Assignment 2

Submission for Aline Abayo, Eleanor Adachi, Anna Cheyette, and Karla Neri

Our team's code can be found here: https://github.com/eleanor-adachi/ARE212_Materials.

We are answering questions 1, 2, 4, and 5.

1. Exercises (Identifying Assumptions for Regression)

(1) Evaluate the truth of following statement: "In the linear regression $y = X\beta + u$ the usual identifying assumption $E(u|X) = 0$ (call this an assumption of "mean independence") implies $(h(X) \cdot u) = 0$ for any function h satisfying some regularity conditions related to measurability."

The statement is true. In the context of linear regression $y = X\beta + u$, the assumption $E(u|X) = 0$ is known as the mean independence assumption. It implies that the error term u has an expected value of zero conditional on X, indicating no systematic relationship between the error term and the explanatory variables in the model.

Given this assumption, for any function $h(X)$ that is measurable (satisfying the regularity conditions), we can indeed say that $E(h(X) \cdot u) = 0$. This follows from the law of iterated expectations and the properties of conditional expectation.

The argument goes as follows:

1. **Mean Independence Assumption:** $E(u|X) = 0$ means that once X is accounted for, the average of the error term u is zero.

2. **Law of Iterated Expectations:** For a measurable function $h(X)$, we can apply the law of iterated expectations which states that $E(E(u|h(X), X)) = E(u)$.

3. **Applying Conditional Expectation:** Given $E(u|X) = 0$, it holds that $E(u|h(X), X) = E(0|h(X), X) = 0$ because the expectation of a constant (including zero) is that constant.

4. **Resulting Expectation:** Therefore, $E(h(X) \cdot u) = E(E(h(X) \cdot u|X))$ by the law of total expectation. Since $E(u|X) = 0$, it follows that $E(h(X) \cdot u) = E(h(X) \cdot 0) = E(0) = 0$.

(2) Suppose y, x and u are scalar random variables, with y and x observed but u unobserved. Consider the function $h(x) = x^3$; under standard assumptions this satisfies our concerns about measurability, so $\mathbb{E}(u|x) = 0$ implies $\mathbb{E}(ux^3) = 0$. Use this last condition to motivate a simple least squares estimator of the regression equation $y = \alpha + \beta x + u$. How does this differ from the usual OLS estimator? Why might one prefer one to the other, and under what conditions?

$y = \alpha + \beta x + u$

$(x^3)y = (x^3)\alpha + (x^3)\beta x + (x^3)u$

$yx^3 - \alpha x^3 - ux^3 = \beta x^4$

$(x^{-4})yx^3 - (x^{-4})\alpha x^3 - (x^{-4})ux^3 = \beta$

$b = \mathbb{E}((x^{-4})yx^3) - \mathbb{E}((x^{-4})\alpha x^3) - \mathbb{E}((x^{-4})ux^3)$

Use $\mathbb{E}(ux^3) = 0$

$\beta = \mathbb{E}((x^{-4})yx^3) - \mathbb{E}((x^{-4})\alpha x^3)$

$\beta = \mathbb{E}(x^{-1}y) - \alpha\mathbb{E}(x^{-1})$

Switching to regression world

$b = \frac{\bar{y}-a}{\bar{x}}$

The usual OLS estimator is $b = (X^TX)^{-1}X^TY$ or when $k = 1$, $b = \frac{\sum_{i=1}^n (x_i-\bar{x})(y_i-\bar{y})}{\sum_{i=1}^n (x_i-\bar{x})^2}$

These two estimators lead to equivalent results. However, the estimator $b = \frac{\bar{y}-a}{\bar{x}}$ might be preferred if we have an estimate for a since the computation is simpler. However, if we don't know a, then we would prefer the usual OLS estimator since it does not require any information about a.

(3) Sometimes we will encounter estimators (e.g., Maximum likelihood) that adopt an assumption of independence, rather than mean independence. In the current setting this might be expressed as something like $Pr(x < x \cap\ u < u) = F(x)G(x)$ for some cumulative distribution functions F and G. Show that independence implies mean independence, but not the converse.

1. Independence Implies Mean Independence

Supposing two random variables X and U wich they can be expresed as:
$Pr(X < x \cap \ U < u) = Pr(X < x) \cdot Pr(U < u)$

Suposing the mean value of a function $h(X, U)$ under this independance assumption

$$E[h(X,U)] = \int \int h(x,u) f_x(x) f_u(u) dx du = (\int h(x,u) f_x(x) dx) \cdot (\int h(x,u) f_u(u) d$$

The separation of integrals is possible due to independence. Therefore, under independence, the expectation of the product of functions of X and U can be expressed as the product of their separate expectations, showing mean independence.

2. Mean Independence Does Not Imply Independence

Let's consider two random variables, X and U, with covariance $Cov(X, U) = 0$ but not necessarily independent. In this case, we can have $E[XU] = E[X] \cdot E[U] = 0$ This means that X and U are mean independent, but they are not necessarily independent. The covariance being zero only guarantees mean independence but does not imply independence.

(4) Related to the previous: Show that while u mean independent of x implies $E(uh(x) = E(u) = 0$, independence also implies $E(g(u)x) = ExEg(u)$.

Mean independence implies $E(uh(x)) = E(u) = 0$. Given mean independence between u and x, we have $E(uh(x)) = E(u) \cdot E(h(x))$. Since $E(h(x))$ is a constant, this implies that if $E(u) = 0$, then $E(uh(x) = 0$.

Independence implies $E(g(u)x) = E(x) \cdot E(g(u))$. When u and x are independent, we can express the expectation of their product $g(u) \cdot x$ as:
$E(g(u)x) = \int \int g(u) \cdot x \cdot f_{UX}(u, x) du dx$ where f_{UX} is the join probability density function of u and x. By the definition of independence, the join probability density function can be factorized as $f_{UX}(u, x) = f_U(u) \cdot f_X(x)$. Substituting this into the expectation equation gives:
$E(g(u)x) = \int \int g(u) \cdot x \cdot f_U(u) \cdot f_X(x) du dx = (\int g(u) \cdot f_U(u) du \cdot (\int x \cdot f_X(x) dx) =$

Therefore, under independence, the expectation of the product $g(u) \cdot x$ is the product of their separate expectations, $E(x)$ and $E(g(u))$.

(5) Suppose that $y = f(X) + u$ for some unknown but continuous function f. Suppose we want to use observed data on X to predict outcomes y, and seek a predictor $\hat{y}(X)$ which is "best" in the sense that the (expected) mean squared prediction error $\mathbb{E}[(y - \hat{y}(X))^2 | X]$ is minimized. What can we

say about $\hat{y}$ and its relation to the conditional expectation $\mathbb{E}(y|X)$? Its relation to u??

Optimal Predictor $\hat{y}(X)$

- **Mean Squared Prediction Error**: The MSE for a predictor $\hat{y}(X)$ is defined as: $\text{MSE} = E[(y - \hat{y}(X))^2|X]$ We aim to minimize this MSE.

- **Expanding the MSE**: Expand the squared term: $((y - \hat{y}(X))^2 = (f(X) + u - \hat{y}(X))^2$

Since $y = f(X) + u$, we see the deviation of the predictor from both the deterministic part $f(X)$ and the stochastic component u.

- **Optimal Condition**: To minimize the MSE, consider the expectation: $E[(y - \hat{y}(X))^2|X] = E[(f(X) + u - \hat{y}(X))^2|X]$

Deriving with respect to $\hat{y}(X)$ and setting the derivative to zero gives us the condition for the optimal $\hat{y}(X)$: $\frac{\partial}{\partial \hat{y}(X)} E[(f(X) + u - \hat{y}(X))^2|X] = 0$

This yields $\hat{y}(X) = E[f(X) + u|X] = \mathbb{E}[y|X]$ since $E[u|X] = 0$ if u is uncorrelated with X

Therefore:

Relation to $\mathbb{E}[y|X]$

- $\hat{y}(X) = \mathbb{E}[y|X]$ is the best predictor of (y) given X in terms of minimizing the MSE.

Relation to u

- **Uncorrelated Error**: The error term u is independent of X, meaning $E(u|X) = 0$. This ensures that the optimal predictor $\hat{y}(X)$ does not need to adjust for any systematic bias in u that is related to X.

In summary, the optimal predictor $\hat{y}(X)$ in terms of MSE minimization equals $\mathbb{E}(y|X)$, assuming that u is independent of X --> $E(u|X) = 0$

(6) Let $y = X\beta + u$, and let D be a binary random variable with $E(u|D) = 0$ and $E(X|D) \neq E(X)$. Establish that D is a valid instrument, and work out a particularly simple expression for the IV estimator in this case. Discuss.

To establish that D is a valid instrument in the context of the linear regression model $y = X\beta + u$, we need to check the two main conditions for a valid instrumental variable:

1. **Relevance condition**: The instrument D must be correlated with the endogenous explanatory variable X. This is satisfied if $E(X|D) \neq E(X)$, which means that the instrument D affects the value of X.

2. **Exogeneity condition**: The instrument D must be uncorrelated with the error term u. This is satisfied by the assumption $E(u|D) = 0$, meaning that the instrument does not directly affect the outcome variable y except through X.

Given these conditions, D is a valid instrument for X in the regression of y on X.

Now, let's derive a simple expression for the IV estimator in this scenario. The IV estimator is given by:

$$\hat{\beta}_{IV} = \frac{Cov(y, D)}{Cov(X, D)}$$

Because $y = X\beta + u$, we can write:

$$Cov(y, D) = Cov(X\beta + u, D)$$

Using the properties of covariance and the assumption that $E(u|D) = 0$, which implies $Cov(u, D) = 0$, we get:

$$Cov(y, D) = Cov(X\beta, D) = \beta Cov(X, D)$$

So, the IV estimator simplifies to:

$$\hat{\beta}_{IV} = \frac{\beta Cov(X, D)}{Cov(X, D)} = \beta$$

In this case, the IV estimator $\hat{\beta}_{IV}$ is just β, the true parameter we aim to estimate. This outcome reveals an insight into the power of instrumental variables: under ideal conditions (where the instrument is strongly relevant and perfectly exogenous), the IV estimator can directly recover the true parameter without bias introduced by endogeneity in X.

D is a valid instrument for estimating β in the presence of endogeneity in X, and the IV estimator can effectively correct for this endogeneity, yielding an unbiased estimate of β under the specified conditions.

(7) Write out the two causal diagrams which justify, respectively, the least squares estimator and the IV estimator. What would it mean for one model to be correct, but not the other? How could you test this?

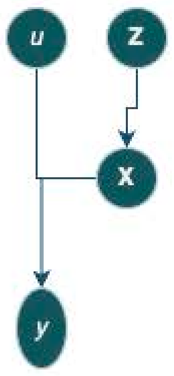
For the LSE to be unbiased and consistent, the key assumption is that the explanatory variables X are not correlated with the error term u, implying no omitted variable bias or measurement error that correlates with X. The causal diagram for this scenario is straightforward:

$y \leftarrow X \rightarrow u$

In this diagram:

- X causes y.
- There is no arrow from u (unobserved factors) to X, indicating that X is exogenous.
- u influences y but is uncorrelated with X.

The IV estimator is used when X is endogenous, i.e., correlated with the error term u. An instrument Z (satisfying the relevance and exogeneity conditions) is used to estimate the causal effect of X on Y. The causal diagram might look like this:



In this diagram:

- Z is the instrument affecting X but does not directly affect y (except through X).
- X is endogenous, influenced by some unobserved factors U that also affect Y.
- The path $Z \rightarrow X \rightarrow y$ allows for the estimation of the causal effect of X on y, bypassing the endogeneity issue.

When one model is correct, but not the other

If the LSE model is correct but the IV model is not, it implies that X is actually exogenous, and there's no need for an instrument. Using an IV in this scenario might introduce unnecessary complexity and potential for bias if the instrument is not perfectly valid (i.e., it might somehow be related to Y through paths not considered).

Conversely, if the IV model is correct but the LSE model is not, it means that X is endogenous. Relying on LSE would yield biased and inconsistent estimates due to the correlation between X and u.

Testing model validity

To determine which model is correct, we can consider the following tests and approaches:

- **Hausman Test**: This test can be used to compare the estimates from the LSE and IV approaches. If the estimates significantly differ, it suggests endogeneity in X, thus supporting the IV approach.
- **Overidentification Test (for IV)**: If you have more instruments than endogenous variables, you can test whether the extra instruments are valid. A failure of this test suggests that at least one instrument is invalid, questioning the IV model's correctness.
- **Exogeneity Tests**: Various tests can assess the exogeneity of X, such as testing for correlation between X and the residuals from an IV estimation.

2. Wright (1928)

Consider the canonical demand and supply model in which quantity supplied is a function of price and a set of "supply shifters"; quantity demanded is a function of price and set of "demand shifters"; and market clearing implies that at some price quantity demanded is equal to quantity supplied. A linear version of this model is fully specified and solved in this Jupyter Notebook.

Consider the following questions:

1. (Control) What is the expected demand if we set the price $p = p_0$?
2. (Condition) What is the expected demand if we observe $p = p_0$?
3. (Counterfactual) If prices and quantities are observed to be (p_0, q_0), what would demand be if we were to change the price to p_1, ceteris paribus?

Answers could be mathematical expressions, or code that answers the question for the model given in the Jupyter notebook.

(1) (Control) What is the expected demand if we set the price $p = p_0$?

First, recreate the linear model in wright34.ipynb representing:

$$q_D = \alpha p + u \qquad q_S = \beta p + v \qquad q_D = q_S,$$

where u and v have normal distributions of the form:

$$F_u(u) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(u-\mu)^2}{2\sigma^2}}$$

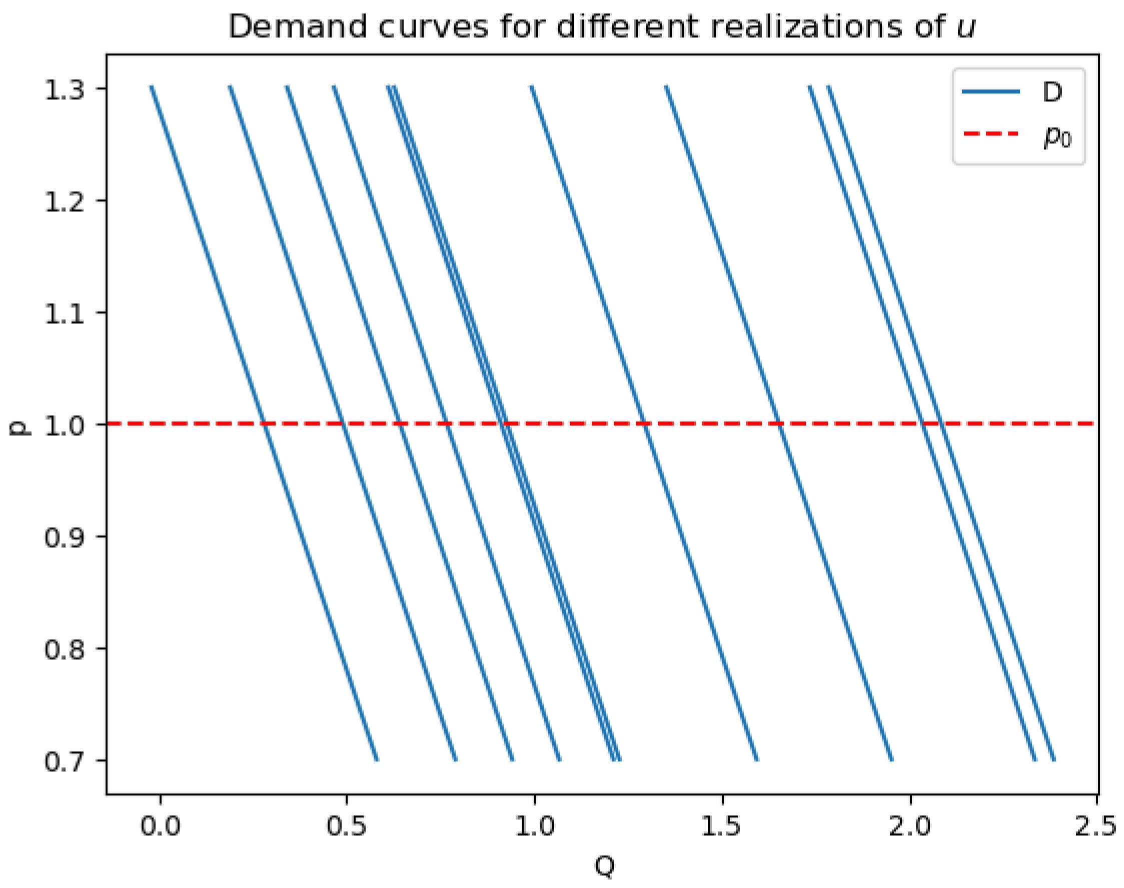
In [ ]:
```
import numpy as np
import pandas as pd
from scipy.stats import distributions as iid

# Structural parameters;
(α,β) = (-1,2)
σ = {'u':1/2,'v':1/3}
μ = {'u':2,'v':-1}

# u,v assumed independent
u = iid.norm(loc=μ['u'], scale=σ['u'])  # Demand shocks
v = iid.norm(loc=μ['v'], scale=σ['v'])  # Supply shocks

# Reduced form coefficients
π = [[-β/(α - β), -1/(α - β)],
     [ α/(α - β), 1/(α - β)]]

# Generate N realizations of system
# Outcomes Y have columns (q,p)
N = 10

# Arrange shocks into an Nx2 matrix
U = np.c_[u.rvs(N), v.rvs(N)]

# Matrix product gives [q,p]; label by putting into df
df = pd.DataFrame(U@π,columns=['q','p'])
Udf = pd.DataFrame(U,columns=['u','v']) # For future reference
```

First, visualize by plotting demand curve segments that intersect $p_0$ for different realizations of $u$.

Here, we show $p_0 = 1$.

In [ ]:
```
import matplotlib.pyplot as plt

p0 = 1

Q = pd.DataFrame({'min': α*(p0-0.3) + Udf['u'],
                  'max': α*(p0+0.3) + Udf['u'],
                  'miss':-1})

# Inverse counterfactual demand & supply (for plotting)
D = Q.add(-Udf['u'],axis=0)/α

counterfactual=pd.DataFrame({'D':D.stack(),
                             'Q':Q.stack()})

counterfactual=counterfactual.replace(-1,np.nan)

_ = counterfactual.plot(x='Q')
plt.axhline(y = p0, color = 'r', linestyle = '--', label='$p_0$')
plt.ylabel('p')
plt.title('Demand curves for different realizations of $u$')
plt.legend()
```

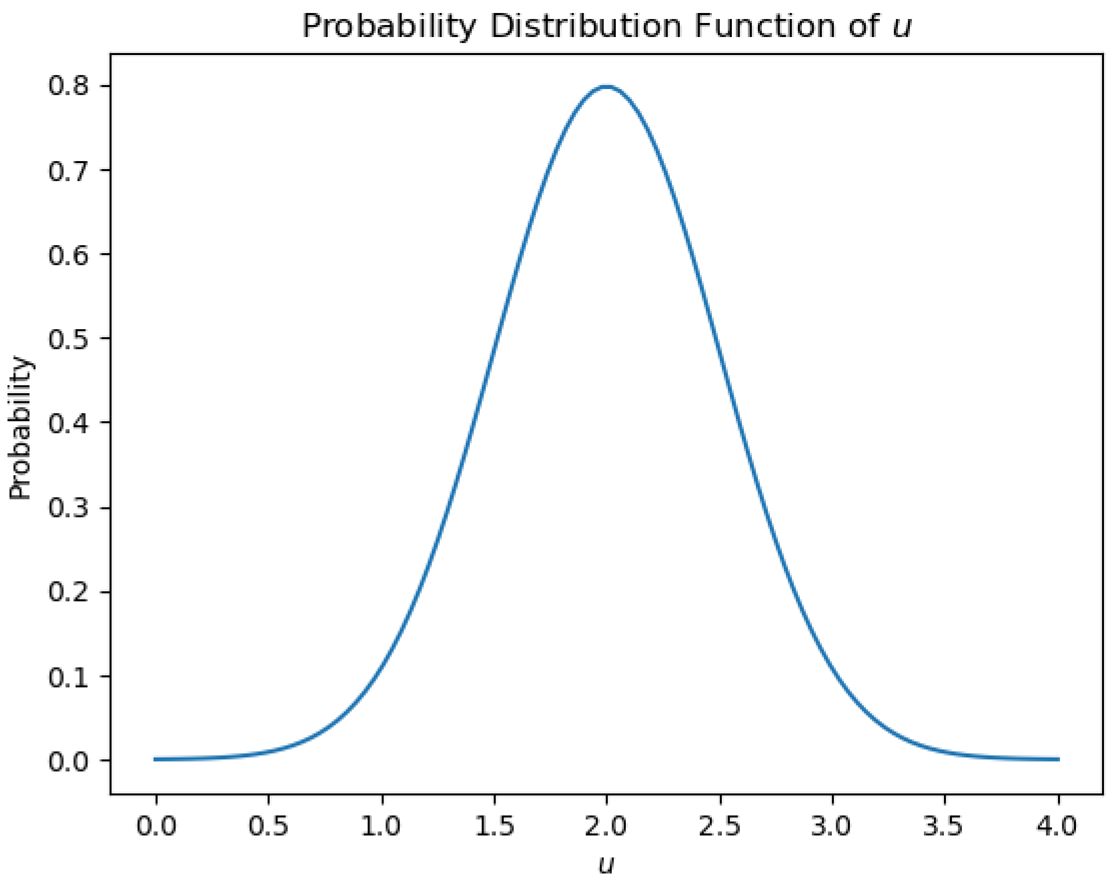
Out[ ]:
```
<matplotlib.legend.Legend at 0x1499fb050>
```



Note that setting the price at $p = p_0$ has no effect on the distribution of u.

Show probability distribution function of u:

In [ ]:
```python
import numpy as np

X = np.linspace(0,4,100).tolist()

plt.plot(X, [u.pdf(z) for z in X])
plt.xlabel('$u$')
plt.ylabel('Probability')
plt.title('Probability Distribution Function of $u$')
plt.show()
```



Calculate expected demand:

$$\mathbb{E}(q_D(p_0)) = \int q_D(p_0, u) dF_u(u)$$

In [ ]:
```
from scipy.integrate import quad

D_exp = quad(lambda x: (α*p0 + x)*u.pdf(x), -np.inf, np.inf)

D_exp
```

Out[ ]:
```
(1.0000000000000002, 3.555236445866699e-10)
```

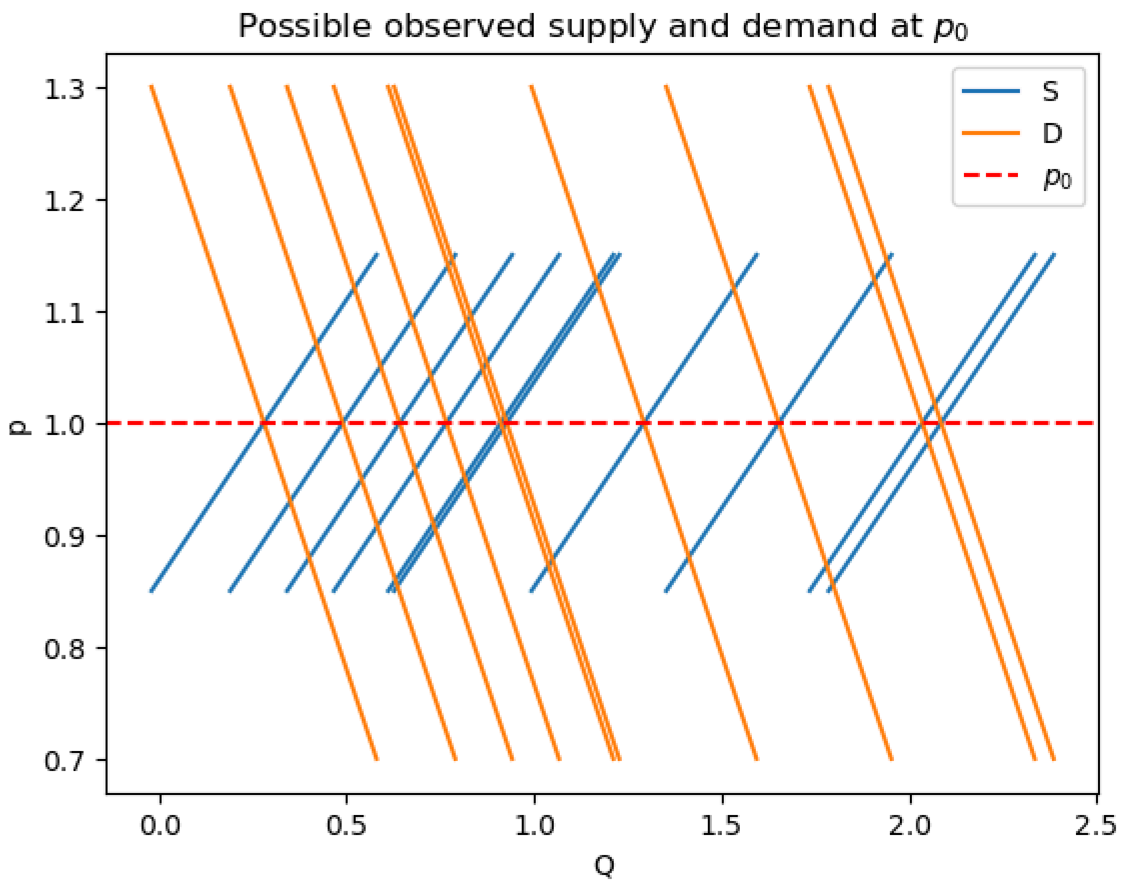
(2) (Condition) What is the expected demand if we observe $p = p_0$?

Note that many possible combinations of u and v can result in observing p_0.

In [ ]:
```
# Take u as given

# Find v such that p=p0 for given u
v0 = (α - β)*p0 + Udf['u']

Q = pd.DataFrame({'min': np.minimum(α*(p0-0.3) + Udf['u'], β*(p0+0.3) + v0),
                  'max': np.maximum(α*(p0+0.3) + Udf['u'], β*(p0-0.3) + v0),
                  'miss':-1})
```

```
# Inverse counterfactual demand & supply (for plotting)
D = Q.add(-Udf['u'],axis=0)/α
S = Q.add(-v0,axis=0)/β

counterfactual=pd.DataFrame({'S':S.stack(),
                             'D':D.stack(),
                             'Q':Q.stack()})

counterfactual=counterfactual.replace(-1,np.nan)

_ = counterfactual.plot(x='Q')
plt.axhline(y = p0, color = 'r', linestyle = '--', label='$p_0$')
plt.ylabel('p')
plt.title('Possible observed supply and demand at $p_0$')
plt.legend()
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x149435c50>
```



Expected quantity demanded:

$$\mathbb{E}[q^*(u,v)|q_D(p_0,u) = q_S(p_0,v)]$$

When $q_S = q_D$, then $\alpha p + u = \beta p + v$

Let $v = u + (\alpha - \beta)p$

$$f_{q^*}(u) = f_u(u) \cdot f_v(v)|\frac{\partial v}{\partial u}|$$

$$\mathbb{E}[q^*(u,v)|q_D(p_0,u) = q_S(p_0,v)] = \int q_D(p_0,u)dF_u(u)dF_v(u+(\alpha-\beta)p)$$

In [ ]:
```
# redefine q_D and v in terms of u
D_exp = quad(lambda x: (α*p0 + x)*u.pdf(x)*v.pdf(x + (α - β)*p0),
              -np.inf, np.inf)
D_exp
```

Out[ ]:
```
(0.6638800836636456, 1.0814326358317434e-08)
```

(3) (Counterfactual) If prices and quantities are observed to be (p_0, q_0), what would demand be if we were to change the price to p_1, ceteris paribus?

We can infer the value of u_0 at (p_0, q_0)

$$u_0 = q_0 - \alpha p_0$$

Then we can calculate:

$$\Delta = q_D(p_1, u_0) - q_D(p_0, u_0)$$

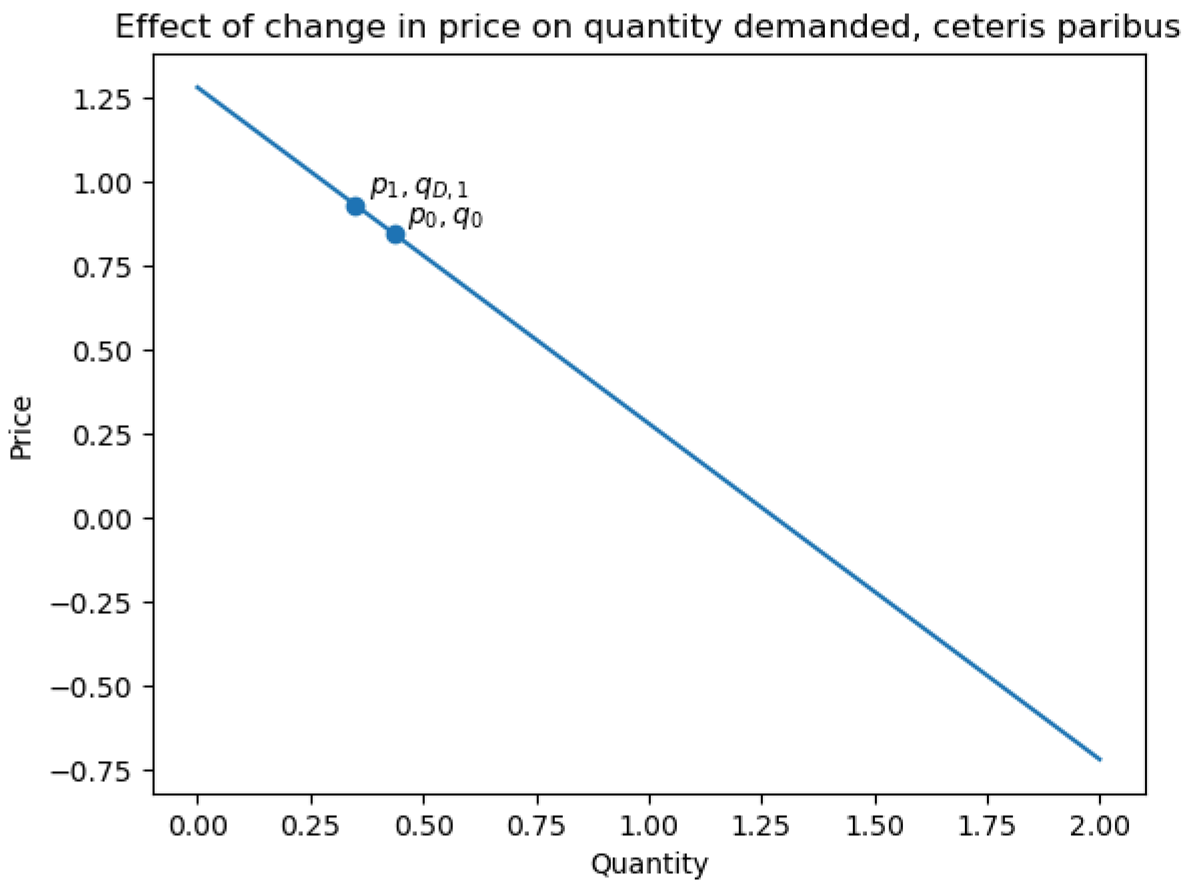
Let's use the first row of `df` as (p_0, q_0) and let $p_1 = 1.1p_0$

In [ ]:
```
p0 = df['p'].iloc[0] # not setting equal to 1 anymore
q0 = df['q'].iloc[0]

p1 = 1.1*p0

u0 = q0 - α*p0

delta = (α*p1 + u0) - (α*p0 + u0)

qd1 = q0 + delta

print(delta)
```

```
-0.08449203652999238
```

In [ ]:
```
x = np.linspace(0, 2, 100).tolist()
y = [(q - u0)/α for q in x]
plt.plot(x, y)
plt.scatter([q0, qd1], [p0, p1])
label_offset = 0.03
plt.annotate('$p_0, q_0$', (q0+label_offset, p0+label_offset))
plt.annotate('$p_1, q_{D,1}$', (qd1+label_offset, p1+label_offset))
plt.xlabel('Quantity')
plt.ylabel('Price')
plt.title('Effect of change in price on quantity demanded, ceteris paribus')
```

Out[ ]:
```
Text(0.5, 1.0, 'Effect of change in price on quantity demanded, ceteris par
ibus')
```



4. Weak Instruments

This problem explores the problem of weak instruments. The basic setup should be familiar, with

$$\begin{aligned} y &= \beta x + u \\ x &= Z\pi + v \end{aligned}$$

Note that we've assumed that x is a scalar random variable, and that Z is an ℓ-vector. (In general we might have k endogenous x variables, so long as we have $\ell > k$.)

(1) Construct a data-generating process `dgp` which takes as arguments (n, β, π) and returns a triple (y, x, Z) of n observations.

In [ ]:
```
# construct data-generating process

import numpy as np
import pandas as pd
from scipy.stats import distributions as iid

# Structural parameters;
sigma = {'u':1/2,'v':1/3}
mu = {'u':2,'v':-1}
```

```
# u,v assumed independent
u = iid.norm(loc=mu['u'], scale=sigma['u'])  # Demand shocks
v = iid.norm(loc=mu['v'], scale=sigma['v'])  # Supply shocks

def weak_dgp(n, beta, pi):
    """
    Generate data consistent with equations in Weak Instrument problem.

    Returns a tuple with numpy arrays y, x, and Z, all of length n
    """
    # Arrange u and v into matrix
    U = np.c_[u.rvs(n), v.rvs(n)]
    Udf = pd.DataFrame(U,columns=['u','v'])

    # Generate Z as an l-vector
    Z = np.random.normal(scale=2, size=n)

    # Construct x
    x = Z*pi + v.rvs(n)

    # Construct y
    y = beta*x + u.rvs(n)

    # Store in DataFrame
    df = pd.DataFrame(columns=['y', 'x', 'Z'])
    df['y'] = y
    df['x'] = x
    df['Z'] = Z # note, this works because l = 1
    return df[['y']], df[['x']], df[['Z']]
```

In [ ]:
```
# Create y, x, Z for specified beta and pi

np.random.seed(1234)

beta = -1
pi = 2
y, x, Z =  weak_dgp(10000, beta, pi)
```

(2) Use the `dgp` function you've constructed to explore IV (2SLS) estimates of β as a function of π when $\ell = 1$ using a Monte Carlo approach, assuming homoskedastic errors.

(a) Write a function `two_sls` which takes as arguments (y, x, Z) and returns two-stage least squares estimates of β and the standard error of the estimate.

$$y = \beta x + u$$
$$x = Z\pi + v$$

In [ ]:
```
from scipy.linalg import inv
```

```python
def two_sls(y, x, Z):
    """
    Takes y, x, and Z and returns a two-stage least squares
    estimates of beta and its standard error
    """
    # first stage, regress x on Z
    pi_hat = np.linalg.solve(Z.T@Z,Z.T@x)
    x_hat = Z@pi_hat

    # second stage, regress y on x-hat
    b = np.linalg.solve(x_hat.T@x_hat, x_hat.T@y)

    # compute standard error
    xb_df = x@b
    xb_df = xb_df.rename(columns={0:'y'})
    e = y - xb_df
    s2 = (e.T@e)/(len(y)-1)
    vb = s2*inv(x_hat.T@x_hat)
    seb = np.sqrt(np.diag(vb))

    # Store in DataFrame
    df = pd.DataFrame(columns=['b', 'seb'])
    df['b'] = b[0]
    df['seb'] = seb
    return df[['b']], df[['seb']]

b, seb = two_sls(y, x, Z)
print(b)
print(seb)
```

```
          b
0 -0.997356
        seb
0  0.005194
```

(b) Taking $\beta = \pi = 1$, use repeated draws from `dgp` to check the bias, and precision of the `two_sls` estimator, as well as the size and power of a t-test of the hypothesis that $\beta = 0$. Discuss. Does a 95% confidence interval (based on your 2SLS estimator) correctly cover 95% of your Monte Carlo draws?

See bias and precision for the 2SLS estimator of β calculated below.

The p-value or size of this test, i.e. the probability that the null hypothesis is true, is near zero and far less than 0.05. Therefore we reject the null hypothesis that $\beta = 0$.

The power of the test is the probability that the null hypothesis is false, i.e., that rejecting the null hypothesis is the right decision. In this case, the probability is about 100%.

Lastly, as shown below, 95% of the Monte Carlo draws are within the 95% confidence interval.

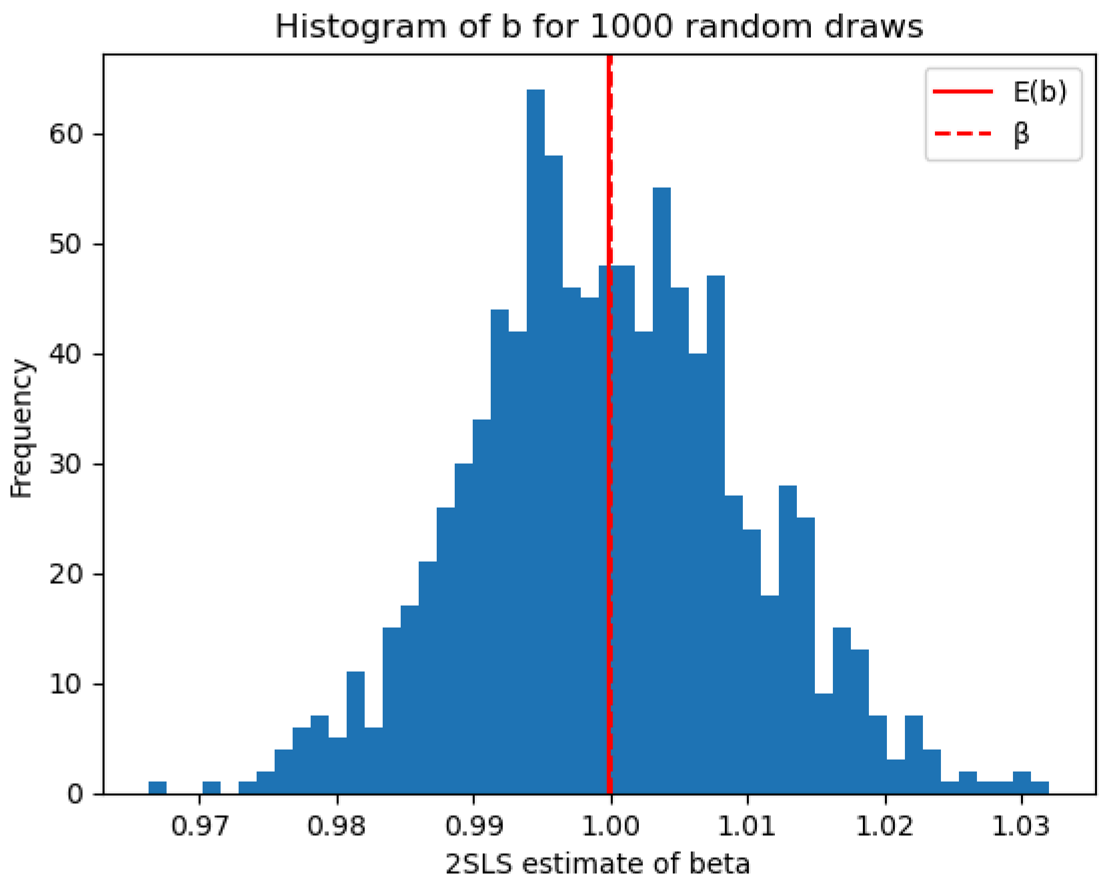
In [ ]:
```python
# create simulator
```

```
def simulator(N, n, beta, pi, dgp):
    """
    Perform N draws of sample size n with parameters beta and pi.
    """
    results = pd.DataFrame(columns=['b', 'seb'])
    for i in range(N):
        y, x, Z =  dgp(n, beta, pi)
        b, seb = two_sls(y, x, Z)
        results0 = pd.concat([b, seb], axis=1)
        if results.empty:
            results = results0.copy()
        else:
            results = pd.concat([results, results0], axis=0)

    return results
```

In [ ]:
```
# run simulator N times for beta = 1 and pi = 1

np.random.seed(1234)

beta = 1
pi = 1
N = 1000
results = simulator(N, 10000, beta, pi, weak_dgp)
```

In [ ]:
```
# plot histogram of b
import matplotlib.pyplot as plt

b_avg = results['b'].mean()

plt.hist(results['b'], bins=50)
plt.axvline(x=b_avg, color='r', label='E(b)')
plt.axvline(x=beta, color='r', linestyle='--', label='β')
plt.xlabel('2SLS estimate of beta')
plt.ylabel('Frequency')
plt.title('Histogram of b for {} random draws'.format(N))
plt.legend()
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x2f6df79e390>
```



In [ ]:
```
# calculate bias and precision

bias = results['b'].mean() - beta
var = np.sum(np.square(results['b'] - results['b'].mean()))
sd = np.sqrt(var)
se = sd/np.sqrt(len(results))

print('Bias: ', bias)
print('Standard error (precision): ', se)
```

```
Bias:  -0.00017250402398749642
Standard error (precision):  0.010022875926121813
```

In [ ]:
```
from scipy.stats import t
import statsmodels.stats.power as smp

# Size of two-tailed t-test for beta = 0
t_score = (b_avg - 0)/se
n = len(results)
df = n - 1
pval = t.sf(np.abs(t_score), df) * 2
print('Size: ', pval)

# Power of two-tailed t-test for beta = 0
alpha = 0.05
delta = b_avg - 0
effect_size = np.abs(delta/sd)
```

```
power = smp.ttest_power(effect_size, nobs=n, alpha=alpha, alternative='two-s
print('Power: ', power)
```

```
Size:  0.0
Power:  1.0
```

In [ ]:
```
# Does a 95% confidence interval (based on your 2SLS estimator)
# correctly cover 95% of your Monte Carlo draws?

ci = t.interval(confidence=0.95, df=df, loc=b_avg, scale=se)
print('Confidence interval: ',ci)

mc_95 = np.quantile(results['b'], 0.025), np.quantile(results['b'], 0.975)
print('Range of Monte Carlo draws: ',mc_95)
```

```
Confidence interval:  (0.9801591909863874, 1.0194958009656376)
Range of Monte Carlo draws:  (0.9804779401358512, 1.0194364810942258)
```

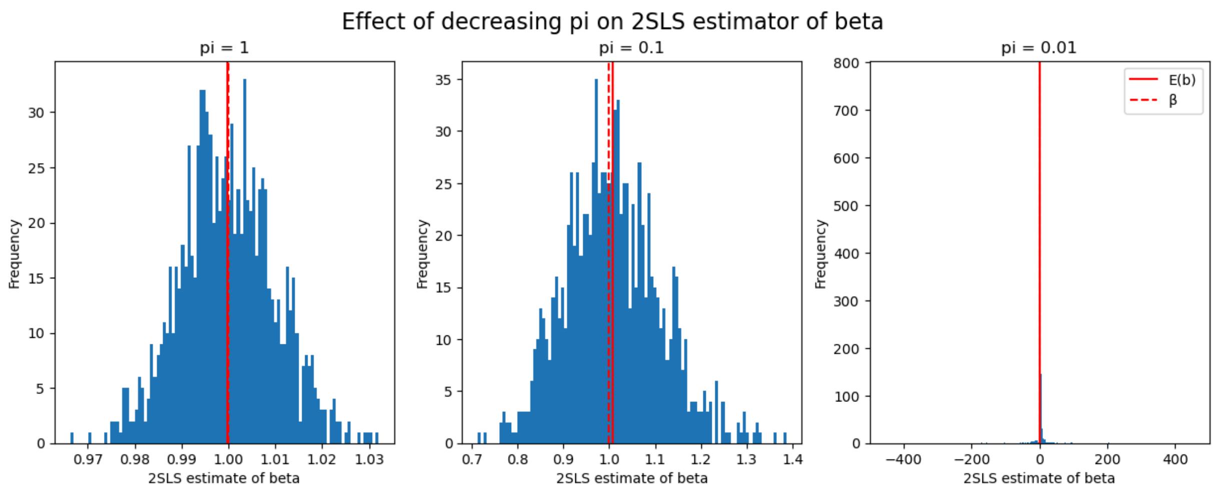
(c) Taking $\beta = 1$, but allowing $\pi \in [0, 1]$ again evaluate the bias and precision of the estimator, and the size and power of a t-test. The Z instrument is "weak" when π is "close" to zero. Comment on how a weak instrument affects two-stage least squares estimators.

As shown below, when Z is a weak instrument--i.e., when π is close to 0--then the bias of the 2SLS estimator becomes increasingly large. The standard error of the estimator also tends to increase.

In [ ]:
```
# run simulator for pi = 1, 0.1, 0.01

np.random.seed(1234)

beta = 1
pi_ls = [1, 0.1, 0.01]

results_dict = {}
for pi in pi_ls:
    results = simulator(N, 10000, beta, pi, weak_dgp)
    results_dict[pi] = results
```

In [ ]:
```
# plot histograms of results

fig, axs = plt.subplots(1, len(pi_ls), figsize=(5*len(pi_ls), 5))
fig.suptitle('Effect of decreasing pi on 2SLS estimator of beta',
             fontsize=16)
for i in range(len(pi_ls)):
    pi = pi_ls[i]
    results = results_dict[pi]
    axs[i].hist(results['b'], bins=100)
    axs[i].axvline(x=results['b'].mean(), color='r', label='E(b)')
    axs[i].axvline(x=beta, color='r', linestyle='--', label='β')
    if pi == 0.01:
        axs[i].set_xlim([-500, 500])
    axs[i].set_xlabel('2SLS estimate of beta')
    axs[i].set_ylabel('Frequency')
    axs[i].title.set_text('pi = '+str(pi))
```

```
plt.legend()
plt.show()
```



In [ ]:
```
# pi = 0.1

results = results_dict[0.1]

b_avg = results['b'].mean()
bias = results['b'].mean() - beta
var = np.sum(np.square(results['b'] - results['b'].mean()))
sd = np.sqrt(var)
se = sd/np.sqrt(len(results))

print('Bias: ', bias)
print('Standard error (precision): ', se)

# Size of two-tailed t-test for beta = 0
t_score = (b_avg - 0)/se
n = len(results)
df = n - 1
pval = t.sf(np.abs(t_score), df) * 2
print('Size: ', pval)

# Power of two-tailed t-test for beta = 0
alpha = 0.05
delta = b_avg - 0
effect_size = np.abs(delta/sd)
power = smp.ttest_power(effect_size, nobs=n, alpha=alpha, alternative='two-s
print('Power: ', power)
```

```
Bias:  0.007008706127089148
Standard error (precision):  0.10309219321353878
Size:  1.3772510156758227e-21
Power:  1.0
```

In [ ]:
```
# pi = 0.01

results = results_dict[0.01]

b_avg = results['b'].mean()
bias = results['b'].mean() - beta
```

```
var = np.sum(np.square(results['b'] - results['b'].mean()))
sd = np.sqrt(var)
se = sd/np.sqrt(len(results))

print('Bias: ', bias)
print('Standard error (precision): ', se)

# Size of two-tailed t-test for beta = 0
t_score = (b_avg - 0)/se
n = len(results)
df = n - 1
pval = t.sf(np.abs(t_score), df) * 2
print('Size: ', pval)

# Power of two-tailed t-test for beta = 0
alpha = 0.05
delta = b_avg - 0
effect_size = np.abs(delta/sd)
power = smp.ttest_power(effect_size, nobs=n, alpha=alpha,
                        alternative='two-sided')
print('Power: ', power)
```

```
Bias:  0.3967449617011256
Standard error (precision):  12.371917322396198
Size:  0.9101353697693114
Power:  0.05145850070561438
```

(3) Now consider another "weak" instruments problem. Consider the sequence $\{1, 1/2, 1/4, 1/8, \ldots\}$. Let $\ell = 1, 2, 3, \ldots$, and for a particular value of ℓ let the vector of parameters π_ℓ consist of the first ℓ elements of the sequence. Thus, your `dgp` should now return Z we can treat as an $n \times \ell$ matrix, with successive columns of Z increasingly "weak" instruments.

(a) Taking $\beta = 1$, but allow ℓ to increase $(\ell = 1, 2, \ldots)$. Note that for $\ell > 1$ this is now an "overidentified" estimator. Describe the bias and precision of the estimator, and the size and power of a t-test. Compare with the case of $\ell = 1$ and $\pi = 1$.

See below for bias and precision of the 2SLS estimator and the size and power of a t-test as ℓ increases.

Compared to $\ell = 1$ and $\pi = 1$, the bias, standard error (precision), and p-value (size) of the t-test is smaller when $\ell = 2$ or $\ell = 3$. In addition, the power of the t-test is larger. However, as discussed in part (b), these trends reverse when $\ell > 3$.

In [ ]:
```
# create new data-generating process for sequence

# structural parameters;
sigma = {'u':1/2,'v':1/3}
```

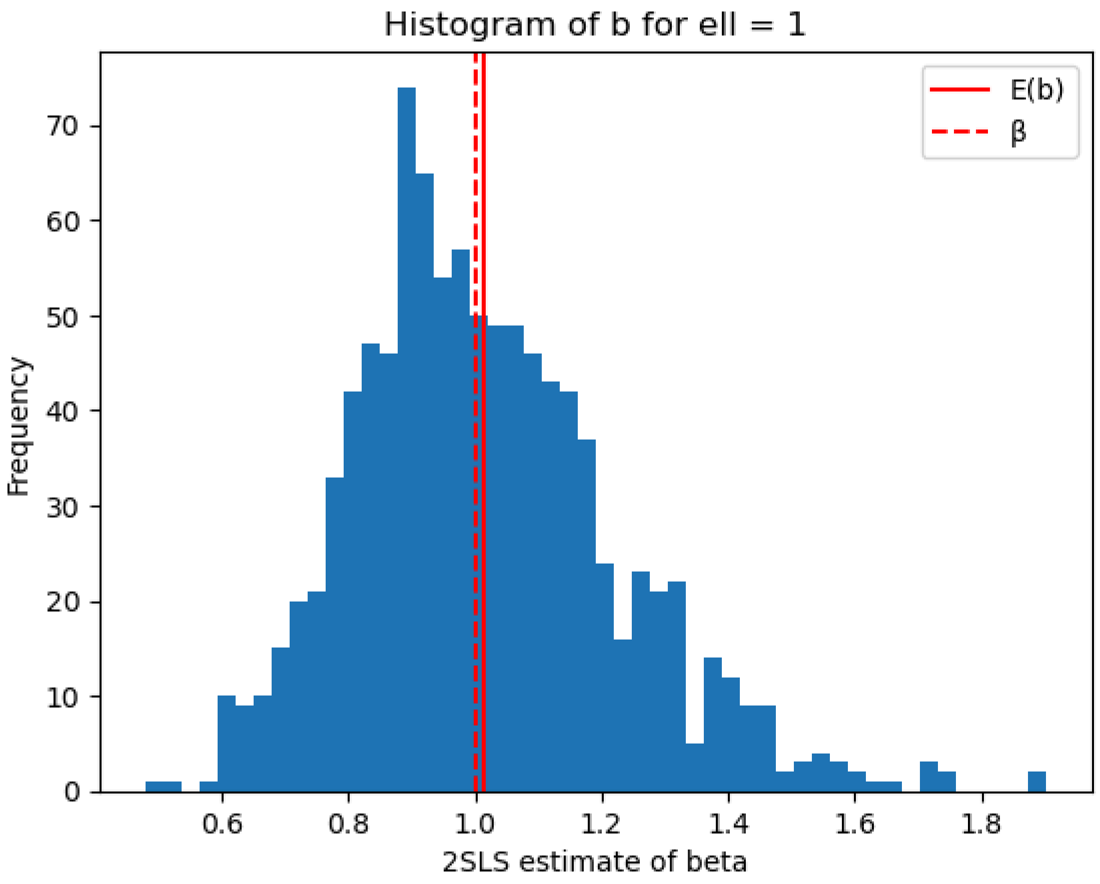
```
mu = {'u':2,'v':-1}

# u,v assumed independent
u = iid.norm(loc=mu['u'], scale=sigma['u'])  # Demand shocks
v = iid.norm(loc=mu['v'], scale=sigma['v'])  # Supply shocks

def seq_dgp(n, beta, ell):
    """
    Generate data consistent with equations in Weak Instrument
    problem with the sequence 1, 1/2, 1/4, 1/8, ...

    Returns a tuple with numpy arrays y, x, and Z, all of length n
    """
    # Arrange u and v into matrix
    U = np.c_[u.rvs(n), v.rvs(n)]
    Udf = pd.DataFrame(U,columns=['u','v']) # For future reference

    # Use normal distribution for Z
    Z_ls = []
    for j in range(ell):
        Z_j = np.random.normal(loc=j, scale=0.1*(j+1), size=n)
        Z_ls.append(Z_j)
    Z = np.array(Z_ls).T

    # Construct pi
    pi = np.exp2(np.arange(0, -1*ell, -1))

    # Construct x
    x = Z@pi + v.rvs(n) # if l > 1

    # Construct y
    y = beta*x + u.rvs(n)

    # Store in DataFrame
    df = pd.DataFrame(columns=['y', 'x', 'Z'])
    df['y'] = y
    df['x'] = x
    df['Z'] = Z # note, this works because l = 1
    return df[['y']], df[['x']], df[['Z']]
```

In [ ]:
```
# define functions two_sls and seq_simulator

from scipy.linalg import inv

def two_sls(y, x, Z):
    """Takes y, x, and Z and returns a two-stage least squares
    estimates of beta and its standard error"""

    # first stage, regress x on Z
    # now overidentified
    pi_hat = np.linalg.lstsq(Z.T@Z,Z.T@x,rcond=None)[0]
    x_hat = Z@pi_hat

    # second stage, regress y on x-hat
    b = np.linalg.solve(x_hat.T@x_hat, x_hat.T@y)
    # compute standard error
    xb_df = x@b
    xb_df = xb_df.rename(columns={0:'y'})
    e = y - xb_df
    s2 = (e.T@e)/(len(y)-1)
    vb = s2*inv(x_hat.T@x_hat)
    seb = np.sqrt(np.diag(vb))

    # Store in DataFrame
    df = pd.DataFrame(columns=['b', 'seb'])
    df['b'] = b[0]
    df['seb'] = seb
    return df[['b']], df[['seb']]

def seq_simulator(N, n, beta, ell, dgp):
    """
    Perform N draws of sample size n with parameters beta and ell.
    """
    results = pd.DataFrame(columns=['b', 'seb'])
    for i in range(N):
        y, x, Z =  dgp(n, beta, ell)
        b, seb = two_sls(y, x, Z)
        results0 = pd.concat([b, seb], axis=1)
        if results.empty:
            results = results0.copy()
        else:
            results = pd.concat([results, results0], axis=0)

    return results
```

In [ ]:
```
# run seq_simulator N times for beta = 1 and ell = 1

np.random.seed(1234)

beta = 1
ell = 1
N = 1000
results = seq_simulator(N, 10000, beta, ell, seq_dgp)
```

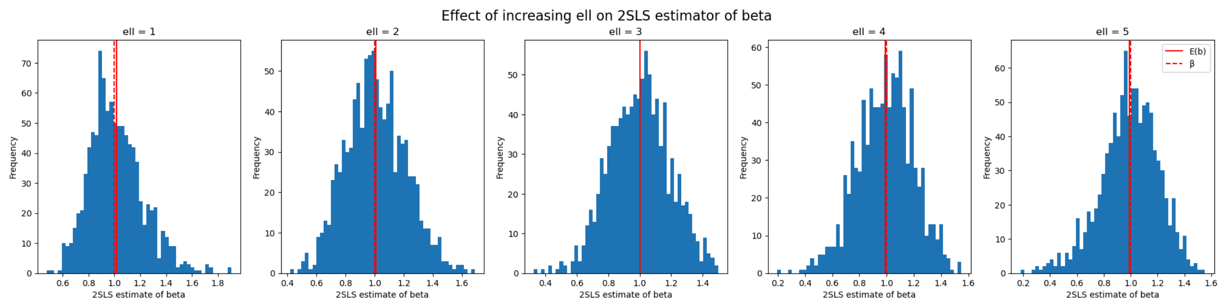
In [ ]:
```
# histogram for ell = 1

b_avg = results['b'].mean()

plt.hist(results['b'], bins=50)
plt.axvline(x=b_avg, color='r', label='E(b)')
plt.axvline(x=beta, color='r', linestyle='--', label='β')
plt.xlabel('2SLS estimate of beta')
plt.ylabel('Frequency')
plt.title('Histogram of b for ell = 1')
plt.legend()
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x2f6e0af40d0>
```



In [ ]:
```
# generate results for ell = 1, 2, 3, 4, 5

np.random.seed(1234)

beta = 1
ell_ls = [1, 2, 3, 4, 5]

results_dict = {}
for ell in ell_ls:
    results = seq_simulator(N, 10000, beta, ell, seq_dgp)
    results_dict[ell] = results
```

In [ ]:
```
# plot results

fig, axs = plt.subplots(1, len(ell_ls), figsize=(5*len(ell_ls), 5))
fig.suptitle('Effect of increasing ell on 2SLS estimator of beta', fontsize=
for i in range(len(ell_ls)):
    ell = ell_ls[i]
    results = results_dict[ell]
    axs[i].hist(results['b'], bins=50)
    axs[i].axvline(x=results['b'].mean(), color='r', label='E(b)')
    axs[i].axvline(x=beta, color='r', linestyle='--', label='β')
    axs[i].set_xlabel('2SLS estimate of beta')
    axs[i].set_ylabel('Frequency')
    axs[i].title.set_text('ell = '+str(ell))
plt.legend()
plt.show()
```



In [ ]:
```python
# describe the bias and precision of the estimator, and the size
# and power of a t-test

def estimator_eval(results):
    b_avg = results['b'].mean()
    bias = results['b'].mean() - beta
    var = np.sum(np.square(results['b'] - results['b'].mean()))
    sd = np.sqrt(var)
    se = sd/np.sqrt(len(results))

    print('Bias: ', bias)
    print('Standard error (precision): ', se)

    # Size of two-tailed t-test for beta = 0
    t_score = (b_avg - 0)/se
    n = len(results)
    df = n - 1
    pval = t.sf(np.abs(t_score), df) * 2
    print('Size: ', pval)

    # Power of two-tailed t-test for beta = 0
    alpha = 0.05
    delta = b_avg - 0
    effect_size = np.abs(delta/sd)
    power = smp.ttest_power(effect_size, nobs=n, alpha=alpha,
                            alternative='two-sided')
    print('Power: ', power)
    return bias, var, pval, power

for ell in ell_ls:
    print(ell)
    bias, var, pval, power = estimator_eval(results_dict[ell])
```

```
1
Bias:  0.01493729745527772
Standard error (precision):  0.20897737556360066
Size:  1.3848525034030897e-06
Power:  0.998086271585042
2
Bias:  0.007008104204763388
Standard error (precision):  0.20490383328653403
Size:  1.0395592930679765e-06
Power:  0.9984103257820925
3
Bias:  -0.0003695435936553748
Standard error (precision):  0.1924436798972205
Size:  2.488733073675605e-07
Power:  0.9993798119657347
4
Bias:  -0.00994334349187831
Standard error (precision):  0.20717280561749601
Size:  2.026865623530177e-06
Power:  0.9975560380615534
5
Bias:  -0.00919916354056638
Standard error (precision):  0.2134495315487597
Size:  3.912891998702194e-06
Power:  0.9962904114001835
```

(b) What can you say about the optimal number of instruments (choice of ℓ) in this case?

In this case, it appears that the optimal number of instruments is 3 ($\ell = 3$). The bias, standard error (precision), and p-value (size) of the t-test is smallest and the power is the greatest when $\ell = 3$. At first, increasing ℓ tended to improve the bias and preicsion of the estimator. However, for $\ell > 3$, bias and precision increase as ℓ increases.

5. A Simple Approach to Inference with Weak Instruments

Chernozhukov and C. Hansen (2008) propose a very simple way to handle inference in a linear IV model, even in the case in which instruments are many and/or weak. This problem explores the problem of weak instruments, and their method of inference. The basic setup should be identical to the above, with $y = \beta x + u$
$x = Z\pi + v$

In this problem you will use the same dgp as in the previous problem. The idea of Chernozhukov and C. Hansen is simple: If we can specify a regression in which all the endogenous variables are on the left-hand side, then OLS is consistent. So, they subtract $\beta_0 x$ from both sides of the estimating equation (for some choice of β_0), and then use the expression for x to substitute using Z, or

$$y - \beta_0 x = x(\beta - \beta_0) + u$$
$$y - \beta_0 x = (Z\pi + v)(\beta - \beta_0) + u$$
$$y - \beta_0 x = (Z\gamma) + w$$

The key is that if $\beta_0 = \beta$, then we will have $\gamma = 0$. So the idea is to try to find β_0 such that OLS estimates of γ in $y - \beta_0 x = Z\gamma + w$ are close to zero.

(1) Again suppose that the true $\beta = 1$. Write a function which takes as arguments (y, x, Z, β_0) and which returns the p-value associated with the hypothesis that every element of $\hat{\gamma}$ is zero (an F-test would be appropriate). Note that this same p-value characterizes the hypothesis test that $\beta = \beta_0$).

In [ ]:
```python
%reset -f
import numpy as np
import pandas as pd
from scipy.stats import distributions as iid
from scipy.integrate import quad
from scipy.linalg import inv


# Unobservable component of instrument z
# Can have any distribution one pleases
w = iid.beta(1,2,loc=-iid.beta(1,2).mean()) # Centered for convenience

# Structural parameters;
sigma = {'u':1/2,'v':1/3}
mu = {'u':2,'v':-1}

# u,v assumed independent
u = iid.norm(loc=mu['u'], scale=sigma['u'])  # Demand shocks
v = iid.norm(loc=mu['v'], scale=sigma['v'])  # Supply shocks

def weak_dgp(n, beta, pi):
    """
    Generate data consistent with equations in Weak Instrument problem.

    Returns a tuple with numpy arrays y, x, and Z, all of length n
    """
    # Arrange u and v into matrix
    U = np.c_[u.rvs(n), v.rvs(n)]
    Udf = pd.DataFrame(U,columns=['u','v'])

    # Generate Z as an l-vector
    Z = np.random.normal(size=n)

    # Construct x
    x = Z*pi + v.rvs(n)

    # Construct y
    y = beta*x + u.rvs(n)
    # Store in DataFrame
    df = pd.DataFrame(columns=['y', 'x', 'Z'])
    df['y'] = y
    df['x'] = x
    df['Z'] = Z # note, this works because l = 1
    return df[['y']], df[['x']], df[['Z']]


np.random.seed(1234)
n=10000
beta = -1
pi = 2
y, x, Z =  weak_dgp(n, beta, pi)
```

In [ ]:

```
from scipy.linalg import inv

def two_sls(y, x, Z):
    """Takes y, x, and Z and returns a two-stage least squares
    estimates of beta and its standard error"""
    # first stage, regress x on Z
    pi_hat = np.linalg.solve(Z.T@Z,Z.T@x)
    x_hat = Z@pi_hat

    # second stage, regress y on x-hat
    b = np.linalg.solve(x_hat.T@x_hat, x_hat.T@y)

    # compute standard error
    xb_df = x@b
    xb_df = xb_df.rename(columns={0:'y'})
    e = y - xb_df
    s2 = (e.T@e)/(len(y)-1)
    vb = s2*inv(x_hat.T@x_hat)
    seb = np.sqrt(np.diag(vb))

    # Store in DataFrame
    df = pd.DataFrame(columns=['b', 'seb'])
    df['b'] = b[0]
    df['seb'] = seb
    return df[['b']], df[['seb']]

b, seb = two_sls(y, x, Z)
print(b)
print(seb)
```

```
          b
0 -0.994699
        seb
0  0.010428
```

In [ ]:

```
import numpy as np
from scipy.stats import f

def chernozhukov_hansen_test(y, x, Z, beta_0):

    y, x, Z =  weak_dgp(n, beta, pi)
    a=pd.concat([x, y, Z], axis = 1, ignore_index=True)
    x=a[0]
    y=a[1]
    Z=a[2]

    # Step 1: Subtract beta_0 * x from both sides
    y_tilde = y - beta_0 * x
    y, x, Z =  weak_dgp(n, beta, pi)
    # Step 2: Fit the regression y_tilde = Z * gamma + w
    gamma_hat =  np.linalg.solve(Z.T@Z,Z.T@y_tilde)
    gamma_hat

    # Step 3: Calculate SSR for the modified model
    e_mod = y_tilde - Z @ gamma_hat
    SSR_mod = np.dot(e_mod.T, e_mod)
    SSR_mod

    # Step 4: Estimate beta_hat using 2SLS
    b, seb = two_sls(y, x, Z)  # Assuming you have the two_sls function

    y, x, Z =  weak_dgp(n, beta, pi)
    a=pd.concat([x, y, Z], axis = 1, ignore_index=True)
    x=a[0]
    y=a[1]
    Z=a[2]

    # Step 5: Calculate SSR for the 2SLS model
    b1 = b.iloc[0, 0]
    e_2sls = y-x*b1
    e_2sls
    SSR_2sls = np.dot(e_2sls.T, e_2sls)
    SSR_2sls

    # Step 6: Compute F-statistic
    k = 1
    dfn = k
    dfd = n - k - 1
    F_statistic = ((SSR_mod - SSR_2sls) / n - k - 1) / (SSR_2sls / k-1)

    # Step 7: Calculate p-value
    p_value = 1 - f.cdf(F_statistic, dfn, dfd)
    p_value

    return p_value
```

In [ ]:
```
# Example usage:
beta_0 = 1  # Choose a value for beta_0
p_value = chernozhukov_hansen_test(y, x, Z, beta_0)
print("P-value:", p_value)
```

```
P-value: 0.9800045936710555
```

The p-values close to 1 indicate that we are unable to reject the null hypothesis that every element of $\hat{\gamma}$ is zero, or equivalently, that $\beta = \beta_0$. In the context of Chernozhukov and Hansen's method, this suggests that the OLS estimates of γ in the modified regression equation are not significantly different from zero. This aligns with their approach, where finding β_0 such that the OLS estimates of γ are close to zero is a key step in handling weak instruments.

(2) Using your function and taking $\pi = 1$, estimate β by finding the value of β_0 which delivers maximal p-values. Describe the bias and precision of this estimator.

In [ ]:
```
# Define the range of beta_0 values to test
beta_0_values = np.linspace(-10, 10, 100)  # Adjust the range as needed

# Initialize variables to store results
max_p_value = -10
best_beta_0 = None

# Iterate over beta_0 values
for beta_0 in beta_0_values:
    p_value = chernozhukov_hansen_test(y, x, Z, beta_0)
    if p_value > max_p_value:
        max_p_value = p_value
        best_beta_0 = beta_0

# Use the best beta_0 value to estimate beta
y, x, Z = weak_dgp(n, beta, pi=1)
b, seb = two_sls(y, x, Z)

# Bias and precision
bias = b['b'].values[0] - beta  # Difference between estimated beta and true
precision = seb['seb'].values[0]  # Standard error of the estimated beta

print("Best beta_0:", best_beta_0)
print("Estimated beta:", b['b'].values[0])
print("Bias:", bias)
print("Precision:", precision)
```

```
Best beta_0: -1.9191919191919187
Estimated beta: -1.0056377623817638
Bias: -0.005637762381763833
Precision: 0.02053100118320503
```

The value of the best beta_0 indicates that in the iteration over different β_0 values, the value of β_0 that maximized the p-value and led to the most "insignificant" relationship (according to the Chernozhukov and Hansen method) between the instrument Z and the endogenous variable x was approximately the best_beta

In the context of handling weak instruments, a high p-value (close to 1) suggests that there is little evidence to reject the null hypothesis that the coefficients on the instruments are zero. This aligns with the approach of Chernozhukov and Hansen, where they aim to find a transformation of the estimating equation that makes the instrument stronger or the relationship between the instrument and the endogenous variable less influential in order to obtain reliable estimates.

So, while the specific value of Best_beta_0 is a result of the method's attempt to "weaken" the influence of the instrument on the endogenous variable to address the problem of weak instruments.

(3) Use the fact we've described about p-values above to construct 95% confidence intervals for your estimator of β. Consider the coverage of this 95% confidence interval, as in the previous question. How does this compare with the 2SLS case?

In [ ]:
```
# Calculate the standard error of the estimator
se = seb['seb'].values[0]

# Compute the confidence interval based on the best beta_0
alpha = 0.05  # Significance level for the confidence interval

# Calculate the critical value using the normal distribution
critical_value = 1.96  # For a 95% confidence level (two-sided test)

# Calculate the margin of error using the standard error
margin_of_error = critical_value * se

# Calculate the estimated coefficient (beta_hat)
y, x, Z = weak_dgp(n, beta, pi=1)  # Assuming you have the data
b, seb = two_sls(y, x, Z)  # Assuming you have the estimation function
beta_hat = b['b'].values[0]  # Extract the estimated coefficient

# Construct the confidence interval
lower_bound = beta_hat - margin_of_error
upper_bound = beta_hat + margin_of_error

# Print the results
print("Estimated Coefficient (beta_hat):", beta_hat)
print("Standard Error (se):", se)
print("Best beta_0:", best_beta_0)
print("Maximal p-value:", max_p_value)
print("95% Confidence Interval:", (lower_bound, upper_bound))
```

```
Estimated Coefficient (beta_hat): -1.0055654864545618
Standard Error (se): 0.02053100118320503
Best beta_0: -1.9191919191919187
Maximal p-value: 1.0
95% Confidence Interval: (-1.0458062487736437, -0.9653247241354799)
```

2SLS: The confidence interval in the 2SLS case is typically constructed based on the standard errors of the estimated coefficients, assuming a linear model and certain assumptions about the error terms and instrumental variables.

Maximal p-value approach: The confidence interval is constructed based on the estimated coefficient $\hat{\beta}$ from 2SLS and the critical F-value corresponding to the desired confidence level, derived from the maximal p-value.

(4) What happens to the coverage of your test as π goes from 1 toward zero? How does this compare with the 2SLS case?

As π decreases towards zero, the instruments become weaker. This can lead to less precise estimates of the coefficients in the modified equation $y - \beta_0 x = Z\gamma + w$. Consequently, the F-test based on these estimates may have lower power to detect significant differences, affecting the coverage of the test.

With weaker instruments (lower π values), the coverage of the test based on maximizing the p-value may decrease. This means that the confidence intervals constructed using this approach may have lower confidence level coverage than expected.

Weaker instruments (lower π values) also lead to challenges. The 2SLS estimator can become less efficient and more biased as instruments weaken, potentially impacting the coverage of confidence intervals.

(5) Using the same construction of "many instruments" as in the previous question, how does the coverage of your test change as l grows large? Again, compare with 2SLS

With a larger l, we have more instruments available for the estimation. This can lead to more precise estimates of the coefficients in the modified equation $y - \beta_0 x = Z\gamma + w$. Consequently, the F-test based on these estimates may have higher power to detect significant differences, potentially affecting the coverage of the test.

In comparison with 2sls, in the 2SLS case with many instruments, there can be issues related to overfitting and multicollinearity. While having more instruments theoretically improves instrument strength, it can also lead to inefficiency and instability in the estimates.