# Homework 2 - Berkeley STAT 157

Handout 1/29/2019, due 2/5/2019 by 4pm in Git by committing to your repository.

In [1]:
```
from mxnet import nd, autograd, gluon
from matplotlib import pyplot as plt
import numpy as np
```

# 1. Multinomial Sampling

Implement a sampler from a discrete distribution from scratch, mimicking the function `mxnet.ndarray.random.multinomial`. Its arguments should be a vector of probabilities $p$. You can assume that the probabilities are normalized, i.e. tha they sum up to $1$. Make the call signature as follows:

```
samples = sampler(probs, shape)

probs   : An ndarray vector of size n of nonnegative numbers summing up to 1
shape   : A list of dimensions for the output
samples : Samples from probs with shape matching shape
```

Hints:

1. Use `mxnet.ndarray.random.uniform` to get a sample from $U[0, 1]$.
2. You can simplify things for `probs` by computing the cumulative sum over `probs`.

In [4]:
```
def sampler(probs, shape):
    ## Add your codes here
    samples = nd.zeros(shape).reshape(-1,)
    num_items = shape[0] * shape[1]
    for i in range(num_items):
        ran_gen = nd.random.uniform()
        counter = 0
        while ran_gen > 0:
            ran_gen -= probs[counter]
            if (ran_gen) <= 0:
                break
            counter += 1
        samples[i] = probs[counter]
    samples = samples.reshape(shape)
    return samples

# a simple test
sampler(nd.array([0.2, 0.3, 0.5]), (2,3))
```

Out[4]:
```
[[0.3 0.3 0.5]
 [0.2 0.5 0.3]]
<NDArray 2x3 @cpu(0)>
```

# 2. Central Limit Theorem

Let's explore the Central Limit Theorem when applied to text processing.

- Download https://www.gutenberg.org/ebooks/84 (https://www.gutenberg.org/files/84/84-0.txt) from Project Gutenberg
- Remove punctuation, uppercase / lowercase, and split the text up into individual tokens (words).
- For the words `a`, `and`, `the`, `i`, `is` compute their respective counts as the book progresses, i.e.

$$n_{\text{the}}[i] = \sum_{j=1}^{i} \{w_j = \text{the}\}$$

- Plot the proportions $n_{\text{word}}[i]/i$ over the document in one plot.
- Find an envelope of the shape $O(1/\sqrt{i})$ for each of these five words. (Hint, check the last page of the sampling notebook (http://courses.d2l.ai/berkeley-stat-157/slides/1_24/sampling.pdf))
- Why can we **not** apply the Central Limit Theorem directly?
- How would we have to change the text for it to apply?

- Why does it still work quite well?

**2.1 Answer) Download [https://www.gutenberg.org/ebooks/84 (https://www.gutenberg.org/files/84/84-0.txt)](https://www.gutenberg.org/files/84/84-0.txt) from Project Gutenberg**

- Remove punctuation, uppercase / lowercase, and split the text up into individual tokens (words).
- For the words `a`, `and`, `the`, `i`, `is` compute their respective counts as the book progresses, i.e.

$$n_{\mathrm{the}}[i] = \sum_{j=1}^{i} \{w_j = \mathrm{the}\}$$

- Plot the proportions $n_{\mathrm{word}}[i]/i$ over the document in one plot.

In [20]:
```
filename = gluon.utils.download('https://www.gutenberg.org/files/84/84-0.txt')
with open(filename) as f:
    book = f.read()

def preprocess(text):
    punctuation = set([',', '.', '-', '(', ')', ';', '/', '*', "'", ' ', ':', '"', '['])
    nums = set('0123456789$')

    return [word.lower() for word in text.split() if not any((x in word) for x in punctuation) and not a

lst = preprocess(book)

a_counts = np.zeros(len(lst))
and_counts = np.zeros(len(lst))
the_counts = np.zeros(len(lst))
i_counts = np.zeros(len(lst))
is_counts = np.zeros(len(lst))

for i in range(0,len(lst)):
    if (i != 0):
        a_counts[i] = a_counts[i-1]
        and_counts[i] = and_counts[i-1]
        the_counts[i] = the_counts[i-1]
        i_counts[i] = i_counts[i-1]
        is_counts[i] = is_counts[i-1]
        if (lst[i] == 'a'):
            a_counts[i] += 1
        elif (lst[i] == 'and'):
            and_counts[i] += 1
        elif (lst[i] == 'the'):
            the_counts[i] += 1
        elif (lst[i] == 'i'):
            i_counts[i] += 1
        elif (lst[i] == 'is'):
            is_counts[i] += 1
    else:
        if (lst[0] == 'a'):
            a_counts[0] = 1
        elif (lst[0] == 'and'):
            and_counts[0] = 1
        elif (lst[0] == 'the'):
            the_counts[0] = 1
```

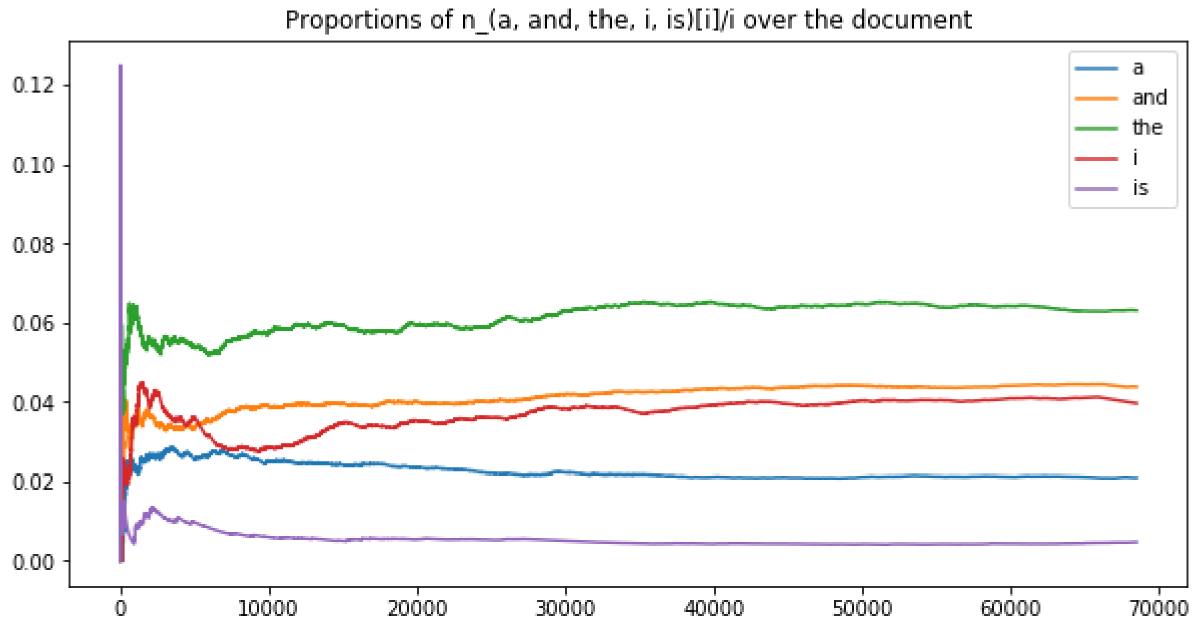
```
        elif (lst[0] == 'i'):
            i_counts[0] = 1
        elif (lst[0] == 'is'):
            is_counts[0] = 1

a_prop = np.zeros(len(lst))
and_prop = np.zeros(len(lst))
the_prop = np.zeros(len(lst))
i_prop = np.zeros(len(lst))
is_prop = np.zeros(len(lst))

for i in range(1,len(lst)):
    a_prop[i] = a_counts[i]/i
    and_prop[i] = and_counts[i]/i
    the_prop[i] = the_counts[i]/i
    i_prop[i] = i_counts[i]/i
    is_prop[i] = is_counts[i]/i
```

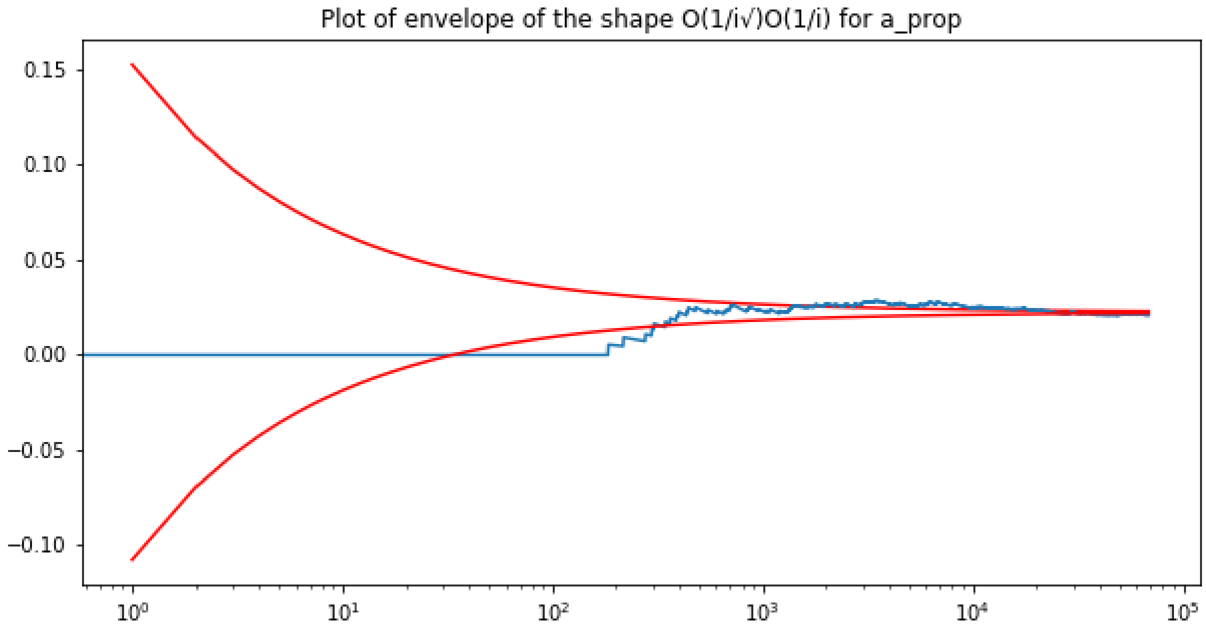
**2.2 Answer) Plot the proportions $n_{\text{word}}[i]/i$ over the document in one plot.**

In [147]:
```
plt.figure(figsize=(10,5))
plt.plot(a_prop, label ='a')
plt.plot(and_prop, label ='and')
plt.plot(the_prop, label ='the')
plt.plot(i_prop, label ='i')
plt.plot(is_prop, label ='is')
plt.title('Proportions of n_(a, and, the, i, is)[i]/i over the document')
plt.legend()
plt.show()
```



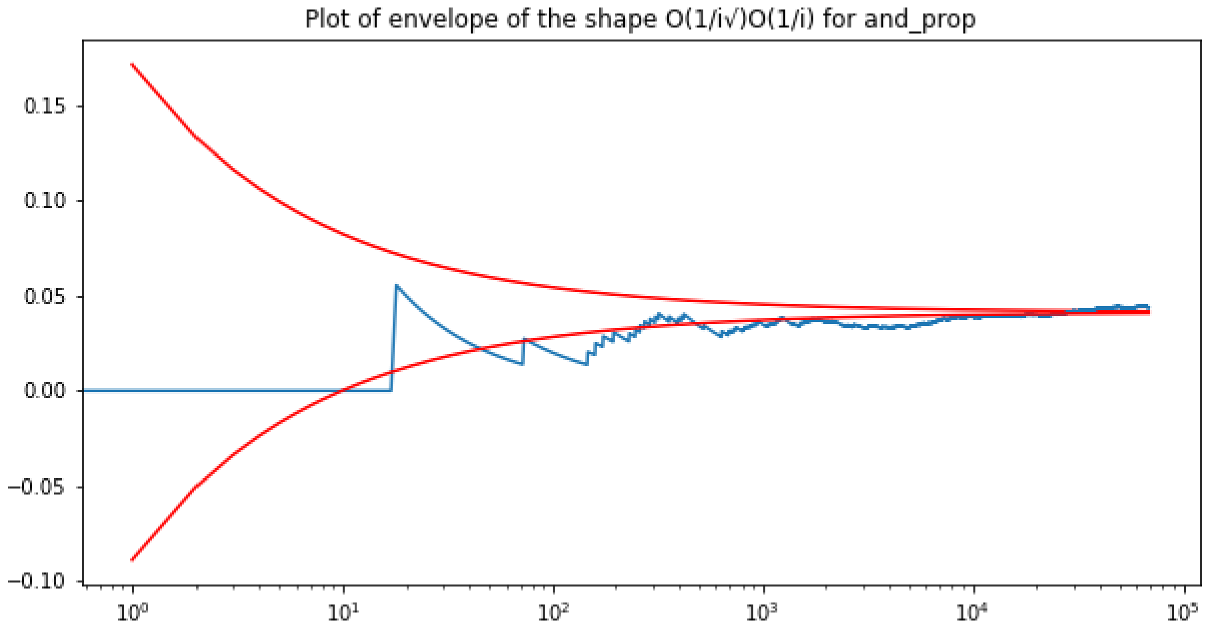

**2.3. Answer) Plot of envelope of the shape O(1/i√)O(1/i) for a_prop.**

In [148]:
```
y = np.arange(1,len(lst)+1).reshape(len(lst),1)
plt.figure(figsize=(10,5))
plt.semilogx(y, 0.13*np.power(y,-0.5) + np.mean(a_prop), 'r')
plt.plot(a_prop)
plt.semilogx(y, -0.13*np.power(y,-0.5) + np.mean(a_prop), 'r')
plt.title('Plot of envelope of the shape O(1/i√)O(1/i) for a_prop')
plt.show()
```



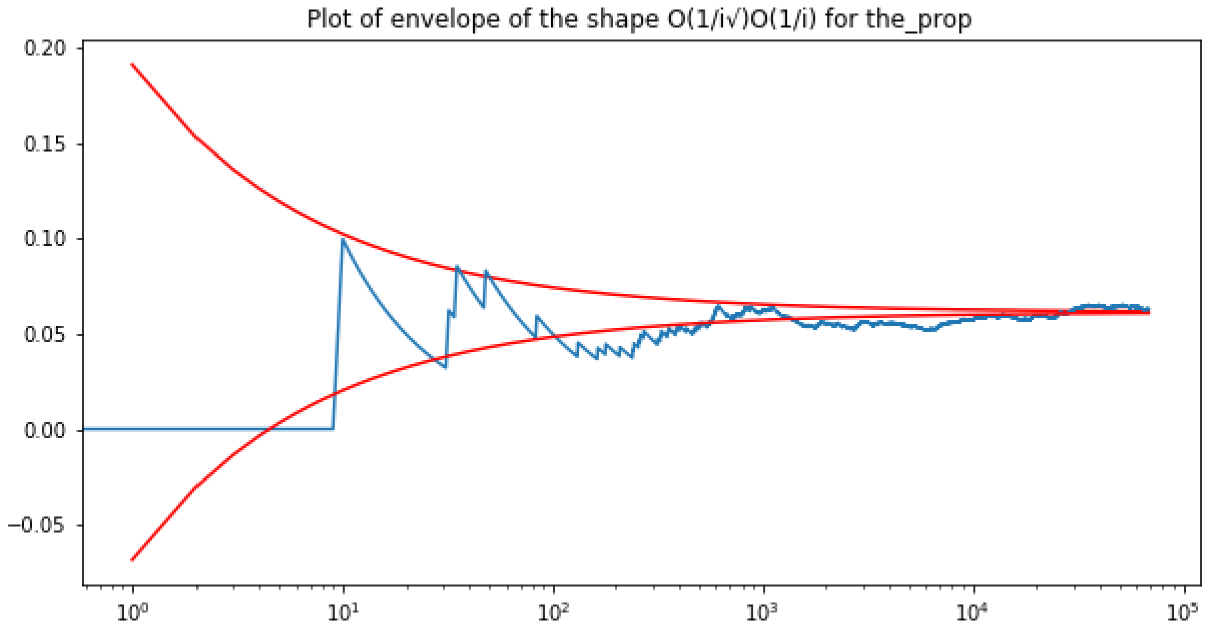

**2.3. Answer) Plot of envelope of the shape O(1/i√)O(1/i) for and_prop.**

In [149]:
```
y = np.arange(1,len(lst)+1).reshape(len(lst),1)
plt.figure(figsize=(10,5))
plt.semilogx(y, 0.13*np.power(y,-0.5) + np.mean(and_prop), 'r')
plt.plot(and_prop)
plt.semilogx(y, -0.13*np.power(y,-0.5) + np.mean(and_prop), 'r')
plt.title('Plot of envelope of the shape O(1/i√)O(1/i) for and_prop')
plt.show()
```



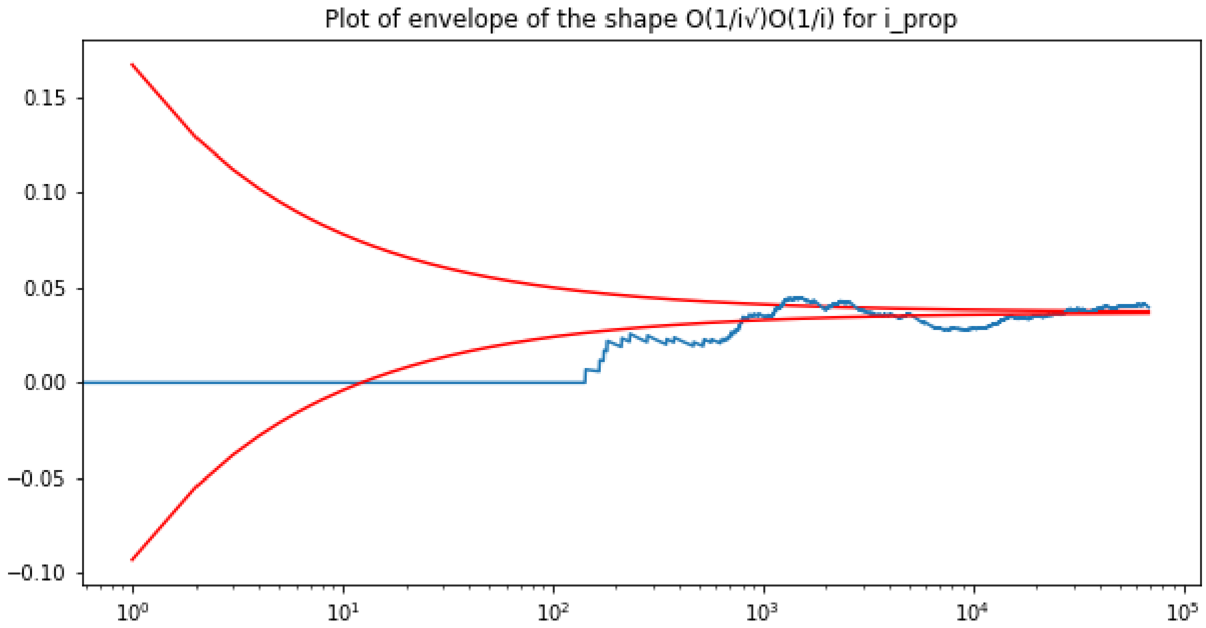

**2.3. Answer) Plot of envelope of the shape O(1/i√)O(1/i) for the_prop.**

In [150]:
```
y = np.arange(1,len(lst)+1).reshape(len(lst),1)
plt.figure(figsize=(10,5))
plt.semilogx(y, 0.13*np.power(y,-0.5) + np.mean(the_prop), 'r')
plt.plot(the_prop)
plt.semilogx(y, -0.13*np.power(y,-0.5) + np.mean(the_prop), 'r')
plt.title('Plot of envelope of the shape O(1/i√)O(1/i) for the_prop')
plt.show()
```



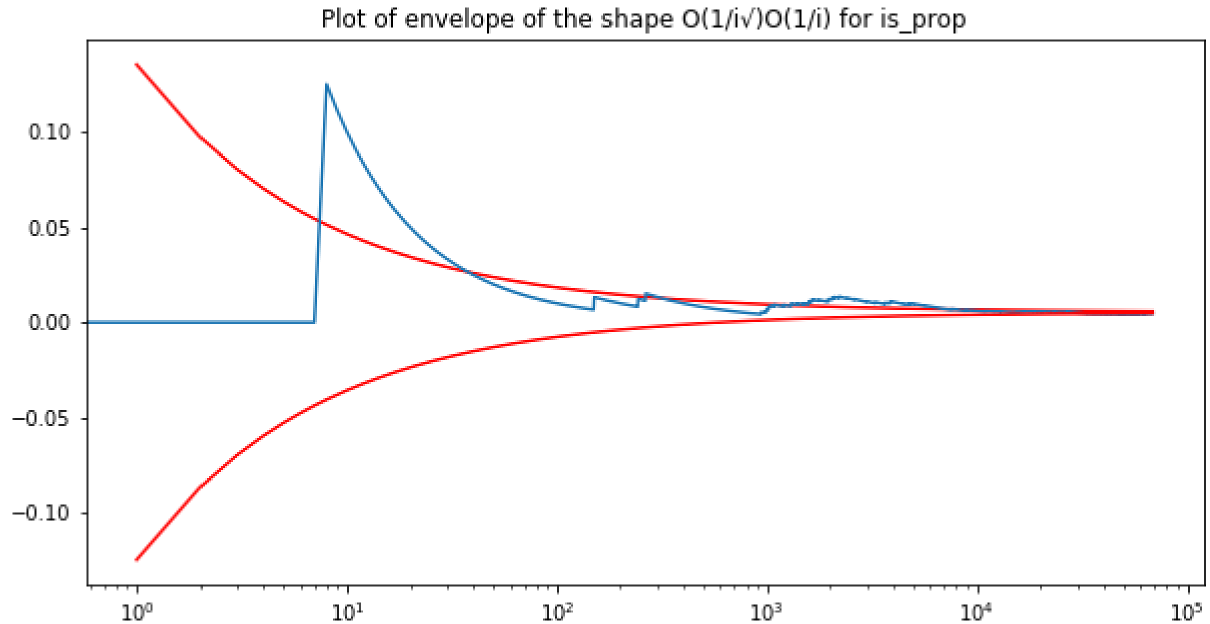

**2.3. Answer) Plot of envelope of the shape O(1/i√)O(1/i) for i_prop.**

In [151]:
```
y = np.arange(1,len(lst)+1).reshape(len(lst),1)
plt.figure(figsize=(10,5))
plt.semilogx(y, 0.13*np.power(y,-0.5) + np.mean(i_prop), 'r')
plt.plot(i_prop)
plt.semilogx(y, -0.13*np.power(y,-0.5) + np.mean(i_prop), 'r')
plt.title('Plot of envelope of the shape O(1/i√)O(1/i) for i_prop')
plt.show()
```




**2.3. Answer) Plot of envelope of the shape O(1/i√)O(1/i) for is_prop.**

In [152]:
```
y = np.arange(1,len(lst)+1).reshape(len(lst),1)
plt.figure(figsize=(10,5))
plt.semilogx(y, 0.13*np.power(y,-0.5) + np.mean(is_prop), 'r')
plt.plot(is_prop)
plt.semilogx(y, -0.13*np.power(y,-0.5) + np.mean(is_prop), 'r')
plt.title('Plot of envelope of the shape O(1/i√)O(1/i) for is_prop')
plt.show()
```




**Answer) Why can we not apply the Central Limit Theorem directly?**

We can't apply the Central Limit Theorem directly because the sequence in which words appear in a document is not independent of one another.

**Answer) How would we have to change the text for it to apply?**

If we shuffle the texts, then we can apply the CLT.

**Answer) Why does it still work quite well?**

It still works well because sample size (# of words) is large and our bag of words are relatively small so dependency isn't magnified.

## 3. Denominator-layout notation

We used the numerator-layout notation for matrix calculus in class, now let's examine the denominator-layout notation.

Given $x, y \in \mathbb{R}$, $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{y} \in \mathbb{R}^m$, we have

$$\frac{\partial y}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y}{\partial x_1} \\ \frac{\partial y}{\partial x_2} \\ \vdots \\ \frac{\partial y}{\partial x_n} \end{bmatrix}, \qquad \frac{\partial \mathbf{y}}{\partial x} = \begin{bmatrix} \frac{\partial y_1}{\partial x}, \frac{\partial y_2}{\partial x}, \ldots, \frac{\partial y_m}{\partial x} \end{bmatrix}$$

and

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial \mathbf{y}}{\partial x_1} \\ \frac{\partial \mathbf{y}}{\partial x_2} \\ \vdots \\ \frac{\partial \mathbf{y}}{\partial x_3} \end{bmatrix} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1}, \frac{\partial y_2}{\partial x_1}, \ldots, \frac{\partial y_m}{\partial x_1} \\ \frac{\partial y_1}{\partial x_2}, \frac{\partial y_2}{\partial x_2}, \ldots, \frac{\partial y_m}{\partial x_2} \\ \vdots \\ \frac{\partial y_1}{\partial x_n}, \frac{\partial y_2}{\partial x_n}, \ldots, \frac{\partial y_m}{\partial x_n} \end{bmatrix}$$

Questions:

1. Assume $\mathbf{y} = f(\mathbf{u})$ and $\mathbf{u} = g(\mathbf{x})$, write down the chain rule for $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$
2. Given $\mathbf{X} \in \mathbb{R}^{m \times n},\ \mathbf{w} \in \mathbb{R}^n,\ \mathbf{y} \in \mathbb{R}^m$, assume $z = ||\mathbf{X}\mathbf{w} - \mathbf{y}||^2$, compute $\frac{\partial z}{\partial \mathbf{w}}$.

**Answers:**

$1. \frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \frac{\partial \mathbf{u}}{\partial \mathbf{x}} \frac{\partial \mathbf{y}}{\partial \mathbf{u}}$ (performed in this order for matrix dimensions to match denominator layout notation.)

2. First expand z. $z = \|\mathbf{Xw} - \mathbf{y}\|^2, = (Xw - y)^T(Xw - y)$

$$
\begin{aligned}
&= (w^T X^T - y^T)(Xw - y) \\
&= w^T XTXw - w^T XTy - y^T Xw + y^T y \\
&= w^T XTXw - 2y^T Xw + y^T y \\
&Now \frac{\partial z}{\partial \mathbf{w}}(w^T X^T Xw - 2y^T Xw + y^T y) = 2w^T (X^T X) - 2y^T X
\end{aligned}
$$

# 4. Numerical Precision

Given scalars `x` and `y`, implement the following `log_exp` function such that it returns

$$
-\log\left(\frac{e^x}{e^x + e^y}\right)
$$

.

In [145]:

```
def log_exp(x, y):
    ## add your solution here
    return -nd.log(nd.exp(x) / (nd.exp(x) + nd.exp(y)))
```

Test your codes with normal inputs:

In [146]:

```
x, y = nd.array([2]), nd.array([3])
z = log_exp(x, y)
z
```

Out[146]:

```
[1.3132617]
<NDArray 1 @cpu(0)>
```

Now implement a function to compute $\partial z/\partial x$ and $\partial z/\partial y$ with `autograd`

In [138]:
```
def grad(forward_func, x, y):
    ## Add your codes here
    x.attach_grad()
    y.attach_grad()
    with autograd.record():
        z = forward_func(x,y)
    z.backward()
```

Test your codes, it should print the results nicely.

In [139]:
```
grad(log_exp, x, y)
print('x.grad =', x.grad)
print('y.grad =', y.grad)
```

```
x.grad =
[-0.7310586]
<NDArray 1 @cpu(0)>
y.grad =
[0.7310586]
<NDArray 1 @cpu(0)>
```

But now let's try some "hard" inputs

In [140]:
```
x, y = nd.array([50]), nd.array([100])
grad(log_exp, x, y)
print('x.grad =', x.grad)
print('y.grad =', y.grad)
```

```
x.grad =
[nan]
<NDArray 1 @cpu(0)>
y.grad =
[nan]
<NDArray 1 @cpu(0)>
```

Does your code return correct results? If not, try to understand the reason. (Hint, evaluate `exp(100)`). Now develop a new function `stable_log_exp` that is identical to `log_exp` in math, but returns a more numerical stable result.

In [143]:
```
def stable_log_exp(x, y):
    if x > y:
        a = x
        b = y
    else:
        a = y
        b = x
    return a + nd.log(1 + nd.exp(b-a)) - x
grad(stable_log_exp, x, y)
print('x.grad =', x.grad)
print('y.grad =', y.grad)
```

```
x.grad =
[-1.]
<NDArray 1 @cpu(0)>
y.grad =
[1.]
<NDArray 1 @cpu(0)>
```

In [ ]: