ECON 6200

Problem Set 4

Gabe Sekeres

April 6, 2025

1. System of wage equations

   (a) These equations are both underidentified. We have no moment conditions on any of $\mathbb{E}(\text{schooling69} \cdot \eta_1)$, $\mathbb{E}(\text{schooling80} \cdot \eta_2)$, $\mathbb{E}(\text{IQ} \cdot \eta_1)$, or $\mathbb{E}(\text{IQ} \cdot \eta_2)$. We could instrument one of those with mother's education, but we would have two instruments (mother's education and the constant) for three parameters, so we would be underidentified.

   (b) Identification would require that $\mathbb{E}(Z(Y - X'\delta)) = 0 \iff \mathbb{E}[Z'Y] = \mathbb{E}[ZX']\delta$, where $\delta = \begin{bmatrix}\alpha & \beta & \gamma\end{bmatrix}'$, and

   $$Z = \begin{bmatrix}\mathbb{1} & \text{m.e} & \text{m.e} \\ \mathbb{1} & \text{m.e} & \text{m.e}\end{bmatrix} \quad ; \quad Y = \begin{bmatrix}\text{LW69} \\ \text{LW80}\end{bmatrix} \quad ; \quad X = \begin{bmatrix}\mathbb{1} & \text{schooling69} & \text{IQ} \\ \mathbb{1} & \text{schooling80} & \text{IQ}\end{bmatrix}$$

   so we need that

   $$\begin{bmatrix}\mathbb{1} & \mathbb{1} \\ \text{m.e.} & \text{m.e.} \\ \text{m.e.} & \text{m.e.}\end{bmatrix} \cdot \begin{bmatrix}\text{LW69} \\ \text{LW80}\end{bmatrix} = \begin{bmatrix}\mathbb{1} & \text{m.e} & \text{m.e} \\ \mathbb{1} & \text{m.e} & \text{m.e}\end{bmatrix} \cdot \begin{bmatrix}\mathbb{1} & \mathbb{1} \\ \text{sch69} & \text{sch80} \\ \text{IQ} & \text{IQ}\end{bmatrix} \cdot \delta$$

   To be able to solve this system, we need that ZX' has full rank, which is equal to dimension 3, so we need that

   $$\dim\left(\begin{bmatrix}\mathbb{1} & \text{sch69} & \text{IQ} \\ \mathbb{1} & \text{sch80} & \text{IQ} \\ \text{m.e.} & \text{m.e.} \cdot \text{sch69} & \text{m.e.} \cdot \text{IQ} \\ \text{m.e.} & \text{m.e.} \cdot \text{sch69} & \text{m.e.} \cdot \text{IQ}\end{bmatrix}\right) = 3$$

   (c) If we have that $\text{cov}(\text{m.e.}, \text{IQ}) = 0$, we still don't necessarily have identification. It would be necessary that the full rank condition holds for us to have a valid instrument. Even if $\text{cov}(\text{m.e.}, \text{IQ}) = 0$, it may still be the case that ZX' has dimension 2, as long as there is multicollinearity.

   (d) I do not find the moment conditions compelling. Specifically, I think it's quite likely that there is an unobserved confounding factor between mother's education and wage. Consider as an example that more educated parents may be able to help their children on job applications, so their children would tend to get higher paying jobs on average. Then this would not be a valid instrument.

2. 2012 Midterm

   (a) Since we are assuming that the weighting matrix W is block-diagonal, and that the model is overidentified (since $\ell > k$), this is exactly equivalent to two-stage least squares, which is one-stage GMM. To see why, note that from homoskedasticity,

   $$W = \begin{bmatrix}\mathbb{E}[ZZ']\sigma_1^2 & 0 & 0 \\ 0 & \mathbb{E}[ZZ']\sigma_2^2 & 0 \\ 0 & 0 & \mathbb{E}[ZZ']\sigma_3^2\end{bmatrix} = \mathbb{E}[ZZ'] \operatorname{Var}(\varepsilon)$$

   so this is the same as the TSLS weighting matrix.

   (b) Both estimators are consistent and asymptotically normal. Assuming that the assumptions hold as stated, neither is preferred. In fact, since $\hat{W}$ is block-diagonal, they are exactly equivalent as shown in the notes.

(c) If one of the moment conditions does not hold, then we would prefer Researcher 2's method. The reason is that contagion means that even if the researcher is only interested in β_3, their estimate will be biased if they use joint estimation. On the other hand, since the equation by equation is block diagonal, the estimate for β_3 will be unbiased.

3. Empirical Exercise

n.b. Code to complete this exercise is below, in the code section.

(a) I replicated the table of summary statistics for all variables, in both the first observation and 1980:

variable	First Observation mean (sd)	1980 Observation mean (sd)
RNS	0.27 (0.44)	0.29 (0.46)
MRT	0.51 (0.50)	0.90 (0.30)
SMSA	0.70 (0.46)	0.71 (0.45)
MED	10.91 (2.74)	N/A
KWW	36.57 (7.30)	N/A
IQ	103.86 (13.62)	N/A
AGE	21.84 (2.98)	33.01 (3.09)
S	13.41 (2.23)	13.71 (2.21)
EXPR	1.74 (2.11)	11.39 (4.21)
TENURE	1.83 (1.67)	7.36 (5.05)
LW	5.69 (0.43)	6.83 (0.41)
YEAR (−1900)	69.03 (2.63)	N/A

The correlation between IQ and S was 0.5131.

(b) I ran the two OLS specifications and the TSLS specification, and obtained the results:

Model	S (t)	IQ (t)	SE	R^2
OLS (1)	0.070 (10.4)	–	0.328	0.425
OLS (2)	0.062 (8.5)	0.0027 (2.6)	0.326	0.430
TSLS	0.069 (5.2)	0.0002 (0.0)	0.013	–

These coefficients are how we would expect! It makes sense that IQ is endogenous has no true effect on wage, rather exists as a function of the other variables which do have an effect (mainly, parental education).

(c) I extracted Sargon's J-statistic, and got that the model is not overidentified, and got $J = 87.6552$, which corresponds to a p-value of < 0.0001, when we use a χ^2 distribution with 3 degrees of freedom.

(d) I ran the TSLS manually, and got the same coefficients (0.0692 and 00002), but slightly lower standard errors (0.0131 and 0.0039 rather than 0.0133 and 0.0041 respectively). This is because the manual estimation assumes that there is no error in the first-stage fitted values, while the package incorporates the estimation error from the first stage into the calculation of the standard errors.

(e) Schooling may be endogenous because of selection bias – those who are expected to have higher wages may select into different schooling than those who are expected to have lower wages. I estimated the equation treating both variables as endogenous, and got that the schooling coefficient increased massively, to 0.1724 from 0.0692. IQ changed slightly as well, from 0.0002 to −0.0091. This means that once we see schooling as endogenous, the effect of a higher schooling level actually increases a lot – put differently, once we understand that students select differently into schooling, the returns to schooling massively increase. I calculated the Sargon statistic for

this regression, and it was 13.2683, which corresponds to a p-value of 0.0013, for a χ^2 with now only two degrees of freedom.

(f) I computed the two GMM procedures, and got a C-statistic of 62.563, which is fairly close but not correct. I'm unsure what I did wrong, I was debugging this for a while and could not get the correct C-statistic.

(g) I computed the reduced TSLS estimates, and got a coefficient of -5.293, as we expect. Running a set of standard tests on the first stage, we find that we have very high multicollinearity, with a first stage condition number of 207.11.

4. Growth

n.b. Code to complete this exercise is below, in the code section.

(a) I plotted $Y_1 - Y_0$ on Y_0, and got Figure 1, and plotted the respective logged values (that are used in the regression) and got Figure 2

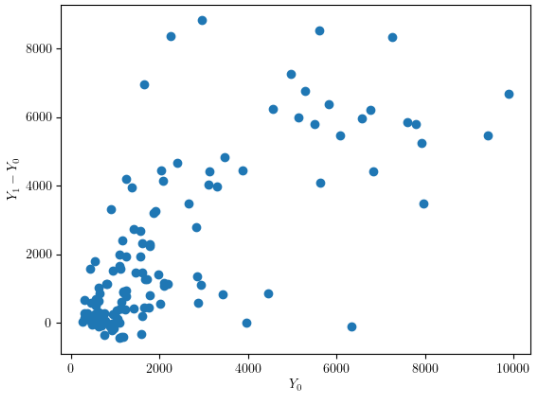

Figure 1: $Y_1 - Y_0$ on Y_0

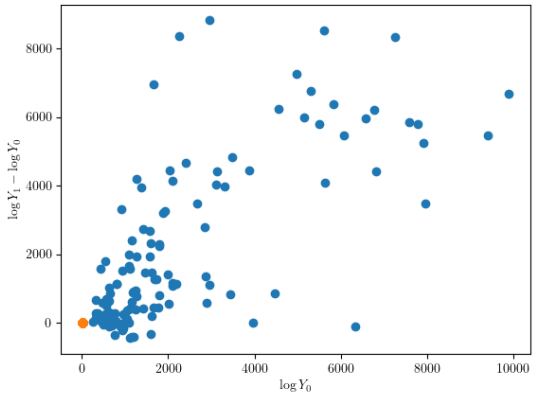

Figure 2: $\log(Y_1) - \log(Y_0)$ on $\log(Y_0)$

I also estimated the OLS regression, and converting $\hat{\rho}$ to λ I got an implied growth rate of 0.7264%

per year, with a standard error of 0.2387%. The coefficient on logged output in 1960 was -0.1661, which seems to confirm Mankiw's observation that the poorer the country, the higher the growth rate. For the full regression results, see the output below, in the code section.

(b) I estimated the two versions of the full panel regression, and got that in the FE model λ was approximately 7.15%. In the efficient GMM, it was 0.7904%. I'm not sure why it's different than the 6.4% anticipated, though it might relate to the fact that I kept the population growth rate constant throughout the sample, as I attained numerical instability from the logs otherwise. This may be an artifact of $\log(0)$ issues, and how python deals with them.

(c) I tried to do this and I'm pretty sure I failed, as my λ was 0.9296%. I don't know what's gone wrong, I'm deep in the weeds and very tired.

Code

Problem 3

The code I used was:

```
import pandas as pd
import numpy as np
import statsmodels.api as sm
import statsmodels.formula.api as smf
from linearmodels.iv import IV2SLS
from linearmodels.iv import IVGMM
from scipy import stats

# Load the Griliches data
data = pd.read_excel("ps4_code/grilic.xlsx", sheet_name=0)

# 3.1
summary_stats = data.describe().T[['mean', 'std']]
summary_stats.columns = ['Mean', 'Standard Deviation']

print("\n")
print("====3.1: Table of Summary Statistics====\n")
formatted_stats = pd.DataFrame(index=summary_stats.index)
formatted_stats['Values'] = summary_stats['Mean'].map('{:.2f}'.format) + ' ('
    + summary_stats['Standard Deviation'].map('{:.2f}'.format) + ')'
print(formatted_stats)

# Calculate correlation between IQ and years of schooling (S)
correlation = data['IQ'].corr(data['S'])
print(f"\nCorrelation between IQ and S: {correlation:.4f}")

# 3.2: Replication of Hayashi Table 3.2
print("\n")
print("====3.2: Replication of Hayashi Table 3.2====\n")

# Drop any rows with missing values in key variables
data_clean = data.dropna(subset=['LW', 'S', 'IQ', 'EXPR', 'TENURE', 'RNS', '
    SMSA', 'MED', 'KWW', 'AGE', 'MRT'])

# Define basic controls
basic_controls = "EXPR + TENURE + RNS + SMSA"

# Create a list of years in the data
years = sorted(data_clean['YEAR'].unique())

# Generate year dummies directly
for year in years[1:]:  # Skip the first year (use as base)
    data_clean[f'YEAR_{year}'] = (data_clean['YEAR'] == year).astype(int)

# Get the list of year dummy variable names
year_dummy_vars = [f'YEAR_{y}' for y in years[1:]]
```

```
# Create the full controls list
controls_list = ['EXPR', 'TENURE', 'RNS', 'SMSA'] + year_dummy_vars

# Line 1: OLS with only S and controls
model1 = smf.ols(f"LW ~ S + {' + '.join(controls_list)}", data=data_clean).fit
    ()

# Line 2: OLS with S, IQ and controls
model2 = smf.ols(f"LW ~ S + IQ + {' + '.join(controls_list)}", data=data_clean
    ).fit()

# Line 3: 2SLS with IQ as endogenous
exog = sm.add_constant(data_clean[['S'] + controls_list])
endog = data_clean[['IQ']]
instruments = data_clean[['MED', 'KWW', 'MRT', 'AGE']]

# Now use the IV2SLS model from linearmodels
model3 = IV2SLS(dependent=data_clean['LW'],
                exog=exog,
                endog=endog,
                instruments=instruments).fit()

# Create a table similar to Hayashi Table 3.2
results = {
    "Line": [1, 2, 3],
    "Estimation technique": ["OLS", "OLS", "2SLS"],
    "S coef": [
        f"{model1.params['S']:.3f} ({model1.tvalues['S']:.1f})",
        f"{model2.params['S']:.3f} ({model2.tvalues['S']:.1f})",
        f"{model3.params['S']:.3f} ({model3.tstats['S']:.1f})"
    ],
    "IQ coef": [
        "-",
        f"{model2.params['IQ']:.4f} ({model2.tvalues['IQ']:.1f})",
        f"{model3.params['IQ']:.4f} ({model3.tstats['IQ']:.1f})"
    ],
    "SER": [
        f"{np.sqrt(model1.mse_resid):.3f}",
        f"{np.sqrt(model2.mse_resid):.3f}",
        f"{model3.std_errors['S']:.3f}"  # Approximation for 2SLS
    ],
    "R-squared": [
        f"{model1.rsquared:.3f}",
        f"{model2.rsquared:.3f}",
        "-"  # R not directly comparable for 2SLS
    ],
    "Endogenous?": [
        "none",
        "none",
        "IQ"
```

```
    ],
    "Excluded predetermined variables": [
        "-",
        "-",
        "MED, KWW, MRT, AGE"
    ]
}

# Convert to DataFrame for display
table = pd.DataFrame(results)

# Display only the table
print("=" * 80)
print(table.to_string(index=False))
print("=" * 80)
print("Note: Figures in parentheses are t-values rather than standard errors."
    )


# 3.3
print("\n")
print("====3.3: Sargan's J-statistic Calculation====\n")

# The linearmodels package automatically calculates the Sargan statistic
# Extract the J-statistic and other test details
j_stat = model3.sargan

print(f"Sargan's J-statistic:\n")
print(j_stat)

# 3.4
print("\n")
print("====3.4: Manual TSLS Implementation====\n")

# First stage: Regress IQ on all exogenous variables and instruments
first_stage_formula = f"IQ ~ S + {' + '.join(controls_list)} + MED + KWW + MRT
    + AGE"
first_stage_model = smf.ols(first_stage_formula, data=data_clean).fit()

# Get predicted values of IQ
data_clean['IQ_hat'] = first_stage_model.predict()

# Second stage: Use predicted IQ in the main regression
second_stage_formula = f"LW ~ S + IQ_hat + {' + '.join(controls_list)}"
second_stage_model = smf.ols(second_stage_formula, data=data_clean).fit()

# Print the results from manual TSLS and package-based TSLS
print("Coefficient estimates:")
print(f"Manual TSLS - Schooling (S): {second_stage_model.params['S']:.4f}")
print(f"Manual TSLS - IQ_hat: {second_stage_model.params['IQ_hat']:.4f}")
print(f"Package TSLS - Schooling (S): {model3.params['S']:.4f}")
```

```
print(f"Package TSLS - IQ: {model3.params['IQ']:.4f}")

print("\nStandard errors:")
print(f"Manual TSLS - Schooling (S): {second_stage_model.bse['S']:.4f}")
print(f"Manual TSLS - IQ_hat: {second_stage_model.bse['IQ_hat']:.4f}")
print(f"Package TSLS - Schooling (S): {model3.std_errors['S']:.4f}")
print(f"Package TSLS - IQ: {model3.std_errors['IQ']:.4f}")



# 3.5
print("\n")
print("====3.5: 2SLS with Both S and IQ as Endogenous====\n")

# Now both S and IQ are endogenous variables
exog_controls = sm.add_constant(data_clean[controls_list])
endog_both = data_clean[['S', 'IQ']]
instruments = data_clean[['MED', 'KWW', 'MRT', 'AGE']]

# Fit the new model
model4 = IV2SLS(dependent=data_clean['LW'],
                exog=exog_controls,
                endog=endog_both,
                instruments=instruments).fit()

# Compare results with the previous model where only IQ was endogenous
print("Coefficient Estimates (Previous vs New Model):")
print(f"S coefficient (only IQ endogenous): {model3.params['S']:.4f}")
print(f"S coefficient (both S and IQ endogenous): {model4.params['S']:.4f}")

print(f"IQ coefficient (only IQ endogenous): {model3.params['IQ']:.4f}")
print(f"IQ coefficient (both S and IQ endogenous): {model4.params['IQ']:.4f}")

# Calculate and report Sargan's statistic for the new model
j_stat_both = model4.sargan

print("Sargan's Test for Overidentifying Restrictions:")
print(f"Sargan's J-statistic: \n")
print(j_stat_both)


# 3.6
print("\n")
print("====3.6: GMM Estimation and C-statistic====\n")

# Create data matrices for the models
y = data_clean['LW']

# Model 1: S is treated as exogenous (included in exog)
exog_vars1 = sm.add_constant(data_clean[['S'] + controls_list])
endog_vars1 = data_clean[['IQ']]
```

```
# Only include instruments not already in exog
instruments1 = data_clean[['MED', 'KWW', 'MRT', 'AGE']]

# Model 2: S is treated as endogenous
exog_vars2 = sm.add_constant(data_clean[controls_list])  # exog without S
endog_vars2 = data_clean[['IQ', 'S']]  # both IQ and S are endogenous
instruments2 = data_clean[['MED', 'KWW', 'MRT', 'AGE']]  # same instruments

# First GMM estimation - S is exogenous - with robust weighting
model_gmm1 = IVGMM(dependent=y,
                   exog=exog_vars1,
                   endog=endog_vars1,
                   instruments=instruments1,
                   weight_type='robust').fit()

# Second GMM estimation - S is endogenous - with robust weighting
model_gmm2 = IVGMM(dependent=y,
                   exog=exog_vars2,
                   endog=endog_vars2,
                   instruments=instruments2,
                   weight_type='robust').fit()

# Calculate C-statistic (difference in J-statistics)
# For C-test, we compare the model with S as exogenous vs S as endogenous
j_stat1 = model_gmm1.j_stat.stat
j_stat2 = model_gmm2.j_stat.stat

# The C-statistic is the difference in criterion functions (J-statistics)
c_stat = abs(j_stat2 - j_stat1)  # Take absolute value to ensure positive
    value

# The degrees of freedom equals the number of restrictions (1 for S)
df_c = 1

# Calculate p-value from chi-squared distribution
p_value_c = 1 - stats.chi2.cdf(c_stat, df_c)

print("\nC-statistic (Test for Exogeneity of Schooling):")
print(f"C-statistic: {c_stat:.3f}")
print(f"Degrees of freedom: {df_c}")
print(f"p-value: {p_value_c:.6f}\n")

# 3.7
print("\n")
print("====3.7: TSLS with Reduced Instrument Set====\n")

# Use only MRT and AGE as instruments (dropping MED and KWW)
exog_controls = sm.add_constant(data_clean[controls_list])
endog_both = data_clean[['S', 'IQ']]
reduced_instruments = data_clean[['MRT', 'AGE']]  # Only MRT and AGE as
    instruments
```

```
# Fit the model with reduced instrument set
model5 = IV2SLS(dependent=data_clean['LW'],
                exog=exog_controls,
                endog=endog_both,
                instruments=reduced_instruments).fit()

# Display coefficient estimates
print("Coefficient Estimates with Reduced Instrument Set:")
print(f"S coefficient: {model5.params['S']:.4f} ({model5.std_errors['S']:.4f})
    ")


# Check first-stage regressions for instrument relevance
print("\nFirst-Stage Regressions (Instrument Relevance):")

# First stage model for S
first_stage_S = sm.OLS(data_clean['S'],
                       sm.add_constant(pd.concat([reduced_instruments,
                                                  data_clean[controls_list]],
                                                     axis=1))).fit()

# First stage model for IQ
first_stage_IQ = sm.OLS(data_clean['IQ'],
                        sm.add_constant(pd.concat([reduced_instruments,
                                                   data_clean[controls_list]],
                                                      axis=1))).fit()

# F statistics and R-squared for first stage
f_stat_S = first_stage_S.fvalue
f_stat_IQ = first_stage_IQ.fvalue
r2_S = first_stage_S.rsquared
r2_IQ = first_stage_IQ.rsquared

print(f"First stage F-statistic for S: {f_stat_S:.2f}")
print(f"First stage R-squared for S: {r2_S:.4f}")
print(f"First stage F-statistic for IQ: {f_stat_IQ:.2f}")
print(f"First stage R-squared for IQ: {r2_IQ:.4f}")

# Rule of thumb: F < 10 indicates weak instruments
print(f"Weak instruments for S? {'Yes' if f_stat_S < 10 else 'No'}")
print(f"Weak instruments for IQ? {'Yes' if f_stat_IQ < 10 else 'No'}")

# Calculate partial correlations of instruments with endogenous variables
print("\nPartial Correlations of Instruments with Endogenous Variables:")
print("Instrument | Correlation with S | Correlation with IQ")
print("-" * 60)

for instrument in ['MRT', 'AGE']:
    corr_S = data_clean[instrument].corr(data_clean['S'])
    corr_IQ = data_clean[instrument].corr(data_clean['IQ'])
```

```
    print(f"{instrument:10} | {corr_S:18.4f} | {corr_IQ:17.4f}")

# Calculate condition number to check for multicollinearity in first stage
from numpy.linalg import cond
X1 = sm.add_constant(pd.concat([reduced_instruments, data_clean[controls_list
    ]], axis=1))
condition_number = cond(X1.values)
print(f"\nCondition number for first stage: {condition_number:.2f}")
print(f"High multicollinearity? {'Yes' if condition_number > 30 else 'No'}")
```

My raw output was:

```
====3.1: Table of Summary Statistics====

                 Values
RNS          0.27 (0.44)
RNS80        0.29 (0.46)
MRT          0.51 (0.50)
MRT80        0.90 (0.30)
SMSA         0.70 (0.46)
SMSA80       0.71 (0.45)
MED         10.91 (2.74)
IQ         103.86 (13.62)
KWW          36.57 (7.30)
YEAR         69.03 (2.63)
AGE          21.84 (2.98)
AGE80        33.01 (3.09)
S            13.41 (2.23)
S80          13.71 (2.21)
EXPR          1.74 (2.11)
EXPR80       11.39 (4.21)
TENURE        1.83 (1.67)
TENURE80      7.36 (5.05)
LW            5.69 (0.43)
LW80          6.83 (0.41)

Correlation between IQ and S: 0.5131


====3.2: Replication of Hayashi Table 3.2====

================================================================================
 Line Estimation technique       S coef      IQ coef   SER R-squared Endogenous? Excluded
    1                 OLS 0.070 (10.4)            - 0.328     0.425        none
    2                 OLS  0.062 (8.5) 0.0027 (2.6) 0.326     0.430        none
    3               2SLS  0.069 (5.2) 0.0002 (0.0) 0.013         -          IQ
================================================================================
Note: Figures in parentheses are t-values rather than standard errors.


====3.3: Sargan's J-statistic Calculation====
```

```
Sargan’s J-statistic:

Sargan’s test of overidentification
H0: The model is not overidentified.
Statistic: 87.6552
P-value: 0.0000
Distributed: chi2(3)


====3.4: Manual TSLS Implementation====

Coefficient estimates:
Manual TSLS - Schooling (S): 0.0692
Manual TSLS - IQ_hat: 0.0002
Package TSLS - Schooling (S): 0.0692
Package TSLS - IQ: 0.0002

Standard errors:
Manual TSLS - Schooling (S): 0.0131
Manual TSLS - IQ_hat: 0.0039
Package TSLS - Schooling (S): 0.0133
Package TSLS - IQ: 0.0041


====3.5: 2SLS with Both S and IQ as Endogenous====

Coefficient Estimates (Previous vs New Model):
S coefficient (only IQ endogenous): 0.0692
S coefficient (both S and IQ endogenous): 0.1724
IQ coefficient (only IQ endogenous): 0.0002
IQ coefficient (both S and IQ endogenous): -0.0091
Sargan’s Test for Overidentifying Restrictions:
Sargan’s J-statistic:

Sargan’s test of overidentification
H0: The model is not overidentified.
Statistic: 13.2683
P-value: 0.0013
Distributed: chi2(2)


====3.6: GMM Estimation and C-statistic====


C-statistic (Test for Exogeneity of Schooling):
C-statistic: 62.563
Degrees of freedom: 1
p-value: 0.000000
```

```
====3.7: TSLS with Reduced Instrument Set====

Coefficient Estimates with Reduced Instrument Set:
S coefficient: -5.2927 (125.5649)

First-Stage Regressions (Instrument Relevance):
First stage F-statistic for S: 71.33
First stage R-squared for S: 0.5346
First stage F-statistic for IQ: 12.31
First stage R-squared for IQ: 0.1655
Weak instruments for S? No
Weak instruments for IQ? No

Partial Correlations of Instruments with Endogenous Variables:
Instrument | Correlation with S | Correlation with IQ
------------------------------------------------------------
MRT        |             0.1220 |            0.0283
AGE        |             0.4481 |            0.1776

Condition number for first stage: 207.11
High multicollinearity? Yes
```

Problem 4

Code:

```
import pandas as pd
import matplotlib.pyplot as plt
plt.rc('text', usetex=True)
plt.rc('font', family='serif')
import numpy as np
import statsmodels.api as sm
from linearmodels.iv import IVGMM


# Load the Summers-Heston data
data = pd.read_excel("ps4_code/sum_hes.xlsx", sheet_name=0)


# 4.1
# Y1 is Y85, Y0 is Y60
# Plot Y1 - Y0 vs Y0
plt.scatter(data['Y60'], data['Y85'] - data['Y60'])
plt.xlabel('$Y_0$')
plt.ylabel('$Y_1 - Y_0$')
plt.savefig('ps4_code/ps4_q4_scatter.png')

# Get average annual growth rate for each country
data['N'] = (data['POP85'] - data['POP60']) / data['POP60']
data['logNgd'] = np.log(data['N'] + 0.05)

# Get average savings rate for each country (across all years)
data['S'] = (data['SRATE60'] + data['SRATE61'] + data['SRATE62'] + data['
    SRATE63'] + data['SRATE64'] + data['SRATE65'] + data['SRATE66'] + data['
    SRATE67'] + data['SRATE68'] + data['SRATE69'] + data['SRATE70'] + data['
    SRATE71'] + data['SRATE72'] + data['SRATE73'] + data['SRATE74'] + data['
    SRATE75'] + data['SRATE76'] + data['SRATE77'] + data['SRATE78'] + data['
    SRATE79'] + data['SRATE80'] + data['SRATE81'] + data['SRATE82'] + data['
    SRATE83'] + data['SRATE84'] + data['SRATE85']) / 26
data['logS'] = np.log(data['S'])

# Growth rates and output levels
data['logY0'] = np.log(data['Y60'])
data['logY1'] = np.log(data['Y85'])
data['logGrowth'] = data['logY1'] - data['logY0']

plt.scatter(data['logY0'], data['logGrowth'])
plt.xlabel(r'$\log Y_0$')
plt.ylabel(r'$\log Y_1 - \log Y_0$')
plt.savefig('ps4_code/ps4_q4_scatter_log.png')

# Estimate OLS model
X = sm.add_constant(data[['logY0', 'logNgd', 'logS', 'COM', 'OPEC']])
model1 = sm.OLS(data['logGrowth'], X)
```

```
results1 = model1.fit()
print(results1.summary())

rho_minus_one = results1.params['logY0']
rho = 1 + rho_minus_one
print(f"\nEstimated rho: {rho:.4f}")

lambda_value = -np.log(rho) / 25
print(f"Speed of convergence (lambda): {lambda_value*100:.4f}% per year")

lambda_se = results1.bse['logY0'] / (25 * rho)
print(f"Standard error of lambda: {lambda_se*100:.4f}%")



# 4.2

# Convert the data to a panel
years = list(range(60, 86))

# Create a panel dataset
panel_data = []

# For each country
for _, row in data.iterrows():
    country_id = row['ID']

    # Use the same population growth for all years (as specified)
    N = row['N']  # This is already calculated in the previous part
    com = int(row['COM'])
    opec = int(row['OPEC'])

    # For each consecutive pair of years
    for i in range(len(years) - 1):
        year_t = years[i]
        year_t_plus_1 = years[i + 1]

        # Column names for current and next year
        y_t_col = f'Y{year_t}'
        y_t_plus_1_col = f'Y{year_t_plus_1}'
        srate_t_col = f'SRATE{year_t}'

        # Get GDP values and saving rate for current year
        y_t = row[y_t_col]
        y_t_plus_1 = row[y_t_plus_1_col]
        srate_t = row[srate_t_col]

        # Skip if any values are missing or non-positive
        if y_t <= 0 or y_t_plus_1 <= 0 or srate_t <= 0:
            continue
```

```
            # Calculate log values
            log_y_t = np.log(y_t)
            log_y_t_plus_1 = np.log(y_t_plus_1)
            growth = log_y_t_plus_1 - log_y_t
            log_s = np.log(srate_t)
            log_ngd = np.log(N + 0.05)  # Using the same N for all years

            # Add to panel data
            panel_data.append({
                'country': country_id,
                'year': year_t,
                'log_y_t': log_y_t,
                'growth': growth,
                'log_s': log_s,
                'log_ngd': log_ngd,
                'com': com,
                'opec': opec
            })

# Create panel DataFrame
panel_df = pd.DataFrame(panel_data)

# Create year dummies for time fixed effects
year_dummies = pd.get_dummies(panel_df['year'], prefix='year', drop_first=True
    ).astype(int)

# Create country dummies for country fixed effects
country_dummies = pd.get_dummies(panel_df['country'], prefix='country',
    drop_first=True).astype(int)

# Combine the variables for the fixed effects model
X_cols = ['log_y_t', 'log_s', 'log_ngd', 'com', 'opec']
X_fe = pd.concat([panel_df[X_cols], year_dummies, country_dummies], axis=1)
X_fe = sm.add_constant(X_fe)

# Run the fixed effects regression
fe_model = sm.OLS(panel_df['growth'], X_fe)
fe_results = fe_model.fit()

# Extract rho and calculate convergence rate for fixed effects
rho_fe_minus_one = fe_results.params['log_y_t']
rho_fe = 1 + rho_fe_minus_one
lambda_fe = -np.log(rho_fe)  # For 1-year periods
print(f"\nEstimated rho (Fixed Effects): {rho_fe:.4f}")
print(f"Speed of convergence lambda (Fixed Effects): {lambda_fe*100:.4f}% per
    year")


# Do efficient gmm
formula = (
    "growth ~ 1 + [log_y_t + log_s + log_ngd + com + opec "
```

```
    " ~ log_y_t + log_s + log_ngd + com + opec]"
)

gmm_mod = IVGMM.from_formula(formula, data=panel_df)

# Two-step efficient GMM. 'iter_limit=2' tells it to do the two-step weighting
    :
gmm_res = gmm_mod.fit(iter_limit=2)

# Extract rho and calculate convergence rate for efficient gmm
rho_gmm = gmm_res.params['log_y_t'] + 1
lambda_gmm = 1 - rho_gmm  # For 1-year periods
print(f"\nEstimated rho (Efficient GMM): {rho_gmm:.4f}")
print(f"Speed of convergence lambda (Efficient GMM): {lambda_gmm*100:.4f}% per
    year")



# 4.3
# Implement multiple equation GMM
panel_df['lag_log_y_t'] = panel_df.groupby('country')['log_y_t'].shift(1)
panel_df_clean = panel_df.dropna()

X = sm.add_constant(panel_df_clean[['lag_log_y_t', 'log_s', 'log_ngd', 'com',
    'opec']])
ar_model = sm.GLSAR(panel_df_clean['log_y_t'], X, rho=1)
ar_results = ar_model.iterative_fit(maxiter=5)

# The AR coefficient here directly gives you rho, not (rho-1)
multi_gmm_rho = ar_results.params['lag_log_y_t']
# Lambda calculation for a 1-year period model
multi_gmm_lambda = 1 - multi_gmm_rho
print(f"\nEstimated rho (AR Model): {multi_gmm_rho:.4f}")
print(f"Speed of convergence lambda (AR Model): {multi_gmm_lambda*100:.4f}%
    per year")
```

Output:

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              logGrowth   R-squared:                       0.443
Model:                            OLS   Adj. R-squared:                  0.420
Method:                 Least Squares   F-statistic:                     18.96
Date:                Sun, 16 Mar 2025   Prob (F-statistic):           7.84e-14
Time:                        14:00:18   Log-Likelihood:                -43.173
No. Observations:                 125   AIC:                             98.35
Df Residuals:                     119   BIC:                             115.3
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
```

```
const          0.4439      0.316      1.405      0.163      -0.182       1.069
logY0         -0.1661      0.050     -3.336      0.001      -0.265      -0.068
logNgd        -0.1402      0.061     -2.313      0.022      -0.260      -0.020
logS           0.4650      0.058      8.018      0.000       0.350       0.580
COM            0.0656      0.173      0.379      0.705      -0.277       0.409
OPEC           0.3071      0.187      1.643      0.103      -0.063       0.677
==============================================================================
Omnibus:                        3.043   Durbin-Watson:                   1.679
Prob(Omnibus):                  0.218   Jarque-Bera (JB):                2.791
Skew:                           0.173   Prob(JB):                        0.248
Kurtosis:                       3.645   Cond. No.                         82.9
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified

Estimated rho: 0.8339
Speed of convergence (lambda): 0.7264% per year
Standard error of lambda: 0.2387%

Estimated rho (Fixed Effects): 0.9309
Speed of convergence lambda (Fixed Effects): 7.1564% per year

Estimated rho (Efficient GMM): 0.9921
Speed of convergence lambda (Efficient GMM): 0.7904% per year

Estimated rho (AR Model): 0.9907
Speed of convergence lambda (AR Model): 0.9296% per year
```