GA2code_2872991_2894452_2903029_2892824_python

October 10, 2025

[ ]:
```python
# Put all library imports in this block!
import pandas as pd
import numdifftools as nd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import math
import matplotlib.dates as mdates
import matplotlib.lines as mlines
from scipy.optimize import minimize
from statsmodels.stats.diagnostic import acorr_ljungbox
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.tsa.stattools import acf
from sklearn.metrics import r2_score
```

[ ]:
```python
# Data prep
data_vintage = pd.read_csv('2025-08-MD.csv')
data_vintage = data_vintage[['sasdate', 'UNRATE', 'INDPRO']]
data_vintage = data_vintage.iloc[1:].copy()

df_log_diff = pd.DataFrame()
df_log_diff['DATE'] = pd.to_datetime(data_vintage['sasdate'])
df_log_diff['UNRATE'] = np.log(data_vintage['UNRATE']).diff()
df_log_diff['INDPRO'] = np.log(data_vintage['INDPRO']).diff()
df_log_diff = df_log_diff.iloc[1:].copy()

display(df_log_diff.head())
```

[ ]:
```python
# Plotting log differences
plt.figure(figsize=(10,5))

## Unemployment rate
plt.subplot(2,1,1)
plt.plot(df_log_diff['DATE'], df_log_diff['UNRATE'], color='black')
plt.title('Unemployment Rate')

## Industrial production
plt.subplot(2,1,2)
```

```
plt.plot(df_log_diff['DATE'], df_log_diff['INDPRO'], color='black')
plt.title('Industrial Production')

plt.tight_layout()
plt.show()
```

1 Question 2

[ ]:
```
model_names = ('AR(1)', 'SESTAR', 'STAR')

def get_total_loss(theta, model_name, x, y = None):
  mu, delta, gamma, alpha, beta = theta
  x = np.asarray(x)
  y = np.asarray(y)

  div = 0

  if model_name == 'SESTAR':
    div = gamma / (1 + np.exp(alpha + beta * x[:-1]))
  elif model_name == 'STAR':
    div = gamma / (1 + np.exp(alpha + beta * y[:-1]))

  resid = x[1:] - mu - (delta + div) * x[:-1]

  return np.sum(resid**2)
```

[ ]:
```
def get_avg_total_loss(theta, model_name, x, y = None):
      # The loss function is the average sum of squared residuals.
  mu, delta, gamma, alpha, beta = theta
  x = np.asarray(x)
  y = np.asarray(y)

  div = 0

  if model_name == 'SESTAR':
    div = gamma / (1 + np.exp(alpha + beta * x[:-1]))
  elif model_name == 'STAR':
    div = gamma / (1 + np.exp(alpha + beta * y[:-1]))

  resid = x[1:] - mu - (delta + div) * x[:-1]

  return np.mean(resid**2)
```

[ ]:
```
# Test point for total loss (OK)
theta_star = (0, 0.3, 1, 0, 2)
```

```
for model in model_names:
  total_loss = get_total_loss(theta_star, model, df_log_diff['UNRATE'],␣
↪df_log_diff['INDPRO'])
  print(f'Total Loss for {model}: {total_loss:.6f}')
```

[ ]:
```
# Results for Q2

theta_tilde = (0, 0.1, 2, 0, 3)
results_Q2 = {}

for model in model_names:
  total_loss = get_total_loss(theta_tilde, model, df_log_diff['UNRATE'],␣
↪df_log_diff['INDPRO'])
  print(f'Total Loss for {model}: {total_loss:.6f}')

  results_Q2[model] = {
      'total_loss': total_loss
  }
```

2 Question 3

[ ]:
```
def estimate_model(theta_init, model_name, x, y=None):

  result = minimize(get_avg_total_loss,
                    theta_init,
                    args=(model_name, x, y),
                    method='BFGS',
                    options={'maxiter': 100,'gtol':1e-8})


  if model_name not in model_names:
    raise ValueError('Invalid model name specified.')

  if not result.success:
     print(f'\nWarning: Optimization failed for {model_name}.')
     print(f'Message: {result.message}\n')
  else:
     print(f'\nOptimization successful for {model_name}. Total Loss: {result.
↪fun:.6f}\n')

  return result.x, result.fun

def fit_values(theta, model_name, x, y=None):
  mu, delta, gamma, alpha, beta = theta
  x = np.asarray(x)
  if y is not None: y = np.asarray(y)
```

```
  div = np.zeros_like(x)
  if model_name == 'SESTAR':
    div[1:] = gamma / (1 + np.exp(alpha + beta * x[:-1]))
  elif model_name == 'STAR':
    div[1:] = gamma / (1 + np.exp(alpha + beta * y[:-1]))

  x_fit = mu + (delta + div[1:]) * x[:-1]
  x_fit = np.insert(x_fit, 0, mu / (1 - delta)) # Add 1st observation as␣
↪unconditional mean

  return x_fit
```

[ ]:
```
# Results for Q3

theta_init = (0, 0.3, 0, 0, 0)
date = df_log_diff['DATE']
x = df_log_diff['UNRATE']
y = df_log_diff['INDPRO']

recessions_periods = [
    ('1973-11-01', '1975-03-31'),
    ('1980-01-01', '1980-07-31'),
    ('2007-12-01', '2009-06-30'),
    ('2020-01-01', '2022-05-31')
]

results_Q3 = {}

for model in model_names:

  loss_init = get_avg_total_loss(theta_init, model, x, y)
  print(f'Initial average loss: {loss_init:.6f}')

  theta_optim, loss_optim = estimate_model(theta_init, model, x, y)

  total_loss= get_total_loss(theta_optim, model, x, y)
  print(f'Total loss: {total_loss:.6f}')


  results_Q3[model] = {
      'theta': theta_optim,
      'optimize_loss': loss_optim,
      'total_loss':total_loss
  }
```

```
    x_fitted = fit_values(theta_optim, model, x, y)

    # Plotting actual vs fitted values
    plt.figure(figsize=(10, 5))
    plt.plot(date, x, label='Observed', color='black')
    plt.plot(date, x_fitted, label=f'Fitted {model}', color='blue')

    for start, end in recessions_periods:
        plt.axvspan(pd.to_datetime(start), pd.to_datetime(end), color='red', alpha=0.3)

    plt.xlabel('Date')
    plt.ylabel('Log-Diff. of UNRATE')
    plt.legend()

    # Uncomment to store plot
    plt.savefig(f'UNRATE_fitted_{model}.png', bbox_inches='tight', dpi=300)
    plt.show()

    plt.close()

theta_hat_star = np.asarray(results_Q3["STAR"]["theta"])
```

[ ]:
```
for model, data in results_Q3.items():
    print(f"\nModel: {model}")
    print("  Theta: [" + ", ".join(f"{val:.4f}" for val in data['theta']) + "]")
    print(f"  Optimized Loss: {data['optimize_loss']:.4f}")
    print(f"  Total Loss: {data['total_loss']:.4f}")
```

3 Question 4

[ ]:
```
assert "STAR" in results_Q3
print("ˆ_STAR =", results_Q3["STAR"]["theta"])
```

[ ]:
```
assert len(theta_hat_star) == 5
date = pd.to_datetime(df_log_diff["DATE"])
```

[ ]:
```
# Results for Q4 (plot G_t and z_t with current ˆ_STAR)

# Parameters and data
mu, delta, gamma, alpha, beta = np.asarray(results_Q3["STAR"]["theta"])
date = pd.to_datetime(df_log_diff["DATE"]).to_numpy()
z    = df_log_diff["INDPRO"].to_numpy()    # Δlog(INDPRO_t)

# Logistic function
```

```python
def logistic(u):
    u = np.clip(u, -30, 30)
    return 1.0 / (1.0 + np.exp(u))

# G_t (time-varying AR coefficient)
G          = delta + gamma * logistic(alpha + beta * z[:-1])
z_align    = z[1:]
date_align = date[1:]

# Color palette
COL_G      = "#d62728"
COL_Z      = "#1f77b4"
COL_COVID  = "black"    # E64A19 soft orange
COL_GRID   = "#a6a6a6"

# Figure
fig, ax = plt.subplots(figsize=(12, 5))
ax.set_facecolor("#f8f8f8")

# Plot the two lines
ln1, = ax.plot(date_align, G, color=COL_G, lw=1.4, ls="--",
               label=r"Time-varying AR $(G_t)$")
ln2, = ax.plot(date_align, z_align, color=COL_Z, lw=1.2, ls="-",
               label=r"Industrial Production Growth␣
↪$\Delta\log(\mathrm{INDPRO}_t)$")

# Axis styling
ax.set_ylabel("Value")
ax.axhline(0, lw=0.8, color=COL_GRID, alpha=0.5)
ax.grid(True, which="major", linestyle="-", linewidth=0.6, color=COL_GRID,␣
↪alpha=0.35)

# Auto-adjust y-limits
ymin = min(z_align.min(), G.min())
ymax = max(z_align.max(), G.max())
pad  = 0.03 * (ymax - ymin if ymax > ymin else 1.0)
ax.set_ylim(ymin - 0.5*pad, ymax + 1.0*pad)

ax.xaxis.set_major_locator(mdates.YearLocator(base=5))
ax.xaxis.set_major_formatter(mdates.DateFormatter("%Y"))

# COVID-19
covid_start, covid_end = pd.Timestamp("2020-01-01"), pd.Timestamp("2022-05-31")
ax.axvline(covid_start, color=COL_COVID, ls="--", lw=1)
ax.axvline(covid_end,   color=COL_COVID, ls="--", lw=1)
covid_legend = mlines.Line2D([], [], color=COL_COVID, ls="--", lw=1,
                             label="COVID-19 period")
```

```
# Legend and title
ax.legend(handles=[ln1, ln2, covid_legend],
          loc="lower left", framealpha=0.9,
          facecolor="white", edgecolor=COL_GRID)
ax.set_title("Time-varying AR Parameter vs Industrial Production Growth (STAR␣
↪model)")

plt.tight_layout()
plt.savefig("Q4_STAR_G_vs_INDPRO.png",
            dpi=300,
            bbox_inches="tight",
            facecolor="white",
            pad_inches=0.05)
plt.show()
```

[ ]:
```
print("^_STAR =", results_Q3["STAR"]["theta"])
print("Lengths -> G:", len(G), " z_align:", len(z_align), " date_align:",␣
↪len(date_align))
print("G_t stats -> min:", np.nanmin(G), " max:", np.nanmax(G), " mean:", np.
↪nanmean(G))

# logistic (0,1)
lo, hi = (delta + min(0, gamma), delta + max(0, gamma))
print("Theoretical range approx:", (lo, hi))

# sign(dG/dz) = sign(- )
print("sign(dG/dz) =", np.sign(-gamma * beta))
```

4 Question 5

[ ]:
```
# Results for Q5 (SESTAR residuals, ACF (lags 1-12), and Ljung-Box test)

# 1) Parameters and series
theta_sestar = np.asarray(results_Q3["SESTAR"]["theta"])
mu, delta, gamma, alpha, beta = theta_sestar
date = pd.to_datetime(df_log_diff["DATE"])
x = df_log_diff["UNRATE"].to_numpy()

# 2) Fitted values and residuals for SESTAR
def logistic(u):
    u = np.clip(u, -30, 30)
    return 1.0 / (1.0 + np.exp(u))

g = gamma * logistic(alpha + beta * x[:-1])
x_hat = mu + (delta + g) * x[:-1]
```

```python
eps = np.empty_like(x, dtype=float)
eps[:] = np.nan
eps[1:] = x[1:] - x_hat                        # residuals aligned to t>=2

# 3) Sample ACF up to lag 12
eps_valid = eps[1:] - np.nanmean(eps[1:])      # demean, drop first NaN
T = len(eps_valid)

def sample_acf(series, max_lag):
    denom = np.sum(series**2)
    acfs = []
    for h in range(1, max_lag+1):
        num = np.sum(series[h:] * series[:-h])
        acfs.append(num / denom)
    return np.array(acfs)

max_lag = 12
rho = sample_acf(eps_valid, max_lag=max_lag)
ci = 1.96 / np.sqrt(T)  # ~95% reference band for white noise

# 4) Plot ACF bars (single, main figure)
max_lag = 12
acf_vals = acf(eps_valid, nlags=max_lag, fft=False)[1:]  # drop lag 0
lags = np.arange(1, max_lag + 1)

fig, ax = plt.subplots(figsize=(8, 4))
plot_acf(eps_valid, lags=max_lag, ax=ax, fft=False, zero=False)
ylim = 1.8 * np.nanmax(np.abs(acf_vals))
ax.set_xticks(lags)
ax.set_xticklabels([str(lag) for lag in lags])
ax.set_ylim(-min(1, ylim), min(1, ylim))
ax.set_xlabel("Lag")
ax.set_ylabel(r"ACF of $\hat{\varepsilon}_t$")
ax.set_title("SESTAR residual ACF (lags 1-12)")
ax.grid(alpha=0.2)
plt.tight_layout()
plt.savefig(f'Q5_ACF_SESTAR.png', bbox_inches='tight', dpi=300)
plt.show()


print("Residual ACF values (lags 1-12):")
for h, r in zip(lags, acf_vals):
    print(f"lag {h:2d}: {r:+.4f}")

# 5) Ljung-Box test up to lag 12
if acorr_ljungbox is not None:
```

```
    lb = acorr_ljungbox(eps_valid, lags=[12], return_df=True)  # columns:␣
↪lb_stat, lb_pvalue
    print(lb)
else:
    print("statsmodels not available -> skipped Ljung-Box test.")

# 6) Print ACF values and CI
print("\nResidual ACF (lags 1..12):")
for h, r in zip(lags, rho):
    print(f"lag {h:2d}: {r:+.3f}")
print(f"\nApprox. 95% reference band: ±{ci:.3f}  (T={T})")

plt.show()
```

5 Question 6

```
[ ]: # Robust std errors
x = df_log_diff['UNRATE'].to_numpy()
y = df_log_diff['INDPRO'].to_numpy()



def q_t(theta, model, x, y=None):
    mu, delta, gamma, alpha, beta = theta
    fitted = fit_values(theta, model, x, y)
    resid = x[1:] - fitted[1:]
    return resid**2

# Newey-West estimator
def nw(grad, p=12):
    T= grad.shape[0]
    Sigma0 = grad.T @ grad / T #lag 0
    Sigma = Sigma0.copy()
    for j in range(1, p+1):
        coef = 1 - j/(p+1)
        Sigmaj = (grad[j:].T @ grad[:-j]) / T #lag j
        Sigma += coef * (Sigmaj + Sigmaj.T)
    return Sigma

def grad(theta, model, x, y=None):
    q_fun = lambda param: q_t(param, model, x, y)
    return nd.Jacobian(q_fun)(theta)

def hessian_mean(theta, model, x, y=None):
    q_mean = lambda param: q_t(param, model, x, y).mean()
    return nd.Hessian(q_mean)(theta)   # (k × k)
```

```
# Results for all the models
robust_se = []

for model in results_Q3.keys():
    theta_hat = results_Q3[model]['theta']
    q = q_t(theta_hat, model, x, y)
    g = grad(theta_hat, model, x, y)
    H = hessian_mean(theta_hat, model, x, y)
    if model == 'AR(1)':
        g = g[:, :2]
        H = H[:2, :2]
    Omega = np.linalg.inv(H)
    Sigma = nw(g)
    T = g.shape[0]
    print(T)
    var = (Omega @ Sigma @ Omega.T)/T
    se = np.sqrt(np.diag(var))
    robust_se.append((model, theta_hat, se, results_Q3[model]['total_loss']))
```

[ ]:

```
# R squared
def r2_stats(model, theta):
    x_fit = fit_values(theta, model, x, y)
    R2 = r2_score(x, x_fit)
    T = len(x)
    k = 2 if model == 'AR(1)' else 5
    R2_adj = 1 - (1 - R2) * (T - 1) / (T - k - 1)
    return R2, R2_adj
```

[ ]:

```
# Latex

robust_dict = {m: {"theta": th, "se": se, "loss": loss} for (m, th, se, loss)␣
 ↪in robust_se}

def pad_params(model, theta, se):
    if model == 'AR(1)':
        theta = np.array([theta[0], theta[1], np.nan, np.nan, np.nan],␣
 ↪dtype=float)
        se    = np.array([se[0],    se[1],    np.nan, np.nan, np.nan],␣
 ↪dtype=float)
    return theta, se

def fnum(v, digs=2):
    return "" if v is None or np.isnan(v) else f"{v:.{digs}f}"

def fse(v, digs=2):
    return "" if v is None or np.isnan(v) else f"({v:.{digs}f})"
```

```python
col_titles   = ['AR', 'SESTAR', 'STAR']
param_labels = [r'$\mu$', r'$\delta$', r'$\gamma$', r'$\alpha$', r'$\beta$']

table_data = {}
for m in results_Q3.keys():
    th, se = robust_dict[m]["theta"], robust_dict[m]["se"]
    th, se = pad_params(m, th, se)
    R2, R2_adj = r2_stats(m, th)
    table_data[m] = {
        "theta": th,
        "se": se,
        "loss": robust_dict[m]["loss"],
        "R2": R2,
        "R2_adj": R2_adj
    }


lines = []
lines.append(r"\begin{table}[htbp]")
lines.append(r"\centering")
lines.append(r"\caption{Estimated parameters with the robust standard errors}")
lines.append(r"\begin{tabular}{lccc}")
lines.append(r"\toprule")
lines.append(r" & " + " & ".join(col_titles) + r" \\")
lines.append(r"\midrule")

for j, lab in enumerate(param_labels):
    est_row = [lab]
    se_row  = [""]
    for m in results_Q3.keys():
        est_row.append(fnum(table_data[m]["theta"][j], 2))
        se_row.append(fse(table_data[m]["se"][j], 2))
    lines.append(" & ".join(est_row) + r" \\")
    lines.append(" & ".join(se_row)  + r" \\")

lines.append(r"\addlinespace")
stat_labels = [r"\emph{Loss}", r"$R^2$", r"$R^2_{\text{adj}}$"]
for stat in stat_labels:
    row = [stat]
    for m in results_Q3.keys():
        if stat == r"\emph{Loss}":
            row.append(fnum(table_data[m]["loss"], 2))
        elif stat == r"$R^2$":
            row.append(fnum(table_data[m]["R2"], 2))
        else:
            row.append(fnum(table_data[m]["R2_adj"], 2))
    lines.append(" & ".join(row) + r" \\")
```

```
lines.append(r"\bottomrule")
lines.append(r"\end{tabular}")
lines.append(r"\end{table}")

tex_str = "\n".join(lines)
print(tex_str)

with open("table1.tex", "w", encoding="utf-8") as f:
    f.write(tex_str)
```

6 Question 7

[ ]:
```
GAMMA_IDX = 2  # (mu, delta, gamma, alpha, beta)

def two_sided_p_from_t(t):
    # Normal approx p-value using erf
    return 2.0 * (1.0 - 0.5 * (1.0 + math.erf(abs(t) / math.sqrt(2.0))))

rows = []
for model in ["SESTAR", "STAR"]:
    theta = table_data[model]["theta"]
    sevec = table_data[model]["se"]
    g_est = float(theta[GAMMA_IDX])
    g_se  = float(sevec[GAMMA_IDX])
    tval  = g_est / g_se
    pval  = two_sided_p_from_t(tval)
    rows.append({
        "Model": model,
        "Parameter": "gamma",
        "Estimate": g_est,
        "SE (robust)": g_se,
        "t-stat": tval,
        "p-value": pval
    })

Q7_wald = pd.DataFrame(rows)


print("Q7 - Wald test (two-sided) for non-linear time-dependence (  = 0)\n")
print(f"{'Model':<8} {'Param':<6} {'Est':>12} {'SE':>12} {'t':>10} {'p':>10}")
for _, r in Q7_wald.iterrows():
    est = fnum(r['Estimate'], 2)
    se  = fse(r['SE (robust)'], 2)
    t   = fnum(r['t-stat'], 3)
    p   = "<1e-16" if (isinstance(r['p-value'], float) and r['p-value'] <
 1e-16) else f"{r['p-value']:.3g}"
```

```
    print(f"{r['Model']:<8} {r['Parameter']:<6} {est:>12} {se:>12} {t:>10} {p:
↪>10}")
```