1, loss, predictions

def loss_func_a(error): #squared loss
    return np.sum(error**2)

def loss_func_b(error): #mean absolute error
    return np.sum(abs(error))

def loss_func_c(error): #Huber loss case 1
    L = 0
    for i in range(len(error)):
        if abs(error[i]) <= 5:
            L += 0.5*(error[i]**2)
        else:
            L += 5*abs(error[i])-.5*(5**2)
    return L

def loss_func_d(error): #Huber loss case 2
    L = 0
    for i in range(len(error)):
        if abs(error[i]) <= .5:
            L += 0.5*(error[i]**2)
```

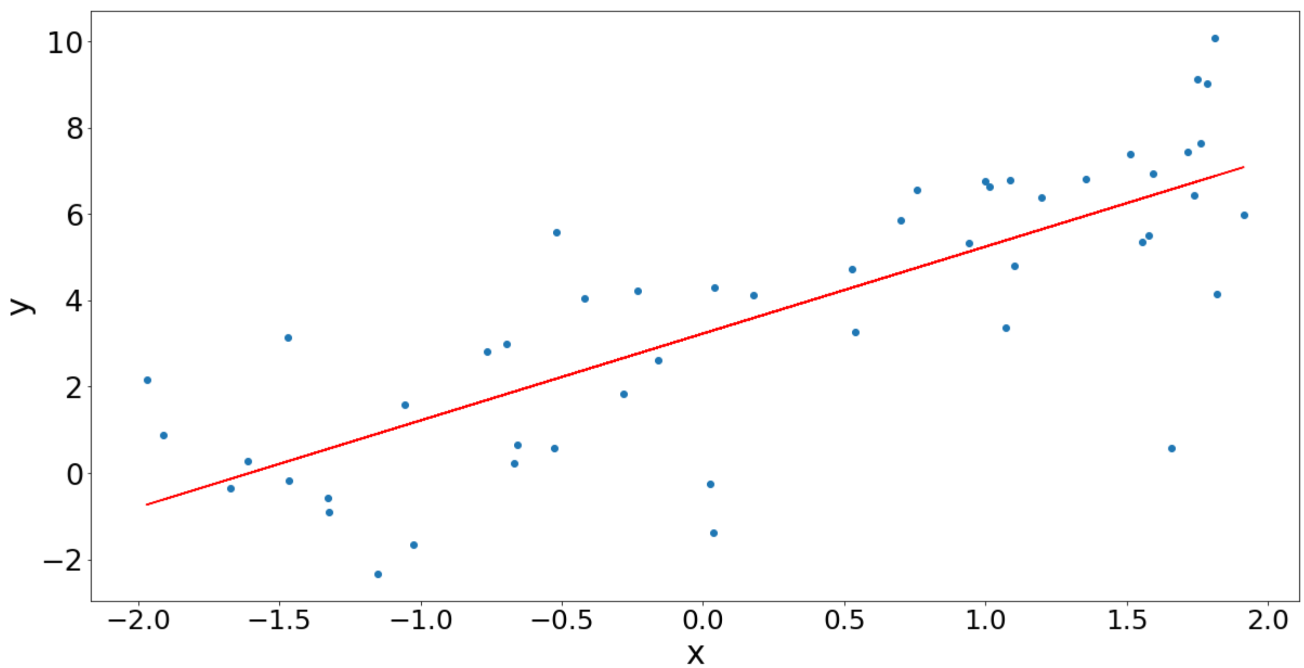
```
        else:

            L += .5*abs(error[i])-.5*(.5**2)
    return L
```

In [195]:

```
#question 4c
def stochastic_gradient_descent(x, y, initial_theta, loss_func, alpha=0.01, prec
ision=0.001):
    current_loss = []
    predictions = []
    all_thetas = []
    number_of_steps = 0
    previous_loss = 0
    epoch = 0
    i = 0 #index
    theta = initial_theta
    prediction = np.dot(x[i,:],theta)
    error = prediction - y[i]
    gradient = x[i,:].T*error
    current_loss.append(loss_func(error))
    number_of_steps+=1
    predictions.append(prediction)
    all_thetas.append(theta)

    while abs(current_loss[number_of_steps-1]-previous_loss) > precision:
        gradient = x[i,:].T*error
        theta = theta - alpha * gradient
        all_thetas.append(theta)
        i += 1
        if i == y.size:
            epoch +=1
            reorder = np.random.permutation(y.size)
            x = x[reorder]
            y = y[reorder]
            i = 0
        prediction = np.dot(x[i,:], theta)
        error = prediction - y[i]
        previous_loss = current_loss[number_of_steps-1]
        current_loss.append(loss_func(error))
        number_of_steps += 1

    return all_thetas, current_loss, predictions
```

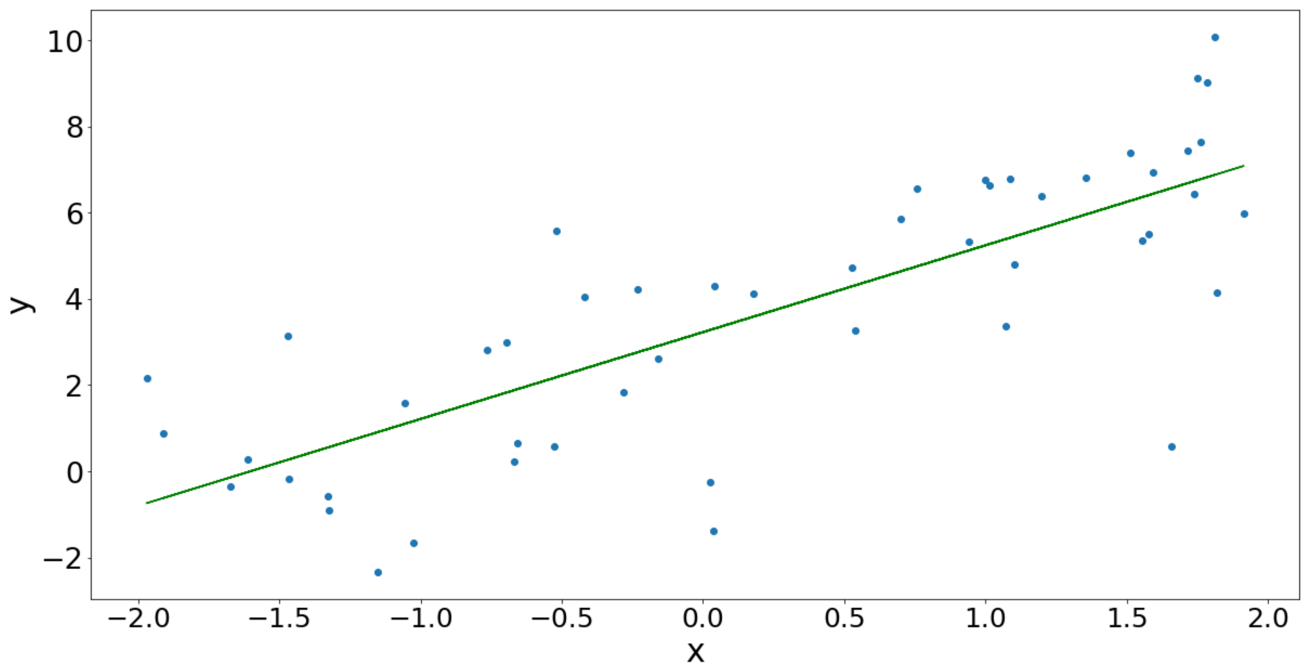
In [196]:

```
#question 5a i)
x=np.random.uniform(low=-2, high=2, size=(50,))
y=3+2*x+np.random.normal(0,2,50)

matrix_x = np.c_[np.ones(x.shape[0]), x]
theta_ai= np.dot(inv(np.dot(matrix_x.T,matrix_x)),np.dot(matrix_x.T,y)) #This is the analytical solution of theta

y_analytical_solution=np.dot(matrix_x,theta_ai)

fig=plt.figure(figsize=(20,10))
plt.scatter(x,y)
plt.plot(x,y_analytical_solution,color='red')
plt.xlabel('x', fontsize=30)
plt.ylabel('y', fontsize=30)
plt.yticks(fontsize =27)
plt.xticks(fontsize =25)
plt.show()
```



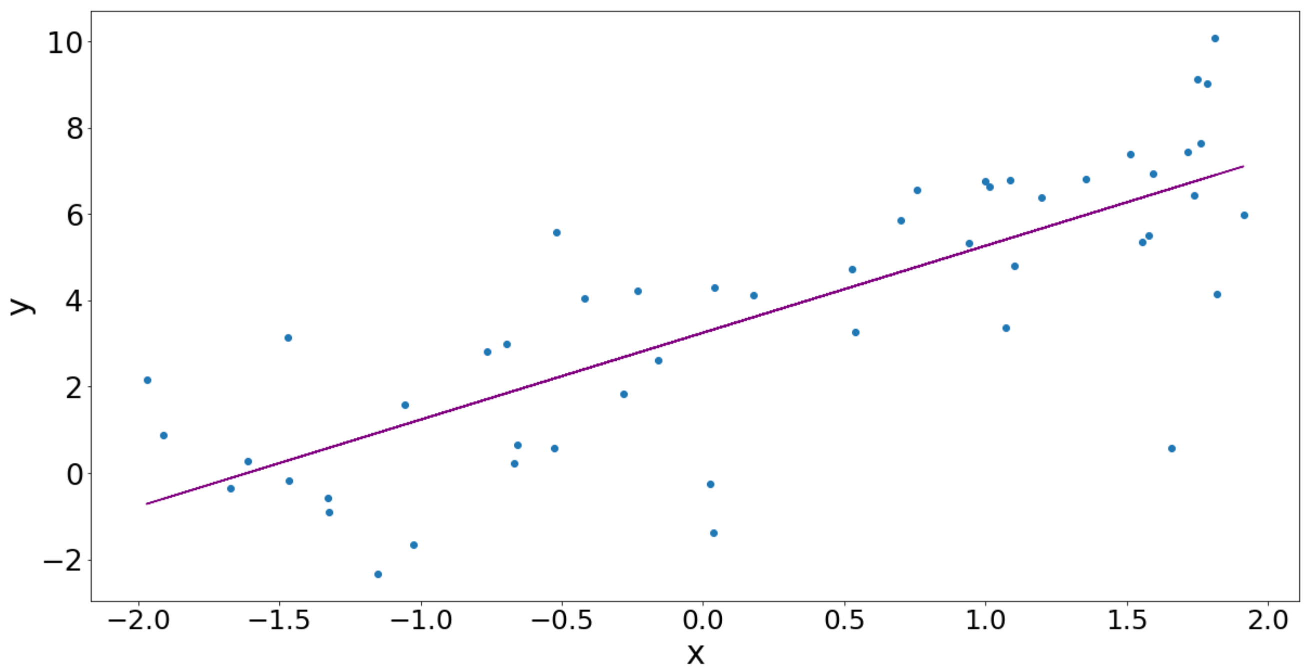
In [197]:

```
#question 5a) ii)
matrix_x = np.c_[np.ones(x.shape[0]), x]
theta_i = np.random.rand(2)
all_thetas1, loss, predictions = gradient_descent(matrix_x, y, theta_i, loss_fun
c_a)

theta_aii = all_thetas1[-1]

y_gradient_descent=np.dot(matrix_x,theta_aii)

fig=plt.figure(figsize=(20,10))
plt.scatter(x,y)
plt.plot(x,y_gradient_descent,color='green')
plt.xlabel('x', fontsize=30)
plt.ylabel('y', fontsize=30)
plt.yticks(fontsize =27)
plt.xticks(fontsize =25)
plt.show()
```



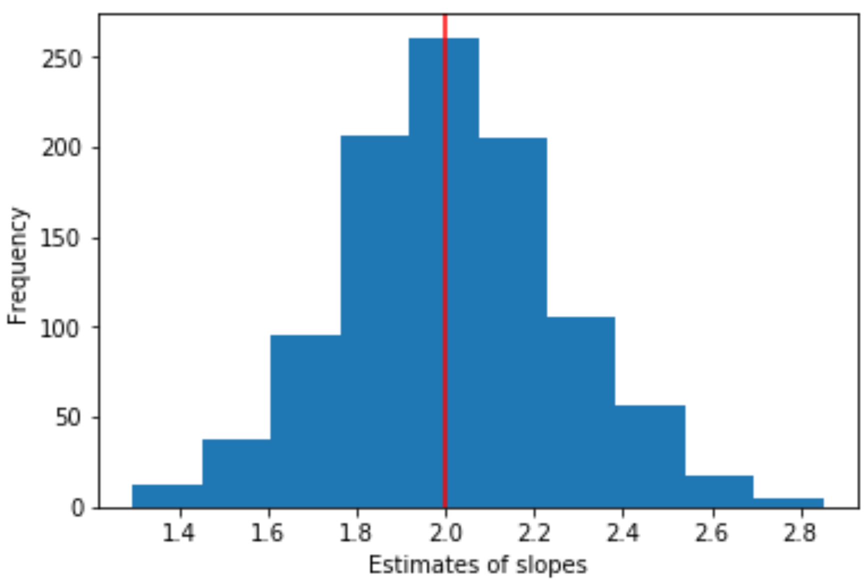
In [198]:

```
#question 5a) iii)
matrix_x = np.c_[np.ones(x.shape[0]), x]
theta_i = np.random.rand(2)
all_thetas2, loss, predictions = stochastic_gradient_descent(matrix_x, y, theta_
i, loss_func_a)

theta_aiii = all_thetas2[-1]

y_stochastic_gradient_descent=np.dot(matrix_x,theta_aiii)

fig=plt.figure(figsize=(20,10))
plt.scatter(x,y)
plt.plot(x,y_stochastic_gradient_descent,color='purple')
plt.xlabel('x', fontsize=30)
plt.ylabel('y', fontsize=30)
plt.yticks(fontsize =27)
plt.xticks(fontsize =25)
plt.show()
```



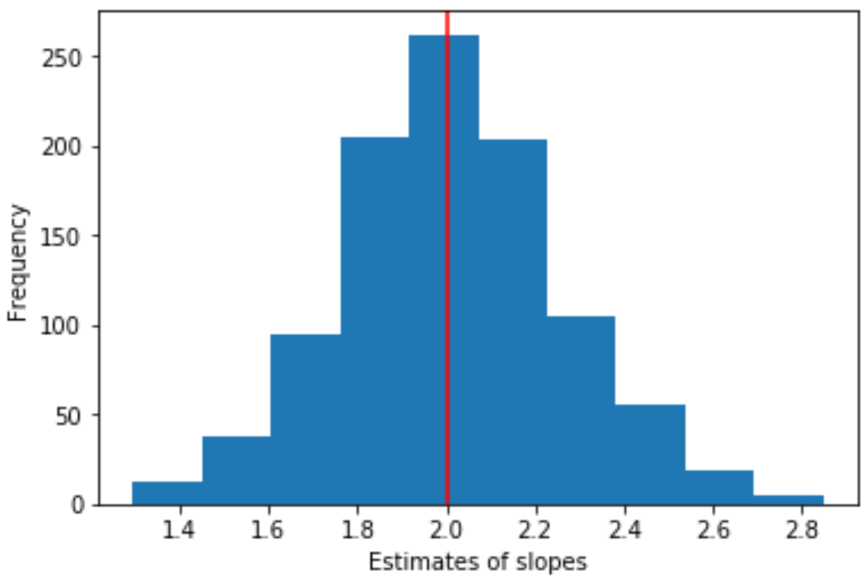
In [172]:

```
#question 5bi) — repeating steps in part a) 1000 times for analytical solution

theta = []
slopes_a = []
x = np.zeros((50,1000))
y = np.zeros((50,1000))
for d in range(1000):
    x[:,d] = np.random.uniform(low=-2, high=2, size=(50,))
    y[:,d] = 3+2*x[:,d]+np.random.normal(0,2,50)
    matrix_x = np.c_[np.ones(x[:,d].shape[0]), x[:,d]]
    theta.append(np.dot(inv(np.dot(matrix_x.T,matrix_x)),np.dot(matrix_x.T,y[:,d
]))) #This is the analytical solution of theta
    slopes_a.append(theta[d-1][1])


plt.hist(slopes_a)
plt.xlabel('Estimates of slopes')
plt.ylabel('Frequency')
plt.axvline(x=2.0, color='red')
plt.show()
```



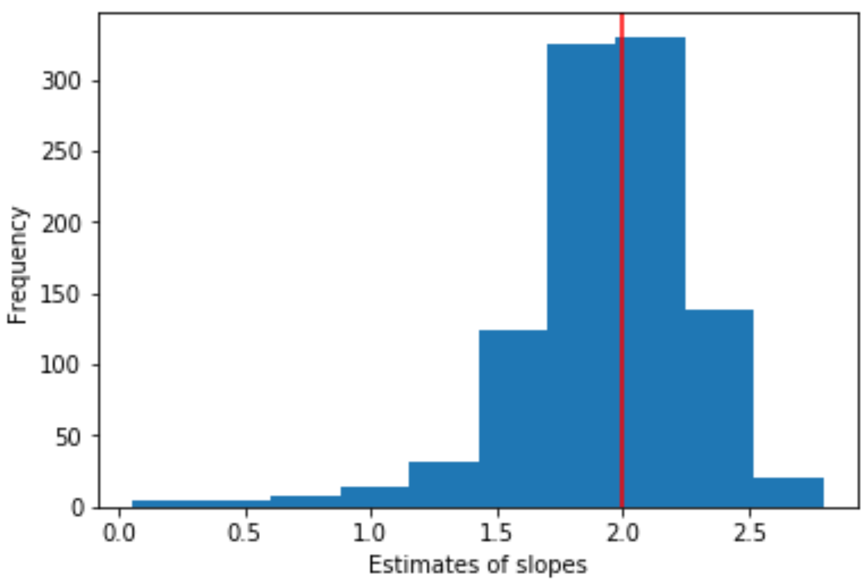
In [173]:

```
#question 5bii) — repeating steps in part a) 1000 times for batch gradient desce
nt
theta_b = []
slopes_b = []
for d in range(1000):
    matrix_x = np.c_[np.ones(x[:,d].shape[0]), x[:,d]]
    theta_i = np.random.rand(2)
    all_thetas2, loss, predictions = gradient_descent(matrix_x, y[:,d], theta_i,
loss_func_a)
    theta_b.append(all_thetas2[-1])
    slopes_b.append(theta_b[d-1][1])

plt.hist(slopes_b)
plt.xlabel('Estimates of slopes')
plt.ylabel('Frequency')
plt.axvline(x=2.0, color='red')
plt.show()
```



In [174]:

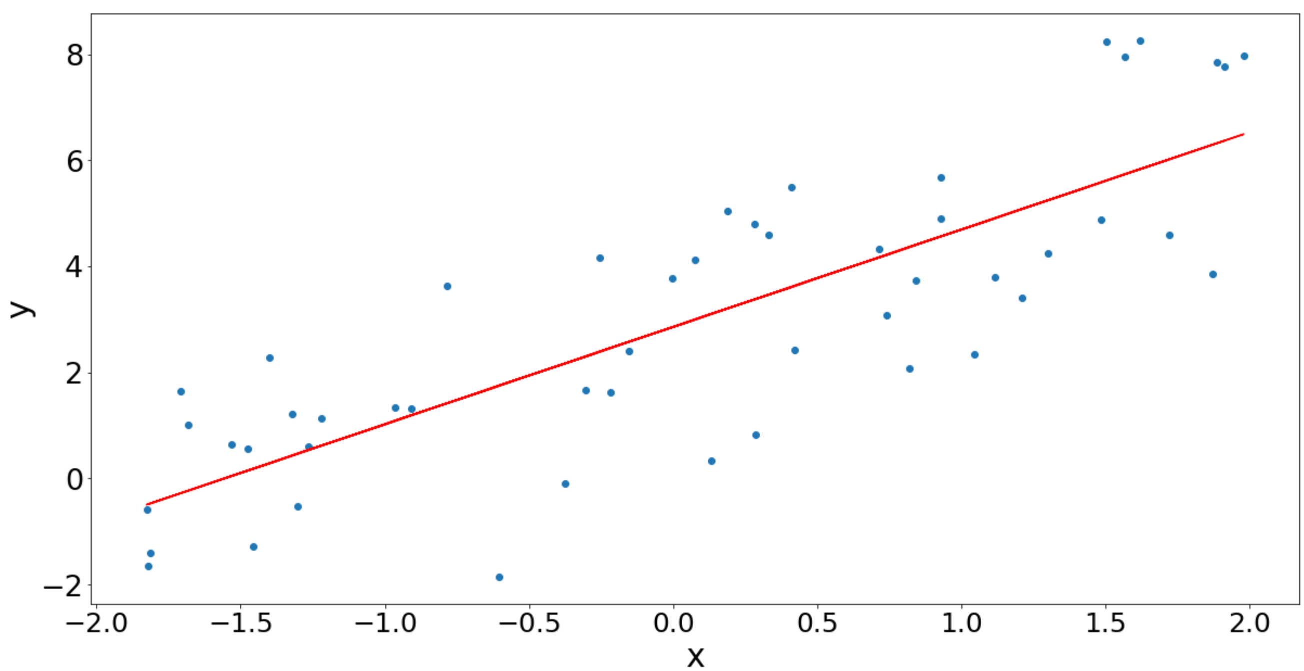
```
#question 5biii) — repeating steps in part a) 1000 times for stochastic gradient
descent
theta_c = []
slopes_c = []
for d in range(1000):
    matrix_x = np.c_[np.ones(x[:,d].shape[0]), x[:,d]]
    theta_i = np.random.rand(2)
    all_thetas2, loss, predictions = stochastic_gradient_descent(matrix_x, y[:,d
], theta_i, loss_func_a)
    theta_c.append(all_thetas2[-1])
    slopes_c.append(theta_c[d-1][1])

plt.hist(slopes_c)
plt.xlabel('Estimates of slopes')
plt.ylabel('Frequency')
plt.axvline(x=2.0, color='red')
plt.show()
```



How the choice of algorithm affects the estimates of the slope parameter: in stochastic gradient descent, the algorithm looks at a randomly selected subset of data points each time, so the estimates are more spread out. It's messier and less accurate than the other two methods.

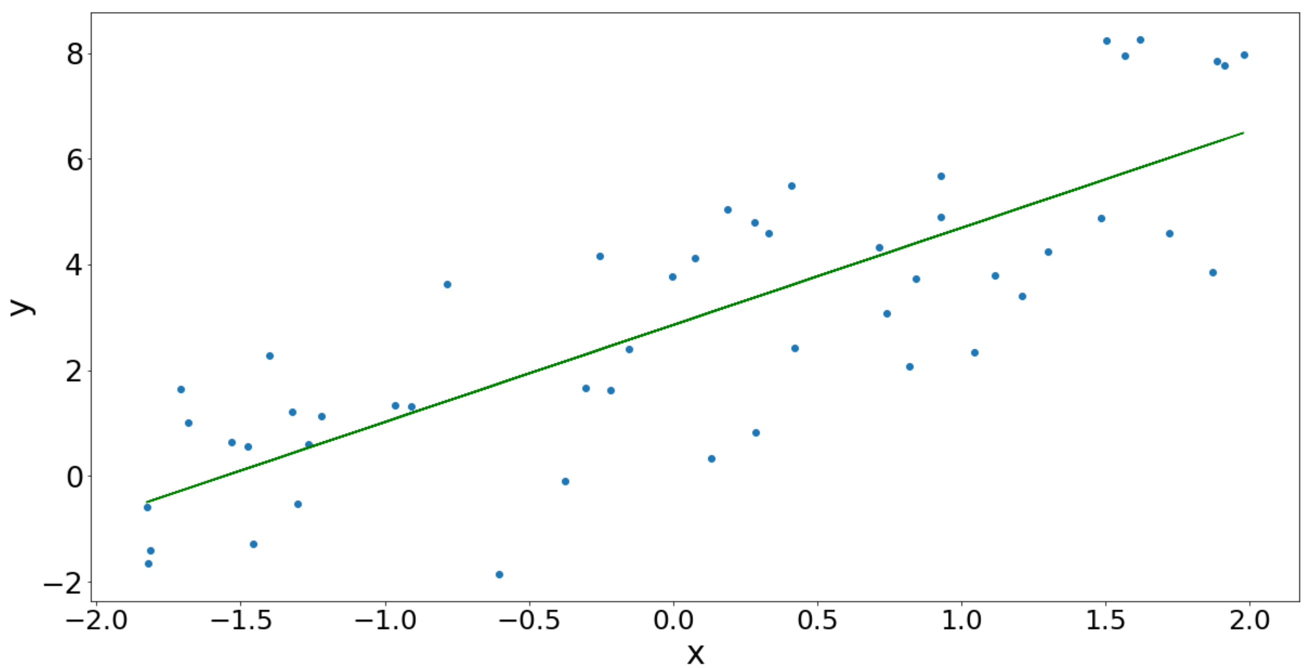
In [176]:

```
#question 5c) i)
x=np.random.uniform(low=-2, high=2, size=(50,))
y=3+2*x+np.random.normal(0,2,50)

matrix_x = np.c_[np.ones(x.shape[0]), x]
theta_ci = np.dot(inv(np.dot(matrix_x.T,matrix_x)),np.dot(matrix_x.T,y)) #This i
s the analytical solution of theta

y_analytical_solution=np.dot(matrix_x,theta_ci)

fig=plt.figure(figsize=(20,10))
plt.scatter(x,y)
plt.plot(x,y_analytical_solution,color='red')
plt.xlabel('x', fontsize=30)
plt.ylabel('y', fontsize=30)
plt.yticks(fontsize =27)
plt.xticks(fontsize =25)
plt.show()
```



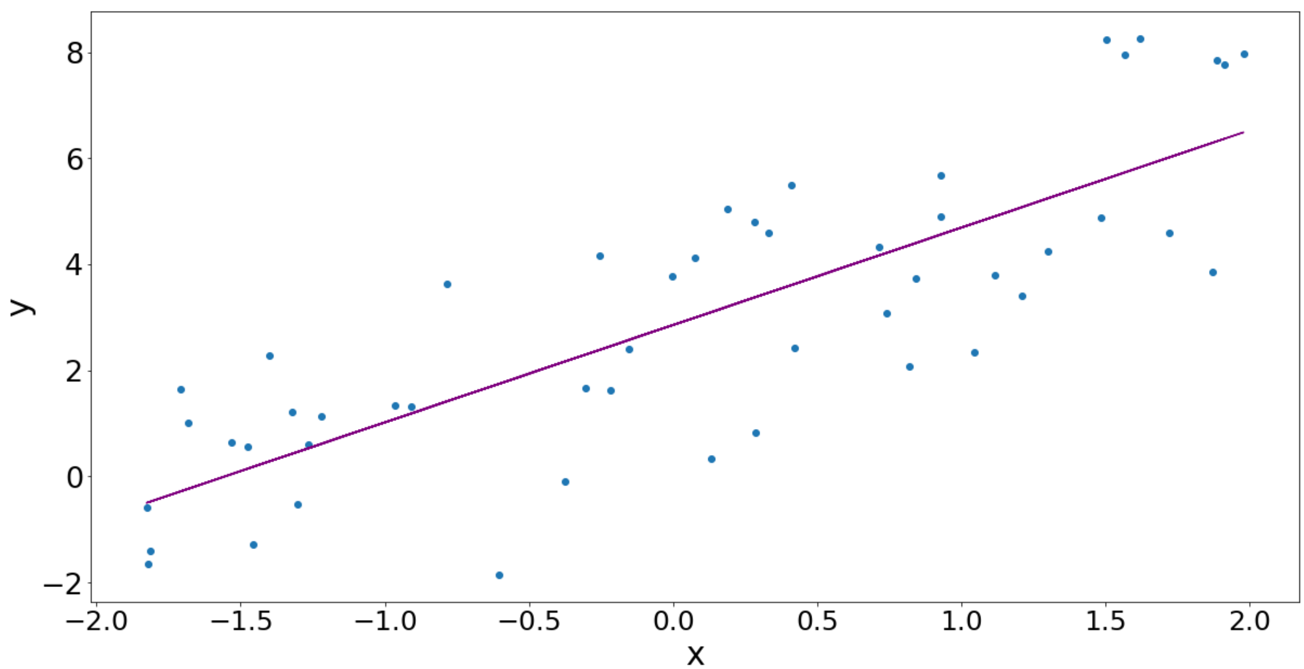
In [177]:

```
#question 5c) ii)
matrix_x = np.c_[np.ones(x.shape[0]), x]
theta_i = np.random.rand(2)
all_thetas1, loss, predictions = gradient_descent(matrix_x, y, theta_i, loss_fun
c_b)

theta_cii = all_thetas1[-1]

y_gradient_descent=np.dot(matrix_x,theta_cii)

fig=plt.figure(figsize=(20,10))
plt.scatter(x,y)
plt.plot(x,y_gradient_descent,color='green')
plt.xlabel('x', fontsize=30)
plt.ylabel('y', fontsize=30)
plt.yticks(fontsize =27)
plt.xticks(fontsize =25)
plt.show()
```



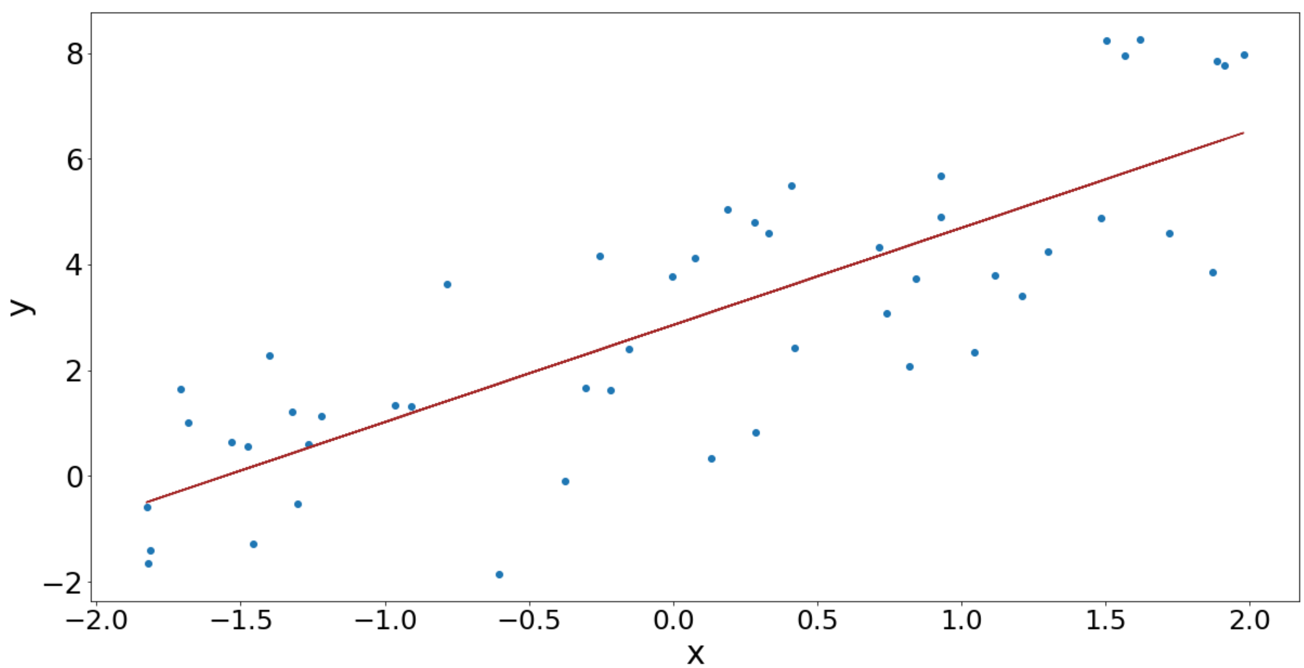
In [178]:

```
#question 5c) iii Huber loss case 1
matrix_x = np.c_[np.ones(x.shape[0]), x]
theta_i = np.random.rand(2)
all_thetas2, loss, predictions = gradient_descent(matrix_x, y, theta_i, loss_fun
c_c)

theta_ciii = all_thetas2[-1]

y_stochastic_gradient_descent=np.dot(matrix_x,theta_ciii)

fig=plt.figure(figsize=(20,10))
plt.scatter(x,y)
plt.plot(x,y_stochastic_gradient_descent,color='purple')
plt.xlabel('x', fontsize=30)
plt.ylabel('y', fontsize=30)
plt.yticks(fontsize =27)
plt.xticks(fontsize =25)
plt.show()
```



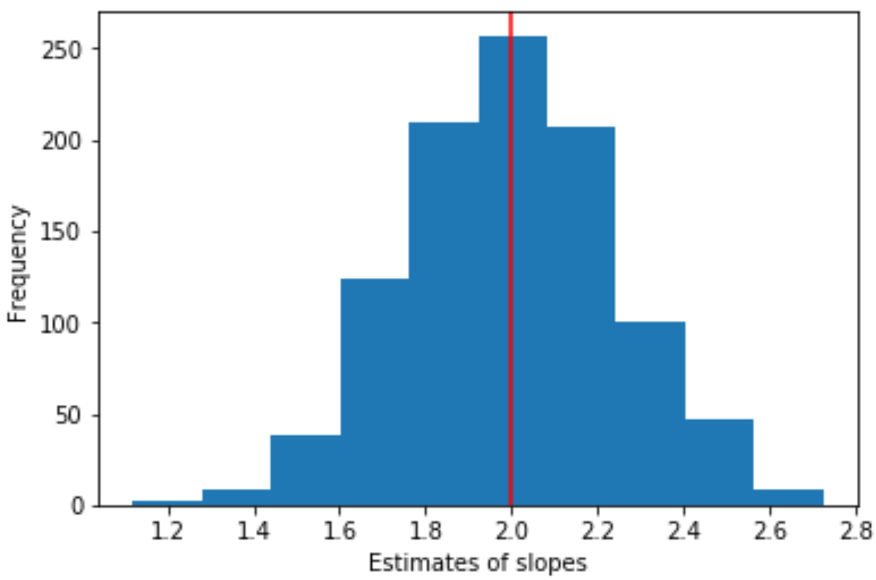
In [179]:

```
#question 5c) iii Huber loss case 2
matrix_x = np.c_[np.ones(x.shape[0]), x]
theta_i = np.random.rand(2)
all_thetas2, loss, predictions = gradient_descent(matrix_x, y, theta_i, loss_fun
c_d)

theta_ciii2 = all_thetas2[-1]

y_stochastic_gradient_descent=np.dot(matrix_x,theta_ciii2)

fig=plt.figure(figsize=(20,10))
plt.scatter(x,y)
plt.plot(x,y_stochastic_gradient_descent,color='brown')
plt.xlabel('x', fontsize=30)
plt.ylabel('y', fontsize=30)
plt.yticks(fontsize =27)
plt.xticks(fontsize =25)
plt.show()
```



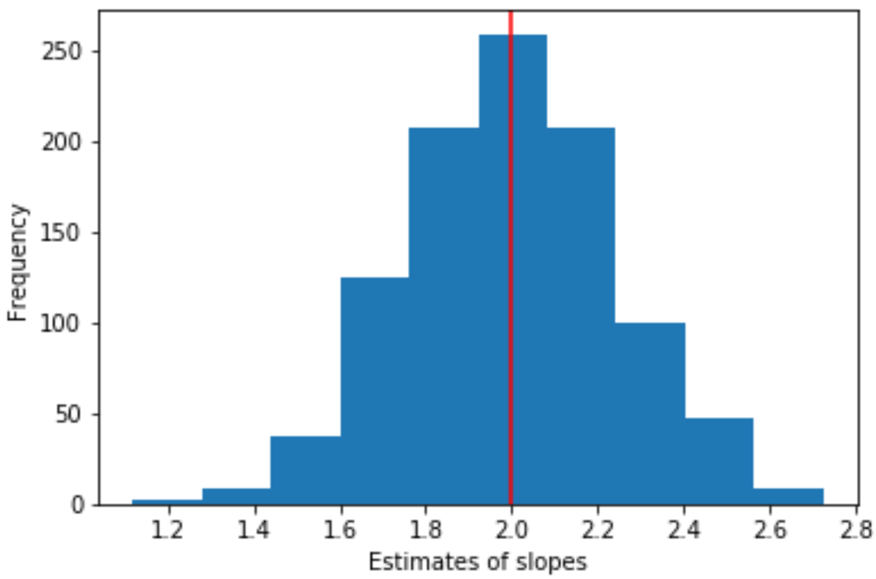
In [147]:

```
#question 5di) — repeating steps in part ci) 1000 times for analytical solution
theta_a = []
slopes_a = []
x = np.zeros((50,1000))
y = np.zeros((50,1000))
for d in range(1000):
    x[:,d] = np.random.uniform(low=-2, high=2, size=(50,))
    y[:,d] = 3+2*x[:,d]+np.random.normal(0,2,50)
    matrix_x = np.c_[np.ones(x[:,d].shape[0]), x[:,d]]
    theta_a.append(np.dot(inv(np.dot(matrix_x.T,matrix_x)),np.dot(matrix_x.T,y[:
,d]))) #This is the analytical solution of theta
    slopes_a.append(theta_a[d-1][1])


plt.hist(slopes_a)
plt.xlabel('Estimates of slopes')
plt.ylabel('Frequency')
plt.axvline(x=2.0, color='red')
plt.show()
```



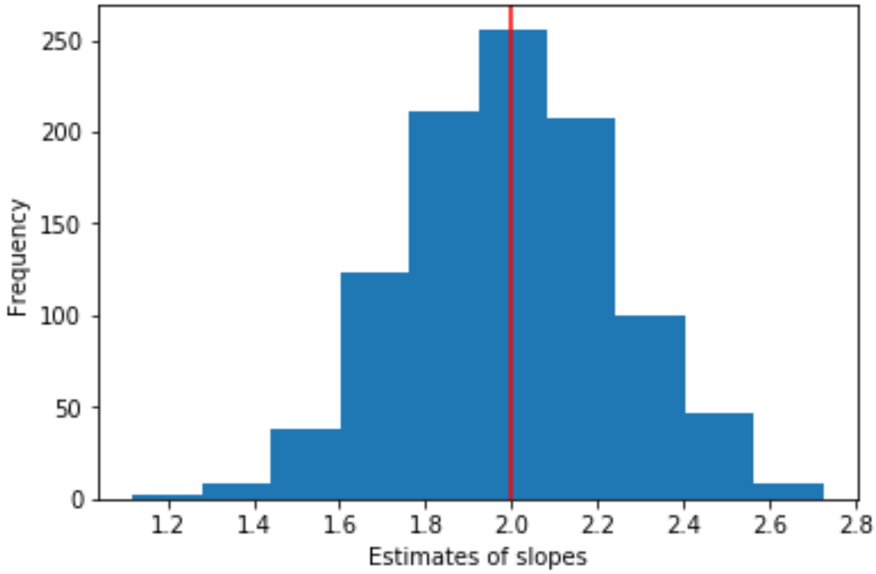
In [150]:

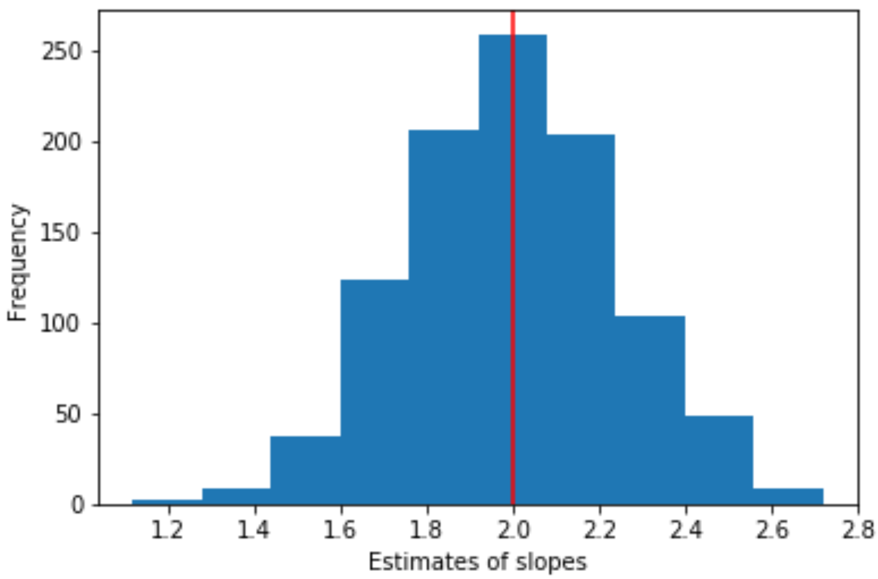
```
#question 5dii) — repeating steps in part cii) 1000 times with batch gradient de
scent
theta_b = []
slopes_b = []
for d in range(1000):
    matrix_x = np.c_[np.ones(x[:,d].shape[0]), x[:,d]]
    theta_i = np.random.rand(2)
    all_thetas2, loss, predictions = gradient_descent(matrix_x, y[:,d], theta_i,
loss_func_b)
    theta_b.append(all_thetas2[-1])
    slopes_b.append(theta_b[d-1][1])

plt.hist(slopes_b)
plt.xlabel('Estimates of slopes')
plt.ylabel('Frequency')
plt.axvline(x=2.0, color='red')
plt.show()
```



In [151]:

```
#question 5diii) – repeating steps in part ciii) 1000 times with batch gradient descent
theta_c1 = []
slopes_c1 = []
for d in range(1000):
    matrix_x = np.c_[np.ones(x[:,d].shape[0]), x[:,d]]
    all_thetas2, loss, predictions = gradient_descent(matrix_x, y[:,d], theta_i, loss_func_c)
    theta_c1.append(all_thetas2[-1])
    slopes_c1.append(theta_c1[d-1][1])

plt.hist(slopes_c1)
plt.xlabel('Estimates of slopes')
plt.ylabel('Frequency')
plt.axvline(x=2.0, color='red')
plt.show()

theta_c2 = []
slopes_c2 = []
for d in range(1000):
    matrix_x = np.c_[np.ones(x[:,d].shape[0]), x[:,d]]
    all_thetas2, loss, predictions = gradient_descent(matrix_x, y[:,d], theta_i, loss_func_d)
    theta_c2.append(all_thetas2[-1])
    slopes_c2.append(theta_c2[d-1][1])

plt.hist(slopes_c2)
plt.xlabel('Estimates of slopes')
plt.ylabel('Frequency')
plt.axvline(x=2.0, color='red')
plt.show()
```





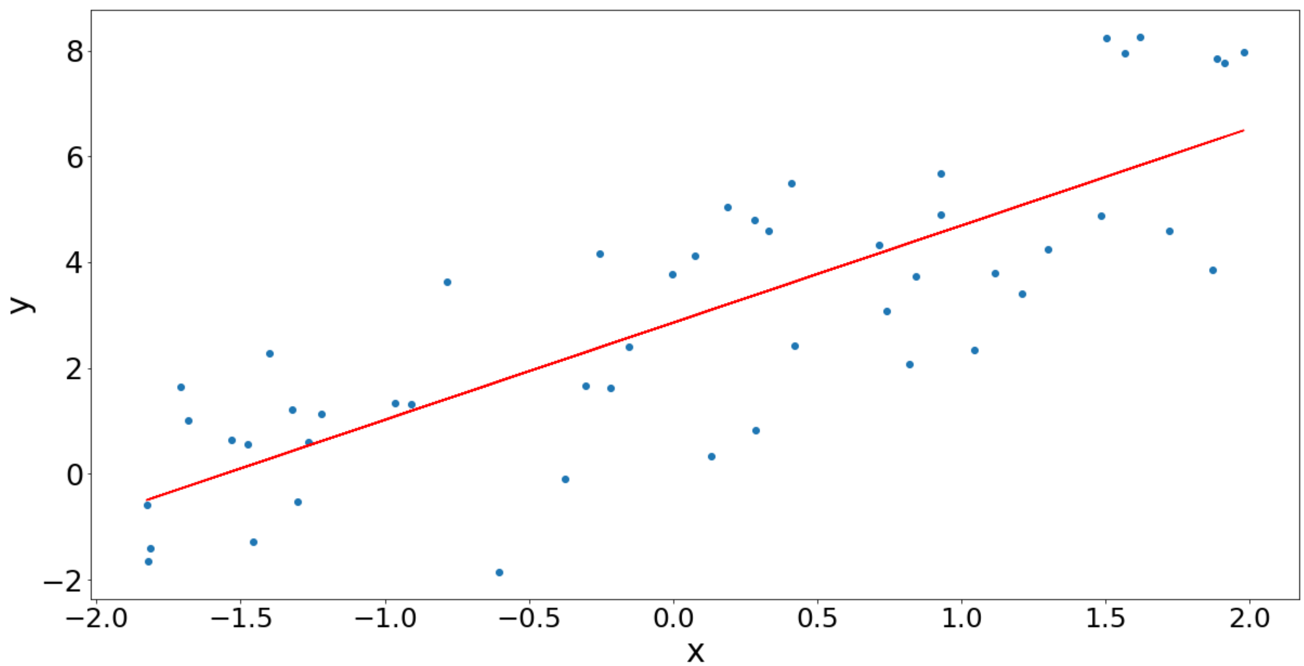
The choice of the loss function in this case does not have a substantial effect on the estimates of the slope parameter.

In [153]:

```
#question 5e)
x=np.random.uniform(low=-2, high=2, size=(50,))
y=3+2*x+np.random.normal(0,2,50)

mask = np.random.rand(50)
for i in range(len(y[mask > .9])):
    if np.random.rand(1) > 0.5:
        y[mask > 0.9][i] = y[mask > 0.9][i]*2
    else:
        y[mask > 0.9][i] = y[mask > 0.9][i]/2
```

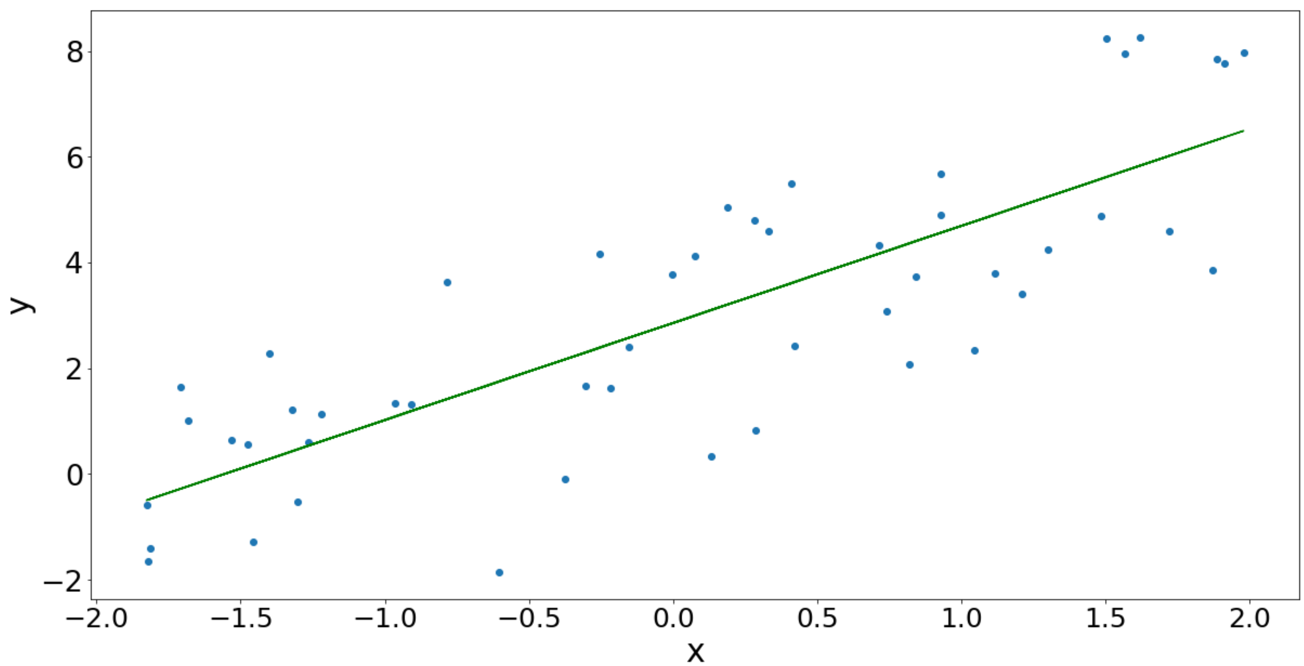
In [181]:

```
#5e) i) squared loss with analytical solution
matrix_x = np.c_[np.ones(x.shape[0]), x]
theta_ei = np.dot(inv(np.dot(matrix_x.T,matrix_x)),np.dot(matrix_x.T,y)) #This i
s the analytical solution of theta

y_analytical_solution=np.dot(matrix_x,theta_ei)

fig=plt.figure(figsize=(20,10))
plt.scatter(x,y)
plt.plot(x,y_analytical_solution,color='red')
plt.xlabel('x', fontsize=30)
plt.ylabel('y', fontsize=30)
plt.yticks(fontsize =27)
plt.xticks(fontsize =25)
plt.show()
```



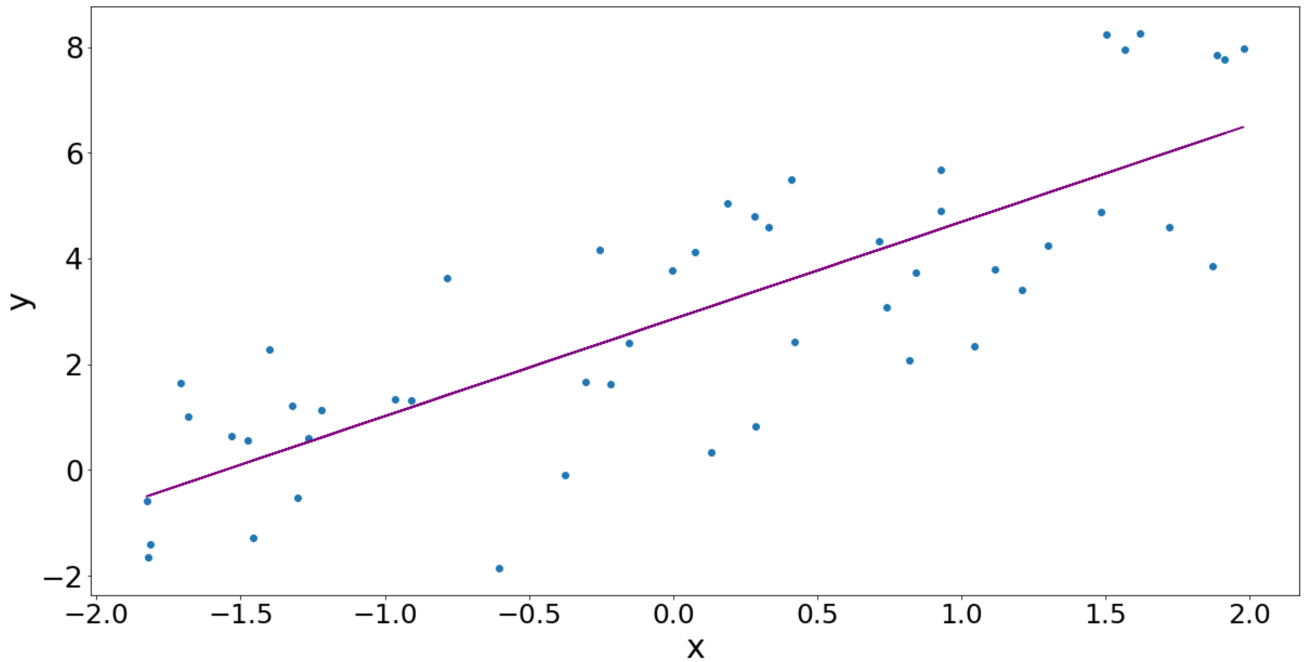
In [182]:

```
#5e) ii) mean absolute error with batch gradient descent
matrix_x = np.c_[np.ones(x.shape[0]), x]
theta_i = np.random.rand(2)
all_thetas1, loss, predictions = gradient_descent(matrix_x, y, theta_i, loss_fun
c_b)

theta_eii = all_thetas1[-1]

y_gradient_descent=np.dot(matrix_x,theta_eii)

fig=plt.figure(figsize=(20,10))
plt.scatter(x,y)
plt.plot(x,y_gradient_descent,color='green')
plt.xlabel('x', fontsize=30)
plt.ylabel('y', fontsize=30)
plt.yticks(fontsize =27)
plt.xticks(fontsize =25)
plt.show()
```

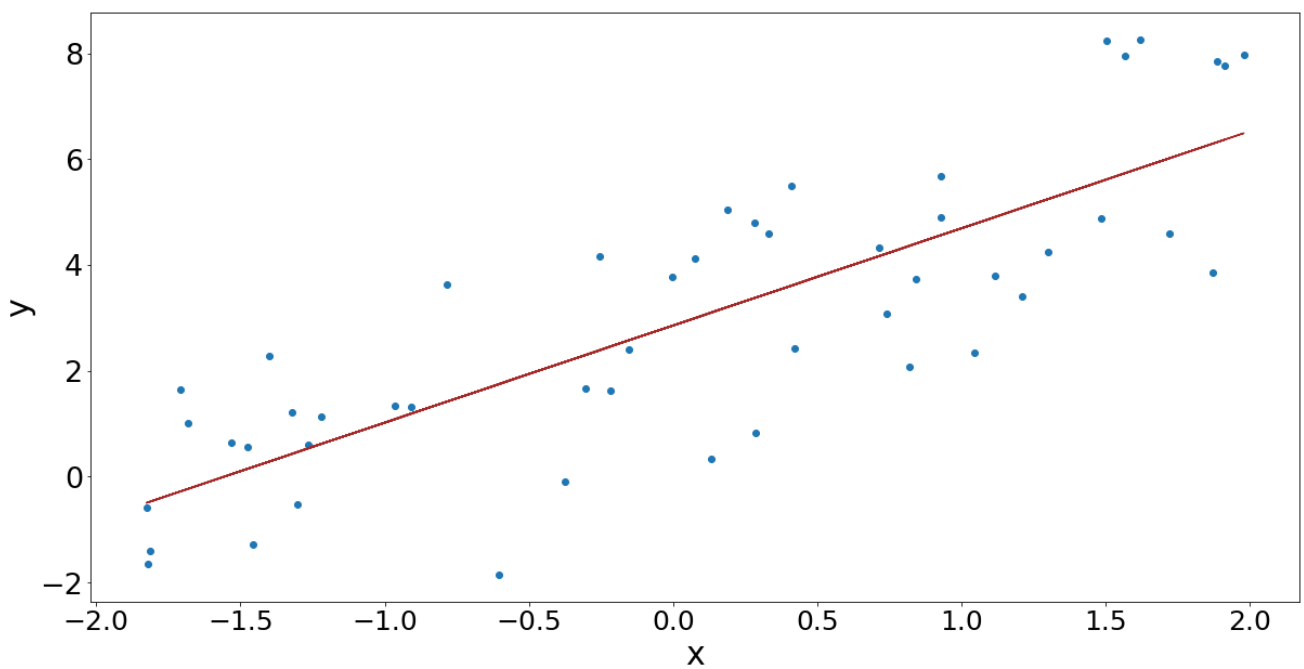


In [183]:

```
#5e) Huber loss with batch gradient descent
matrix_x = np.c_[np.ones(x.shape[0]), x]
theta_i = np.random.rand(2)
all_thetas2, loss, predictions = gradient_descent(matrix_x, y, theta_i, loss_fun
c_c)

theta_eiii = all_thetas2[-1]

y_stochastic_gradient_descent=np.dot(matrix_x,theta_eiii)

fig=plt.figure(figsize=(20,10))
plt.scatter(x,y)
plt.plot(x,y_stochastic_gradient_descent,color='purple')
plt.xlabel('x', fontsize=30)
plt.ylabel('y', fontsize=30)
plt.yticks(fontsize =27)
plt.xticks(fontsize =25)
plt.show()

matrix_x = np.c_[np.ones(x.shape[0]), x]
theta_i = np.random.rand(2)
all_thetas2, loss, predictions = gradient_descent(matrix_x, y, theta_i, loss_fun
c_d)

theta_eiii2 = all_thetas2[-1]

y_stochastic_gradient_descent=np.dot(matrix_x,theta_eiii2)

fig=plt.figure(figsize=(20,10))
plt.scatter(x,y)
plt.plot(x,y_stochastic_gradient_descent,color='brown')
plt.xlabel('x', fontsize=30)
plt.ylabel('y', fontsize=30)
plt.yticks(fontsize =27)
plt.xticks(fontsize =25)
plt.show()
```



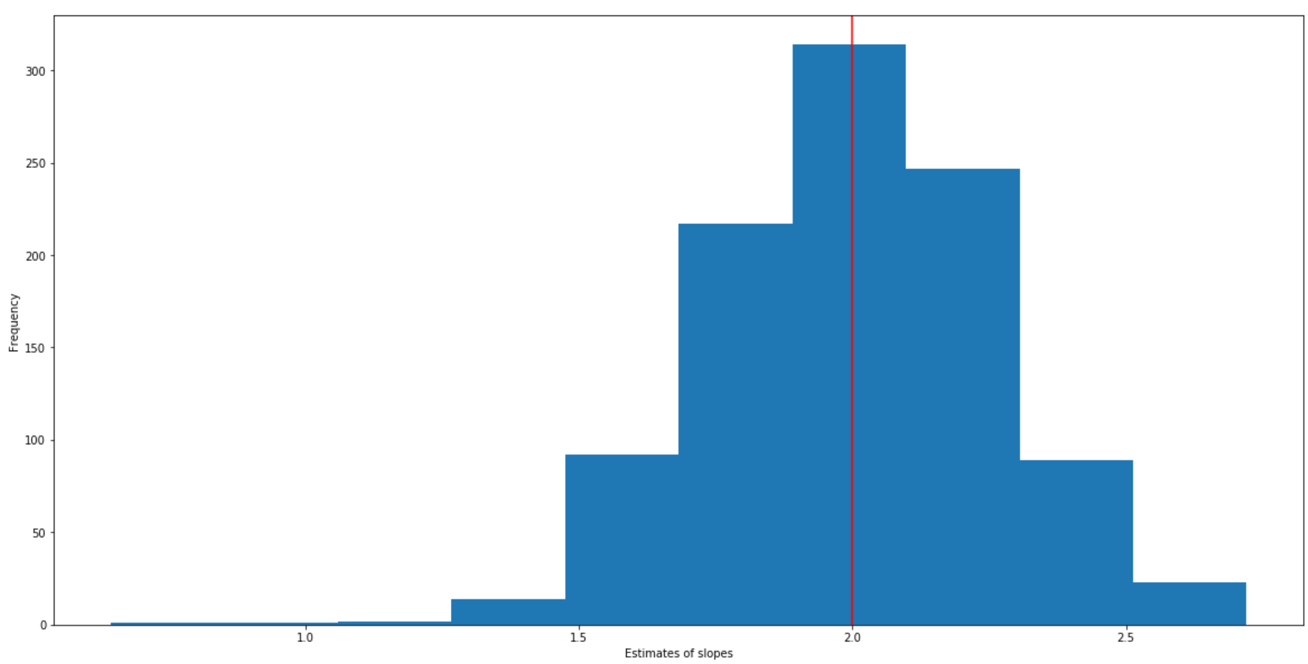
8
6
4
2
0
−2
y
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
x



8
6
4
2
0
−2
y
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
x

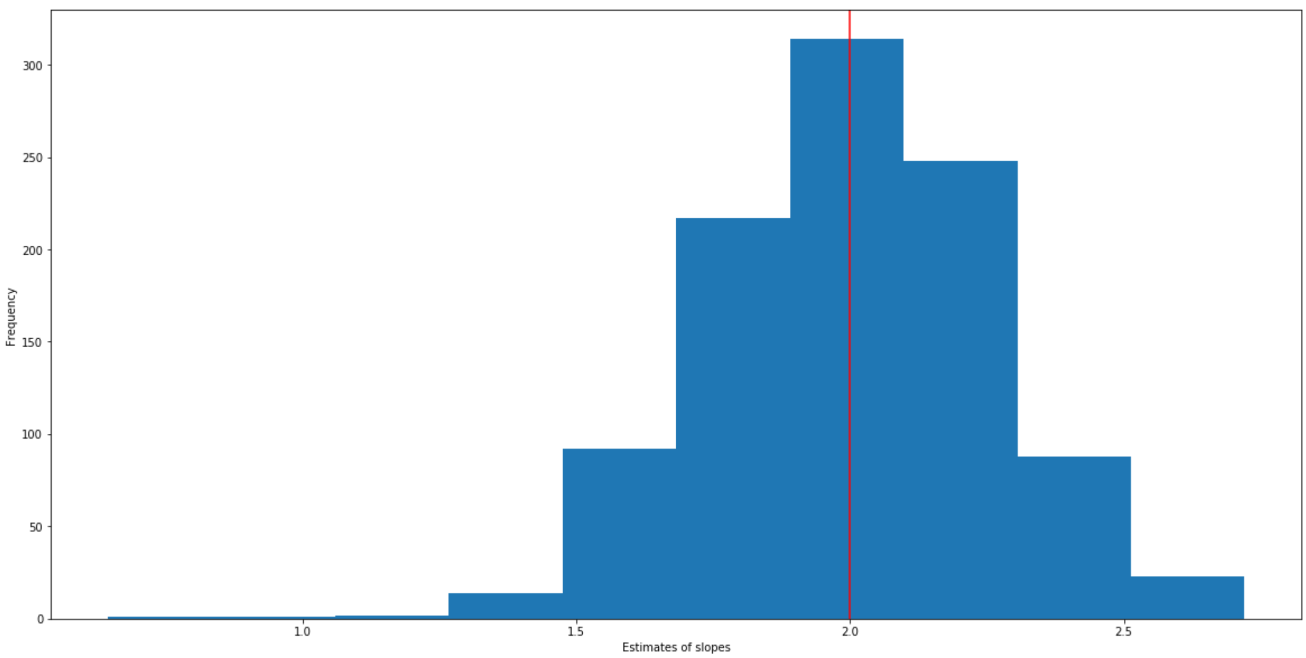
In [191]:

```
#5f) i) repeat e) 1000 times for squared loss with analytical solution
theta_a = []
slopes_a = []
x = np.zeros((50,1000))
y = np.zeros((50,1000))
for d in range(1000):
    x[:,d] = np.random.uniform(low=-2, high=2, size=(50,))
    y[:,d] = 3+2*x[:,d]+np.random.normal(0,2,50)
    mask = np.random.rand(50)
    for i in range(len(y[mask > .9])):
        if np.random.rand(1) > 0.5:
            y[mask > 0.9][i] = y[mask > 0.9][i]*2
        else:
            y[mask > 0.9][i] = y[mask > 0.9][i]/2
    matrix_x = np.c_[np.ones(x[:,d].shape[0]), x[:,d]]
    theta_a.append(np.dot(inv(np.dot(matrix_x.T,matrix_x)),np.dot(matrix_x.T,y[:
,d]))) #This is the analytical solution of theta
    slopes_a.append(theta_a[d-1][1])

fig=plt.figure(figsize=(20,10))
plt.hist(slopes_a)
plt.xlabel('Estimates of slopes')
plt.ylabel('Frequency')
plt.axvline(x=2.0, color='red')
plt.show()
```



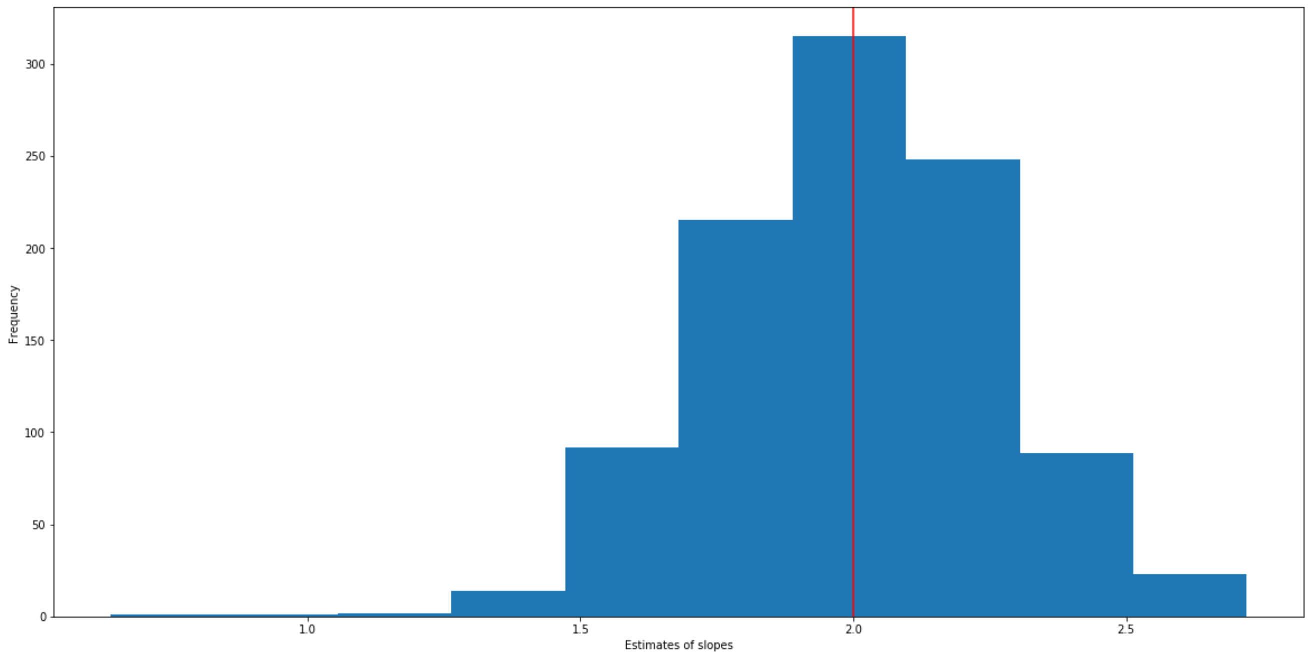
In [192]:

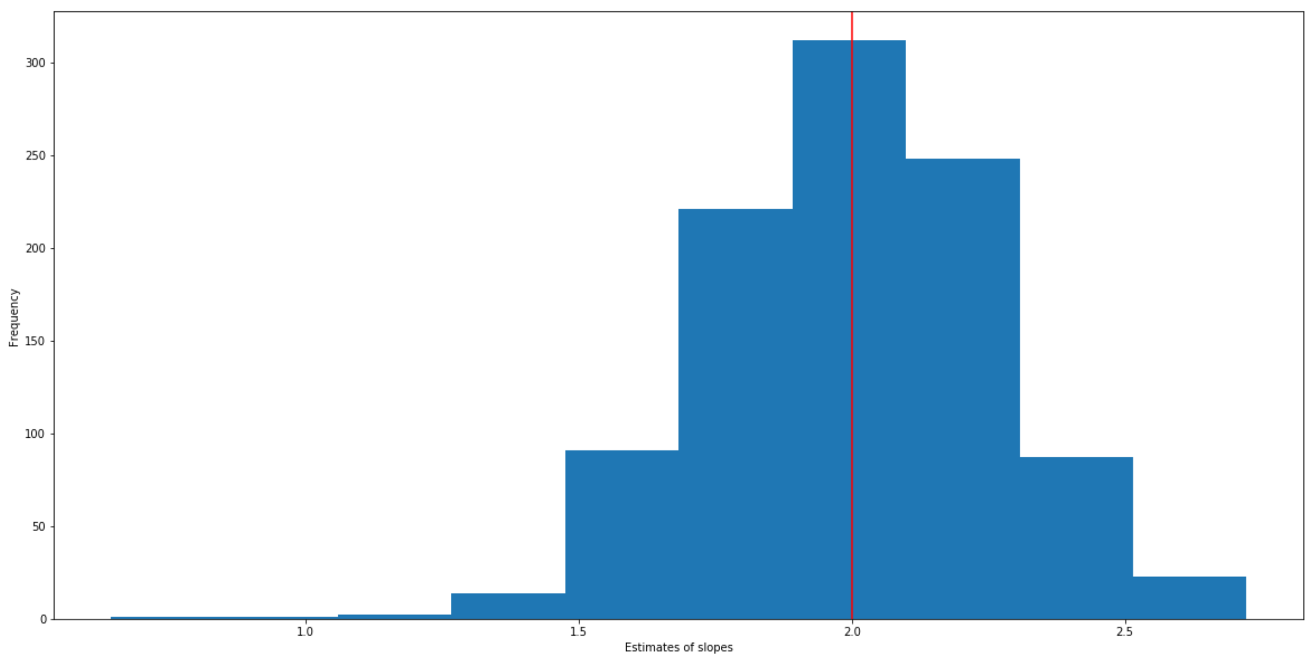
```
#5f) ii) repeat e 1000 times for mean absolute error with batch gradient descent
theta_b = []
slopes_b = []
for d in range(1000):
    matrix_x = np.c_[np.ones(x[:,d].shape[0]), x[:,d]]
    theta_i = np.random.rand(2)
    all_thetas2, loss, predictions = gradient_descent(matrix_x, y[:,d], theta_i,
loss_func_b)
    theta_b.append(all_thetas2[-1])
    slopes_b.append(theta_b[d-1][1])

fig=plt.figure(figsize=(20,10))
plt.hist(slopes_b)
plt.xlabel('Estimates of slopes')
plt.ylabel('Frequency')
plt.axvline(x=2.0, color='red')
plt.show()
```



In [193]:

```
#question 5fiii) — repeating steps in part e 1000 times for Huber loss with batc
h gradient descent
theta_c1 = []
slopes_c1 = []
for d in range(1000):
    matrix_x = np.c_[np.ones(x[:,d].shape[0]), x[:,d]]
    all_thetas2, loss, predictions = gradient_descent(matrix_x, y[:,d], theta_i,
loss_func_c)
    theta_c1.append(all_thetas2[-1])
    slopes_c1.append(theta_c1[d-1][1])

fig=plt.figure(figsize=(20,10))
plt.hist(slopes_c1)
plt.xlabel('Estimates of slopes')
plt.ylabel('Frequency')
plt.axvline(x=2.0, color='red')
plt.show()

theta_c2 = []
slopes_c2 = []
for d in range(1000):
    matrix_x = np.c_[np.ones(x[:,d].shape[0]), x[:,d]]
    all_thetas2, loss, predictions = gradient_descent(matrix_x, y[:,d], theta_i,
loss_func_d)
    theta_c2.append(all_thetas2[-1])
    slopes_c2.append(theta_c2[d-1][1])

fig=plt.figure(figsize=(20,10))
plt.hist(slopes_c2)
plt.xlabel('Estimates of slopes')
plt.ylabel('Frequency')
plt.axvline(x=2.0, color='red')
plt.show()
```





The choice of the loss function in this case does not have a substantial effect on the estimates of the slope parameter.