HW4

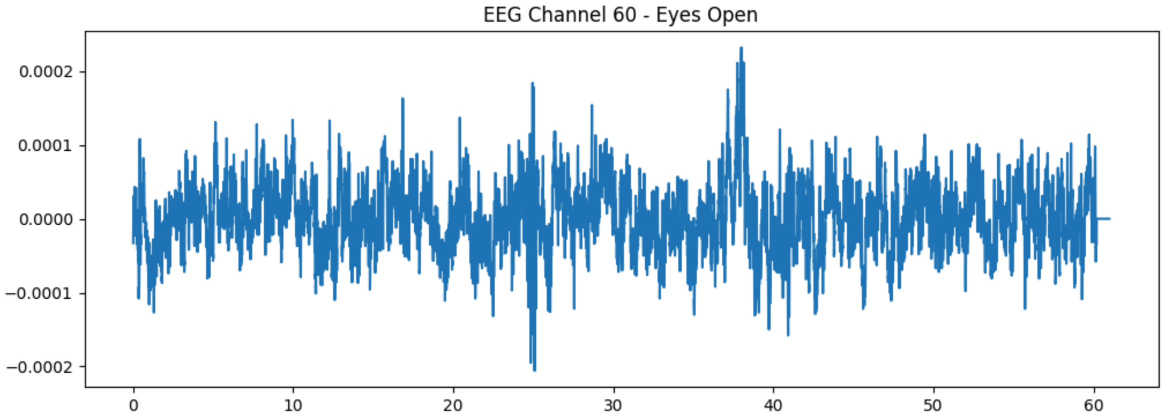
Problem 1

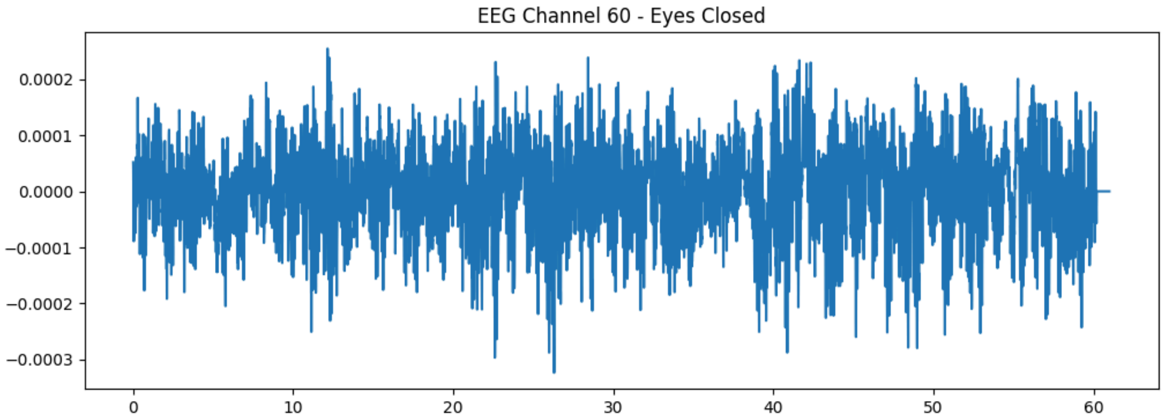
Part a

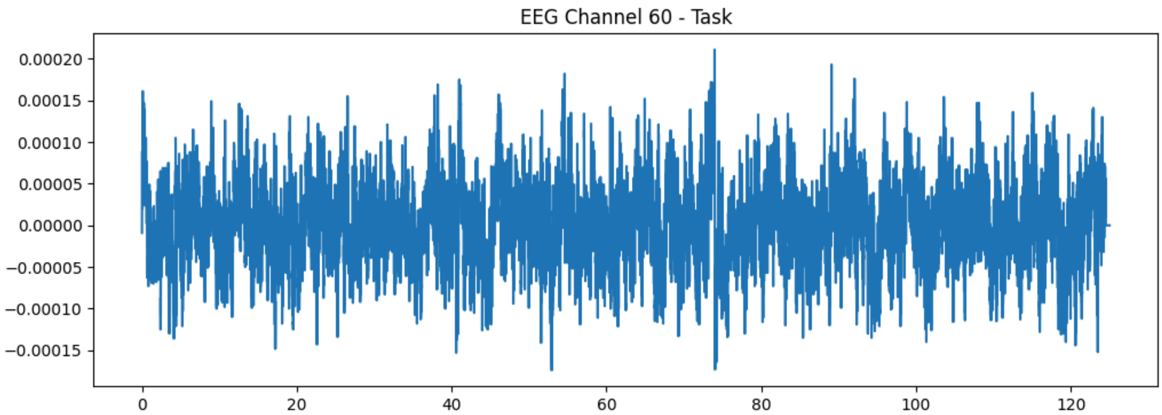
In [76]:

```
import mne
import matplotlib.pyplot as plt

raw1 = mne.io.read_raw_edf('S001R01.edf', preload=True)  # Eyes Open
raw2 = mne.io.read_raw_edf('S001R02.edf', preload=True)  # Eyes Closed
raw3 = mne.io.read_raw_edf('S001R05.edf', preload=True)  # Task

data1, times1 = raw1[:]
data2, times2 = raw2[:]
data3, times3 = raw3[:]

y1 = data1[59, :].squeeze()
y2 = data2[59, :].squeeze()
y3 = data3[59, :].squeeze()

plt.figure(figsize=(12, 4))
plt.plot(times1, y1)
plt.title('EEG Channel 60 - Eyes Open')
plt.show()

plt.figure(figsize=(12, 4))
plt.plot(times2, y2)
plt.title('EEG Channel 60 - Eyes Closed')
plt.show()

plt.figure(figsize=(12, 4))
plt.plot(times3, y3)
plt.title('EEG Channel 60 - Task')
plt.show()
```

```
Extracting EDF parameters from c:\Users\dkkdk\Documents\grad\248\HW4\S001R01.ed
f...
EDF file detected
Setting channel info structure...
Creating raw.info structure...
Reading 0 ... 9759  =      0.000 ...    60.994 secs...
Extracting EDF parameters from c:\Users\dkkdk\Documents\grad\248\HW4\S001R02.ed
f...
EDF file detected
Setting channel info structure...
Creating raw.info structure...
Reading 0 ... 9759  =      0.000 ...    60.994 secs...
Extracting EDF parameters from c:\Users\dkkdk\Documents\grad\248\HW4\S001R05.ed
f...
EDF file detected
Setting channel info structure...
Creating raw.info structure...
Reading 0 ... 19999  =      0.000 ...   124.994 secs...
```







The three EEG time series exhibit noticeable differences in amplitude and variability:

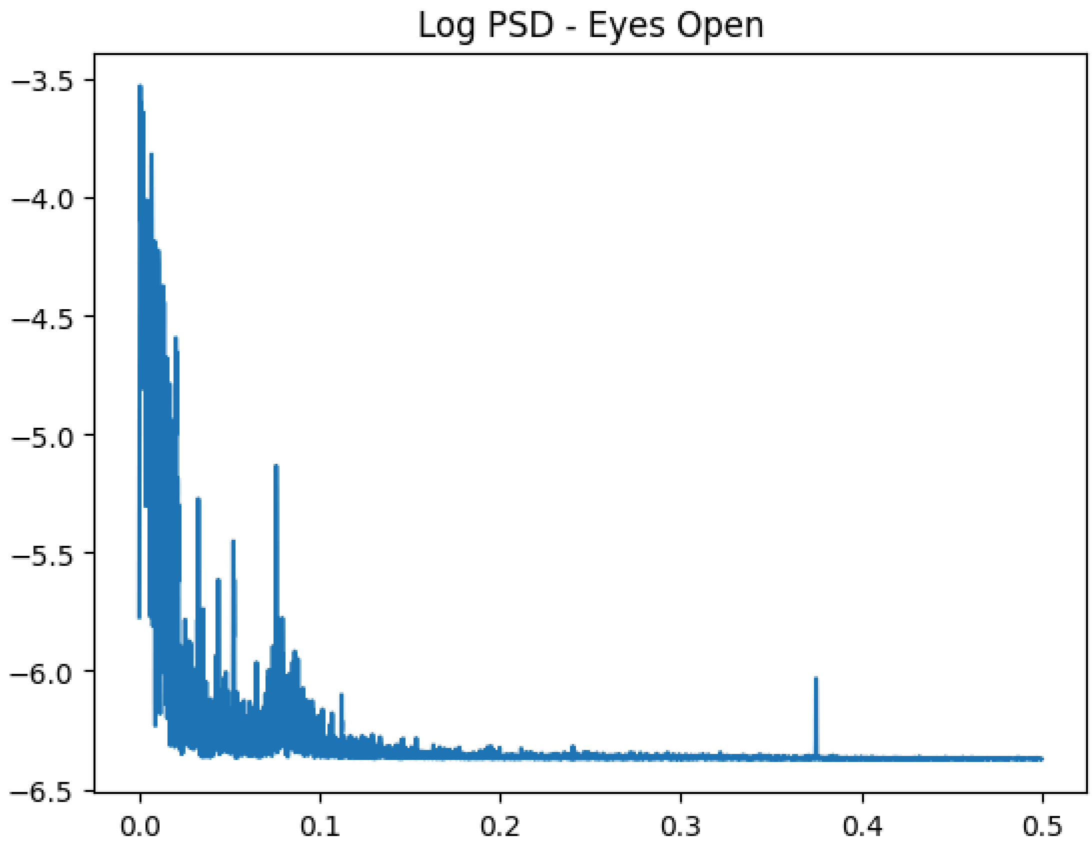
- Eyes Open: The signal appears relatively stable with lower variance and fewer extreme fluctuations.
- Eyes Closed: The amplitude variability increases, with more frequent and pronounced spikes, possibly indicating stronger alpha wave activity.
- Task: The signal is more irregular but has smaller amplitude overall, suggesting increased cognitive engagement and possible suppression of dominant rhythms.

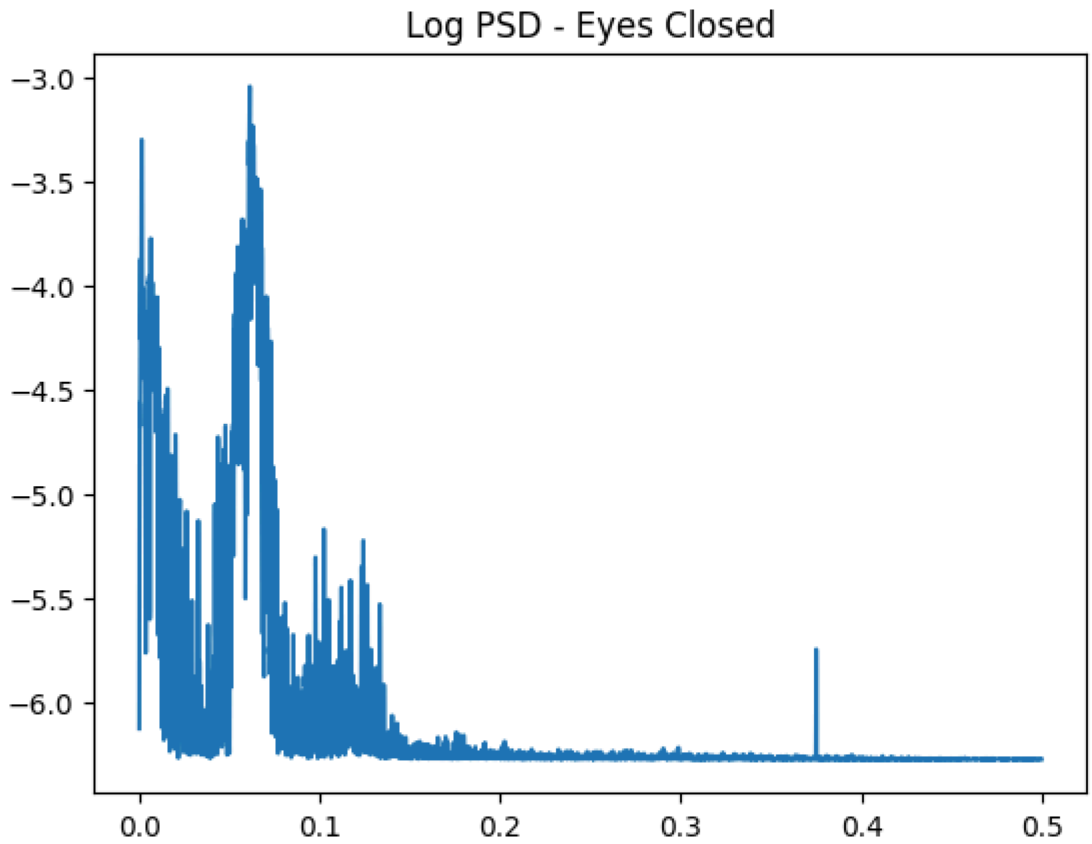
These visual differences suggest changing brain states across conditions.

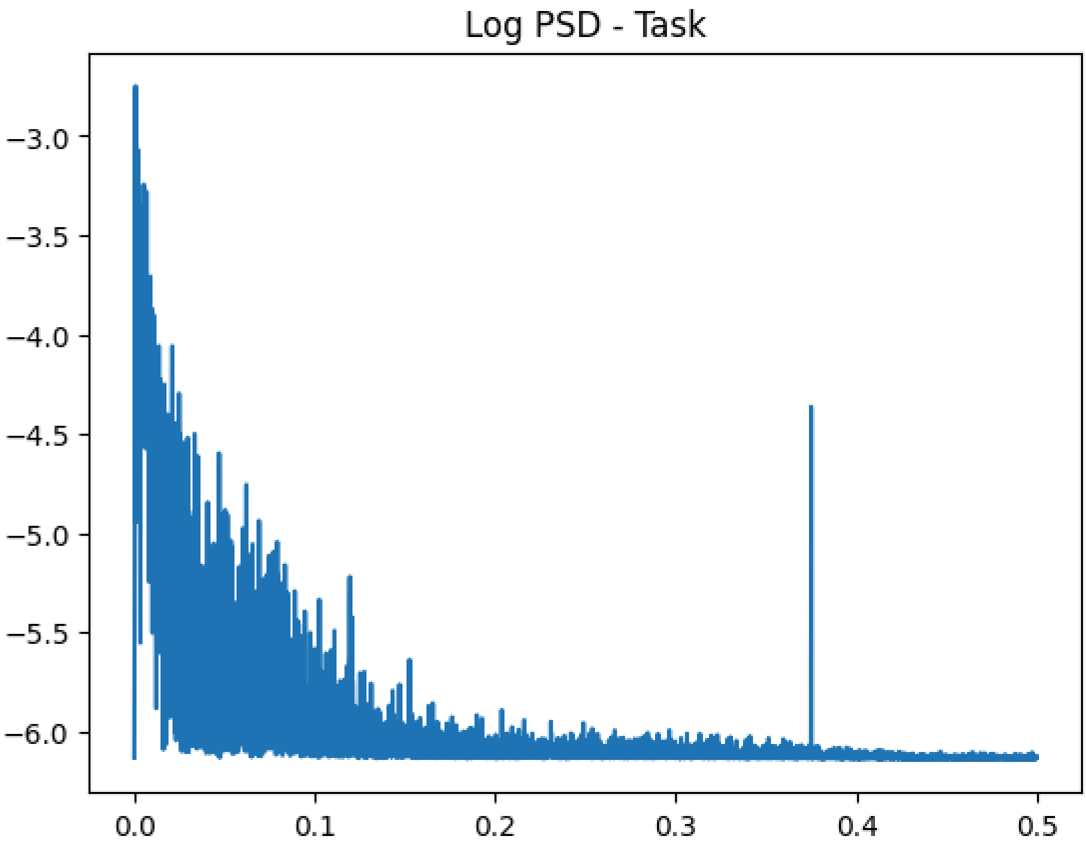
Part b

In [77]:
```
import numpy as np
import cvxpy as cp

def periodogram(y):
    fft_y = np.fft.fft(y)
    n = len(y)
    fourier_freqs = np.arange(1/n, 1/2, 1/n)
    m = len(fourier_freqs)
    pgram_y = (np.abs(fft_y[1:m+1]) ** 2)/n
    return fourier_freqs, pgram_y

def spectrum_estimator_ridge(y, lambda_val):
    freq, I = periodogram(y)
    m = len(freq)
    n = len(y)
    alpha = cp.Variable(m)
    neg_likelihood_term = cp.sum(cp.multiply((n * I / 2), cp.exp(-2 * alpha)) +
    smoothness_penalty = cp.sum(cp.square(alpha[2:] - 2 * alpha[1:-1] + alpha[:-
    objective = cp.Minimize(neg_likelihood_term + lambda_val * smoothness_penalt
    problem = cp.Problem(objective)
    problem.solve()
    return alpha.value, freq

lambda_val = 1

alpha1, freq1 = spectrum_estimator_ridge(y1, lambda_val)
alpha2, freq2 = spectrum_estimator_ridge(y2, lambda_val)
alpha3, freq3 = spectrum_estimator_ridge(y3, lambda_val)

plt.plot(freq1, alpha1)
plt.title('Log PSD - Eyes Open')
plt.show()

plt.plot(freq2, alpha2)
plt.title('Log PSD - Eyes Closed')
plt.show()

plt.plot(freq3, alpha3)
plt.title('Log PSD - Task')
plt.show()
```

```
c:\Users\dkkdk\AppData\Local\Programs\Python\Python310\lib\site-packages\cvxpy\pr
oblems\problem.py:1481: UserWarning: Solution may be inaccurate. Try another solv
er, adjusting the solver settings, or solve with verbose=True for more informatio
n.
  warnings.warn(
```


Log PSD - Eyes Open
−3.5
−4.0
−4.5
−5.0
−5.5
−6.0
−6.5
0.0
0.1
0.2
0.3
0.4
0.5


Log PSD - Eyes Closed
−3.0
−3.5
−4.0
−4.5
−5.0
−5.5
−6.0
0.0
0.1
0.2
0.3
0.4
0.5



We estimated the power spectral density (PSD) using two methods:

1. Periodogram – a basic Fourier-based method, which is noisy and not smooth.
2. Regularized PSD – we estimated the log-spectrum by minimizing a penalized likelihood, using a smoothness penalty on the second differences (ridge regularization).

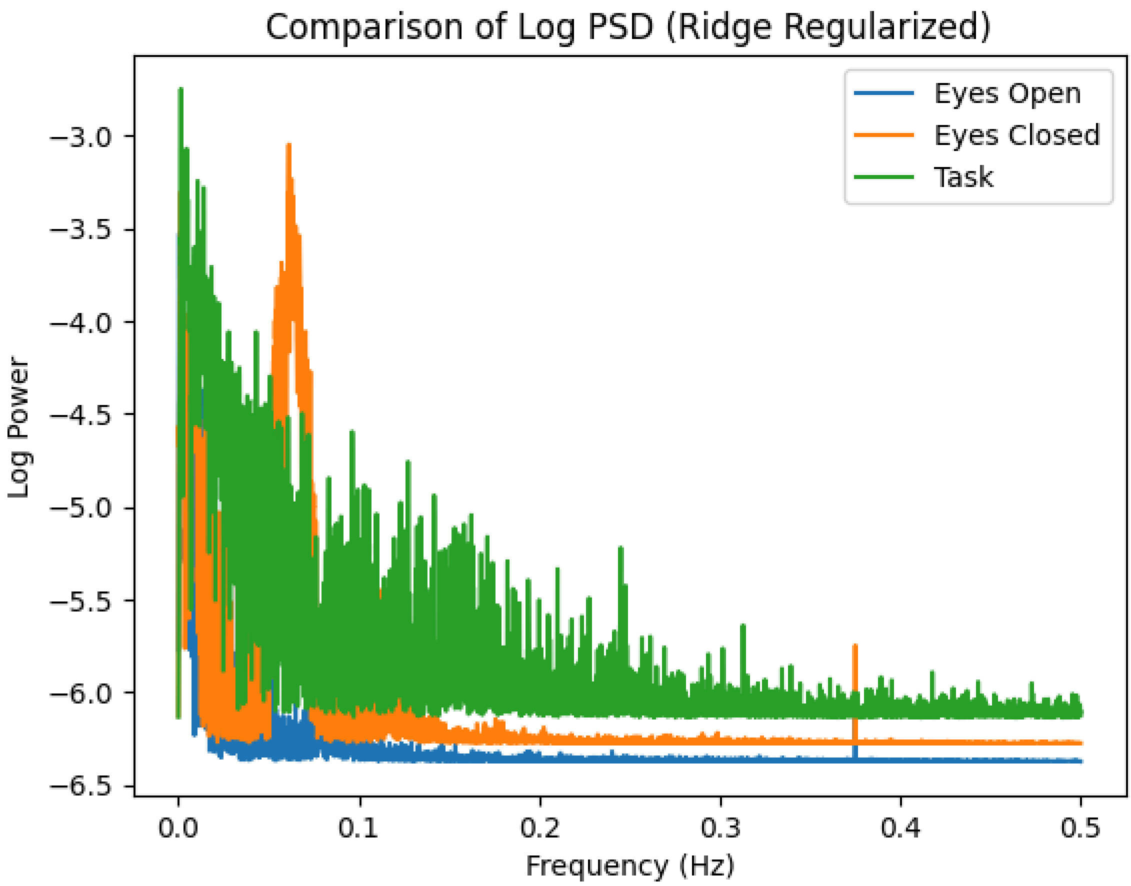
We chose the tuning parameter $\lambda = 1$ to balance smoothness and detail.

Compared to the periodogram, the regularized PSD is smoother and reveals clearer peaks—especially around 0.08 for eyes closed, which is consistent with alpha waves.

Part c

In [78]:
```
min_len = min(len(freq1), len(freq2), len(freq3))

f = freq1[:min_len]
a1 = alpha1[:min_len]
a2 = alpha2[:min_len]
a3 = alpha3[:min_len]

plt.plot(f, a1, label='Eyes Open')
plt.plot(f, a2, label='Eyes Closed')
plt.plot(f, a3, label='Task')
plt.legend()
plt.title('Comparison of Log PSD (Ridge Regularized)')
plt.xlabel('Frequency (Hz)')
```

```
plt.ylabel('Log Power')
plt.show()
```



The eyes closed condition shows a strong peak around 0.08 Hz, indicating strong alpha activity. The eyes open condition has lower power across all frequencies and no clear peaks. The task condition shows higher overall power and broader spectral content, suggesting more brain activity during the task.

Problem 2

Part a

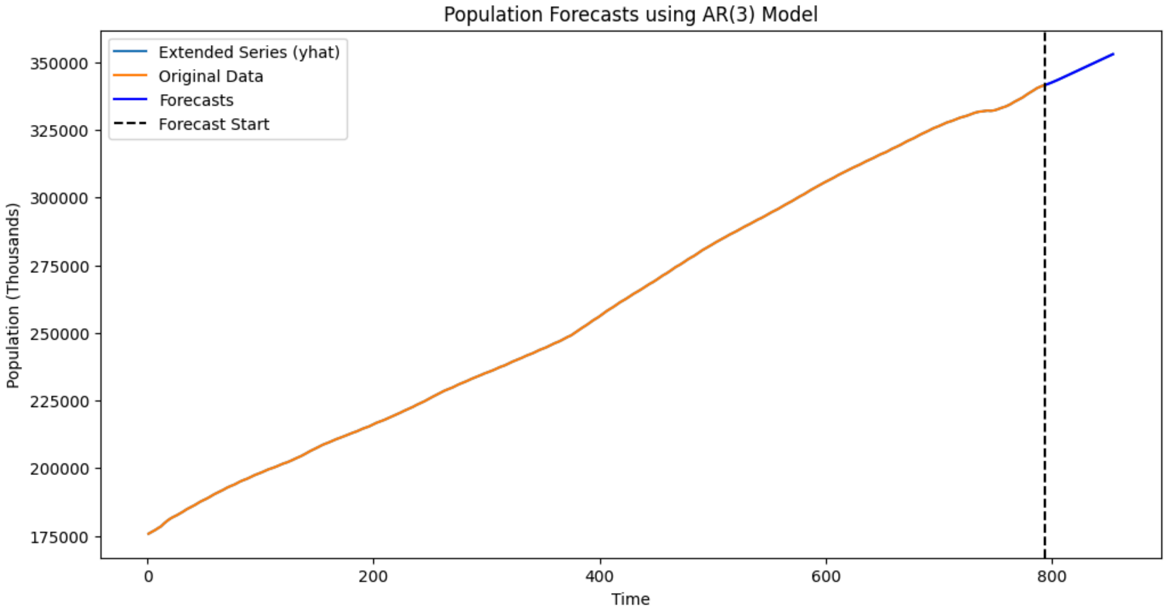
In [79]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm

data = pd.read_csv('POPTHM.csv')
y = data['POPTHM'].to_numpy()
p = 3
n = len(y)
yreg = y[p:]
Xmat = np.ones((n-p,1))
for j in range(1,p+1):
    col = y[p-j:n-j].reshape(-1,1)
    Xmat = np.column_stack([Xmat,col])

armod = sm.OLS(yreg, Xmat).fit()
print(armod.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       1.000
Model:                            OLS   Adj. R-squared:                  1.000
Method:                 Least Squares   F-statistic:                 2.517e+09
Date:                Fri, 11 Apr 2025   Prob (F-statistic):               0.00
Time:                        22:49:43   Log-Likelihood:                -3304.9
No. Observations:                 791   AIC:                             6618.
Df Residuals:                     787   BIC:                             6636.
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         19.4668      3.935      4.948      0.000      11.743      27.190
x1             2.3427      0.032     72.688      0.000       2.279       2.406
x2            -1.7704      0.064    -27.871      0.000      -1.895      -1.646
x3             0.4277      0.032     13.254      0.000       0.364       0.491
==============================================================================
Omnibus:                      527.570   Durbin-Watson:                   1.934
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            41844.850
Skew:                           2.212   Prob(JB):                         0.00
Kurtosis:                      38.356   Cond. No.                     3.20e+06
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.2e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Part b

In [80]:

```python
import matplotlib.pyplot as plt

k = 60
yhat = np.concatenate([y, np.full(k, -9999)])
for i in range(1, k + 1):
    ans = armod.params[0]
    for j in range(1, p + 1):
        ans += armod.params[j] * yhat[n + i - j - 1]
    yhat[n + i - 1] = ans
predvalues = yhat[n:]

plt.figure(figsize=(12, 6))
time_all = np.arange(1, n + k + 1)
plt.plot(time_all, yhat, label='Extended Series (yhat)', color='C0')
plt.plot(range(1, n + 1), y, label='Original Data', color='C1')
plt.plot(range(n + 1, n + k + 1), predvalues, label='Forecasts', color='blue')
plt.axvline(x=n, color='black', linestyle='--', label='Forecast Start')
plt.xlabel('Time')
plt.ylabel('Population (Thousands)')
plt.title('Population Forecasts using AR(3) Model')
plt.legend()
plt.show()
```



Yes, the predictions make intuitive sense. The forecasted population continues the overall upward trend seen in the historical data, reflecting the long-term growth of the U.S. population

Part c

In [81]:

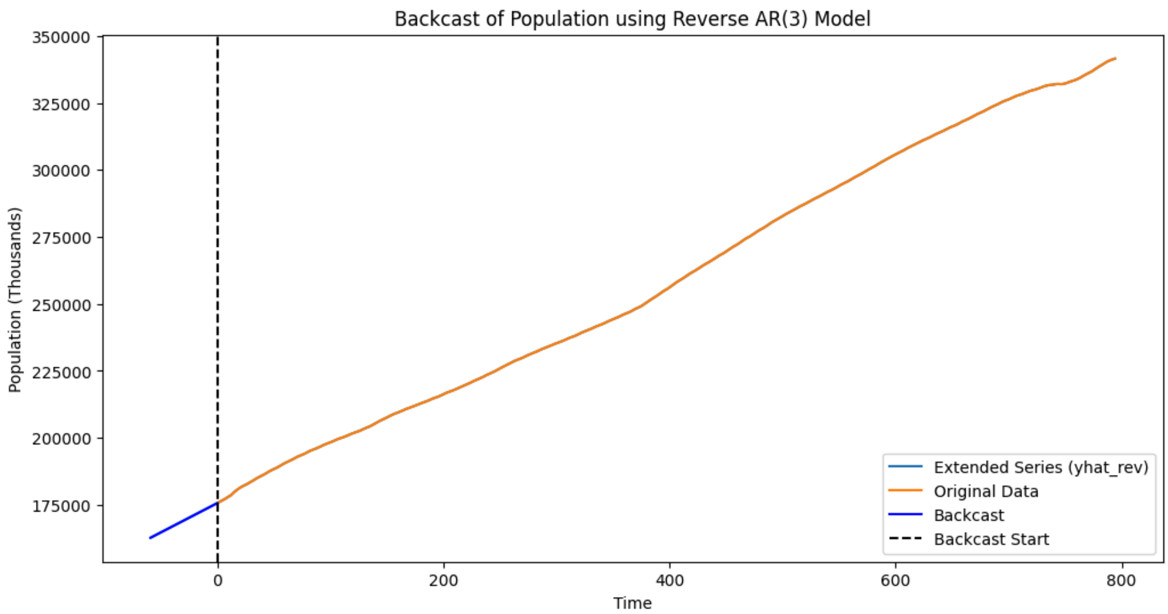
```
yreg_rev = y[:n - p]
Xmat_rev = np.ones((n - p, 1))
for j in range(1, p + 1):
    col = y[j : n - p + j].reshape(-1, 1)
    Xmat_rev = np.column_stack([Xmat_rev, col])

armod_rev = sm.OLS(yreg_rev, Xmat_rev).fit()
print(armod_rev.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       1.000
Model:                            OLS   Adj. R-squared:                  1.000
Method:                 Least Squares   F-statistic:                 2.524e+09
Date:                Fri, 11 Apr 2025   Prob (F-statistic):               0.00
Time:                        22:49:44   Log-Likelihood:                -3304.0
No. Observations:                 791   AIC:                             6616.
Df Residuals:                     787   BIC:                             6635.
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        -18.8330      3.934     -4.787      0.000     -26.556     -11.110
x1             2.3406      0.032     72.624      0.000       2.277       2.404
x2            -1.7673      0.063    -27.852      0.000      -1.892      -1.643
x3             0.4267      0.032     13.254      0.000       0.364       0.490
==============================================================================
Omnibus:                      512.824   Durbin-Watson:                   1.933
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            27139.367
Skew:                           2.240   Prob(JB):                         0.00
Kurtosis:                      31.344   Cond. No.                     3.20e+06
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.2e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Only the intercept differs significantly (19.47 vs. –18.83), which is expected due to the reversal of time.

Part d

In [82]:
```python
k = 60
yhat_rev = np.concatenate([np.full(k, -9999), y])
for i in range(k - 1, -1, -1):
    ans = armod_rev.params[0]
    for j in range(1, p + 1):
        ans += armod_rev.params[j] * yhat_rev[i + j]
    yhat_rev[i] = ans
predvalues_rev = yhat_rev[:k]

plt.figure(figsize=(12, 6))
time_all_rev = np.arange(-k + 1, n + 1)
plt.plot(time_all_rev, yhat_rev, label='Extended Series (yhat_rev)', color='C0')
plt.plot(range(1, n + 1), y, label='Original Data', color='C1')
plt.plot(range(-k + 1, 1), predvalues_rev, label='Backcast', color='blue')
plt.axvline(x=0, color='black', linestyle='--', label='Backcast Start')
plt.xlabel('Time')
plt.ylabel('Population (Thousands)')
plt.title('Backcast of Population using Reverse AR(3) Model')
plt.legend()
plt.show()
```



Yes, the backcasted values appear to make intuitive sense. The predictions smoothly extend the historical trend backward and align well with the early part of the observed population data.

Problem 3

Part a

In [83]:
```
import numpy as np
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt

df = pd.read_csv("MRTSSM4453USN.csv")
df["DATE"] = pd.to_datetime(df["observation_date"])
df = df.dropna()
y = df["MRTSSM4453USN"].to_numpy()
dates = df["DATE"]

k = 36
y_train, y_test = y[:-k], y[-k:]
dates_train, dates_test = dates[:-k], dates[-k:]

results = []
for p in range(1, 25):
    yreg = y_train[p:]
    Xmat = np.ones((len(yreg), 1))
    for j in range(1, p + 1):
        col = y_train[p - j : -j].reshape(-1, 1)
        Xmat = np.column_stack([Xmat, col])

    model = sm.OLS(yreg, Xmat).fit()

    yhat = np.concatenate([y_train, np.full(k, -9999.0)])
    for i in range(1, k + 1):
        pred = model.params[0]
        for j in range(1, p + 1):
            pred += model.params[j] * yhat[len(y_train) + i - j - 1]
```

```
        yhat[len(y_train) + i - 1] = pred

    mse = np.mean((yhat[-k:] - y_test) ** 2)
    results.append((p, mse))

best_p, best_mse = min(results, key=lambda x: x[1])
print(f"Best AR order by test MSE: p = {best_p}, MSE = {best_mse:.2f}")
```

```
Best AR order by test MSE: p = 15, MSE = 68704.17
```

Problem 4

Part a

In [84]:

```
import pandas as pd
import numpy as np
import statsmodels.api as sm

df = pd.read_csv("PCESV.csv")
y = df["PCESV"].to_numpy()
n_total = len(y)
n_test = 24

y_train = y[:n_total - n_test]
y_test = y[n_total - n_test:]

results = []
for p in range(1, 11):
    yreg = y_train[p:]
    Xmat = np.ones((len(y_train) - p, 1))
    for j in range(1, p + 1):
        col = y_train[p - j : -j].reshape(-1, 1)
        Xmat = np.column_stack([Xmat, col])
    model = sm.OLS(yreg, Xmat).fit()
    results.append((p, model.aic, model))

best_p, best_aic, best_model = min(results, key=lambda x: x[1])

print(f"Best AR order selected by AIC: p = {best_p}")
print(best_model.summary())
```

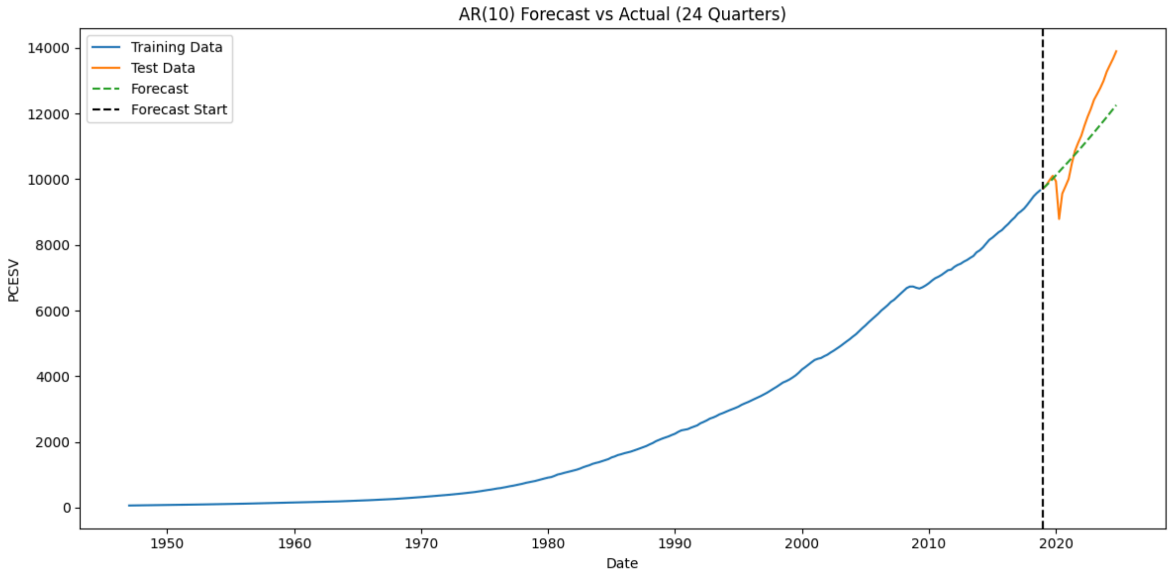
```
Best AR order selected by AIC: p = 10
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       1.000
Model:                            OLS   Adj. R-squared:                  1.000
Method:                 Least Squares   F-statistic:                 1.514e+06
Date:                Fri, 11 Apr 2025   Prob (F-statistic):               0.00
Time:                        22:49:44   Log-Likelihood:                -1082.1
No. Observations:                 278   AIC:                             2186.
Df Residuals:                     267   BIC:                             2226.
Df Model:                          10
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          2.1211      1.121      1.891      0.060      -0.087       4.329
x1             1.6480      0.061     27.094      0.000       1.528       1.768
x2            -0.5240      0.114     -4.591      0.000      -0.749      -0.299
x3            -0.1510      0.114     -1.327      0.186      -0.375       0.073
x4            -0.0699      0.114     -0.613      0.540      -0.294       0.155
x5             0.0059      0.113      0.052      0.958      -0.216       0.228
x6             0.2603      0.113      2.296      0.022       0.037       0.484
x7             0.0755      0.114      0.660      0.510      -0.150       0.301
x8            -0.5654      0.117     -4.823      0.000      -0.796      -0.335
x9             0.4978      0.120      4.143      0.000       0.261       0.734
x10           -0.1754      0.064     -2.720      0.007      -0.302      -0.048
==============================================================================
Omnibus:                       85.822   Durbin-Watson:                   2.001
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              426.263
Skew:                          -1.162   Prob(JB):                     2.74e-93
Kurtosis:                       8.604   Cond. No.                     1.77e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 1.77e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

I choose the p which has the lowest AIC, which is p=10

Part b

In [85]:
```
import matplotlib.pyplot as plt

k = 24
yhat = np.concatenate([y_train, np.full(k, -9999.0)])

for i in range(1, k + 1):
    ans = best_model.params[0]
    for j in range(1, best_p + 1):
        ans += best_model.params[j] * yhat[len(y_train) + i - j - 1]
    yhat[len(y_train) + i - 1] = ans

predvalues = yhat[-k:]

df["observation_date"] = pd.to_datetime(df["observation_date"])
dates = df["observation_date"]
dates_train = dates[:n_total - k]
```

```
dates_test = dates[n_total - k:]

plt.figure(figsize=(12, 6))
plt.plot(dates_train, y_train, label="Training Data", color="C0")
plt.plot(dates_test, y_test, label="Test Data", color="C1")
plt.plot(dates_test, predvalues, label="Forecast", color="C2", linestyle="--")
plt.axvline(x=dates_test.iloc[0], color="black", linestyle="--", label="Forecast
plt.xlabel("Date")
plt.ylabel("PCESV")
plt.title(f"AR({best_p}) Forecast vs Actual (24 Quarters)")
plt.legend()
plt.tight_layout()
plt.show()
```



There is a slight underestimation in the later periods where the actual PCE accelerates more sharply. Despite this, the predictions are reasonably accurate and reflect the long-term growth behavior of the series.

Part c

In [86]:
```
import numpy as np
import statsmodels.api as sm

log_y = np.log(df["PCESV"].to_numpy())
log_y_train = log_y[:n_total - n_test]
log_y_test = log_y[n_total - n_test:]

p_log = 4
yreg_log = log_y_train[p_log:]
Xmat_log = np.ones((len(log_y_train) - p_log, 1))
for j in range(1, p_log + 1):
    col = log_y_train[p_log - j : -j].reshape(-1, 1)
    Xmat_log = np.column_stack([Xmat_log, col])

log_model = sm.OLS(yreg_log, Xmat_log).fit()

print(log_model.summary())
```

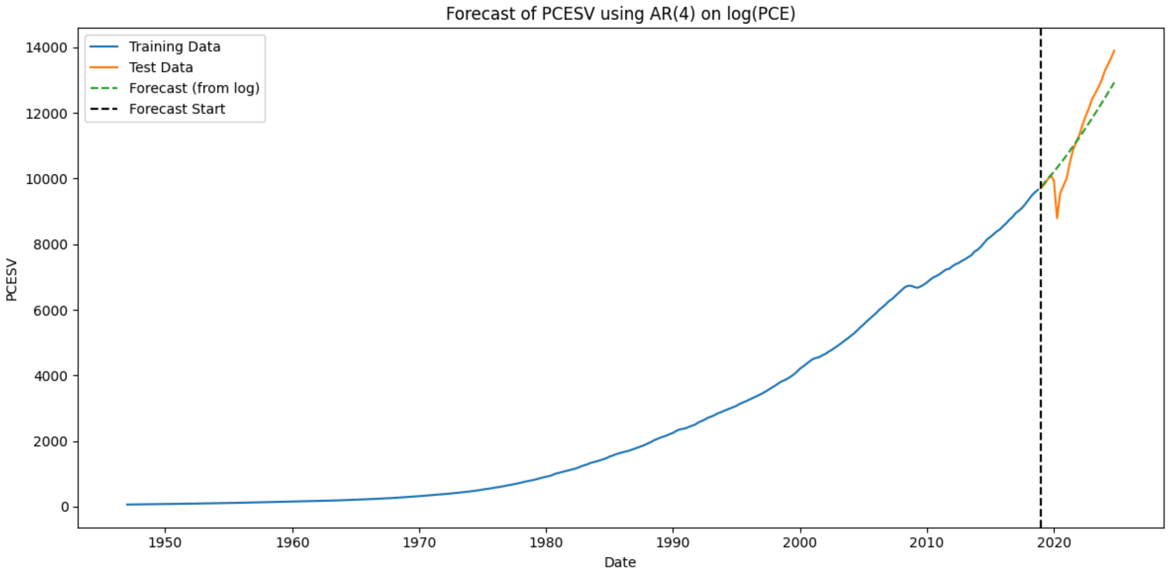
```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       1.000
Model:                            OLS   Adj. R-squared:                  1.000
Method:                 Least Squares   F-statistic:                 6.830e+06
Date:                Fri, 11 Apr 2025   Prob (F-statistic):               0.00
Time:                        22:49:45   Log-Likelihood:                 1098.0
No. Observations:                 284   AIC:                            -2186.
Df Residuals:                     279   BIC:                            -2168.
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0070      0.002      3.487      0.001       0.003       0.011
x1             1.3889      0.059     23.453      0.000       1.272       1.505
x2            -0.1513      0.102     -1.485      0.139      -0.352       0.049
x3            -0.0937      0.102     -0.920      0.358      -0.294       0.107
x4            -0.1444      0.059     -2.441      0.015      -0.261      -0.028
==============================================================================
Omnibus:                       18.967   Durbin-Watson:                   2.006
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               55.845
Skew:                           0.139   Prob(JB):                     7.47e-13
Kurtosis:                       5.154   Cond. No.                     6.27e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 6.27e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Part d

In [87]:

```python
import matplotlib.pyplot as plt

k = 24
log_yhat = np.concatenate([log_y_train, np.full(k, -9999.0)])

for i in range(1, k + 1):
    ans = log_model.params[0]
    for j in range(1, p_log + 1):
        ans += log_model.params[j] * log_yhat[len(log_y_train) + i - j - 1]
    log_yhat[len(log_y_train) + i - 1] = ans

pred_pcesv = np.exp(log_yhat[-k:])

true_pcesv_test = np.exp(log_y_test)

plt.figure(figsize=(12, 6))
plt.plot(dates_train, np.exp(log_y_train), label="Training Data", color="C0")
plt.plot(dates_test, true_pcesv_test, label="Test Data", color="C1")
plt.plot(dates_test, pred_pcesv, label="Forecast (from log)", color="C2", linest
plt.axvline(x=dates_test.iloc[0], color="black", linestyle="--", label="Forecast
plt.xlabel("Date")
plt.ylabel("PCESV")
plt.title("Forecast of PCESV using AR(4) on log(PCE)")
plt.legend()
```

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



The AR(4) model fitted on log-transformed data produces forecasts that closely follow the long-term growth trend of the test data. The predicted values generally lie near the actual observations and capture the upward trajectory well. However, some of the short-term volatility—such as the dip around 2020—is not reflected in the forecast.

Part e

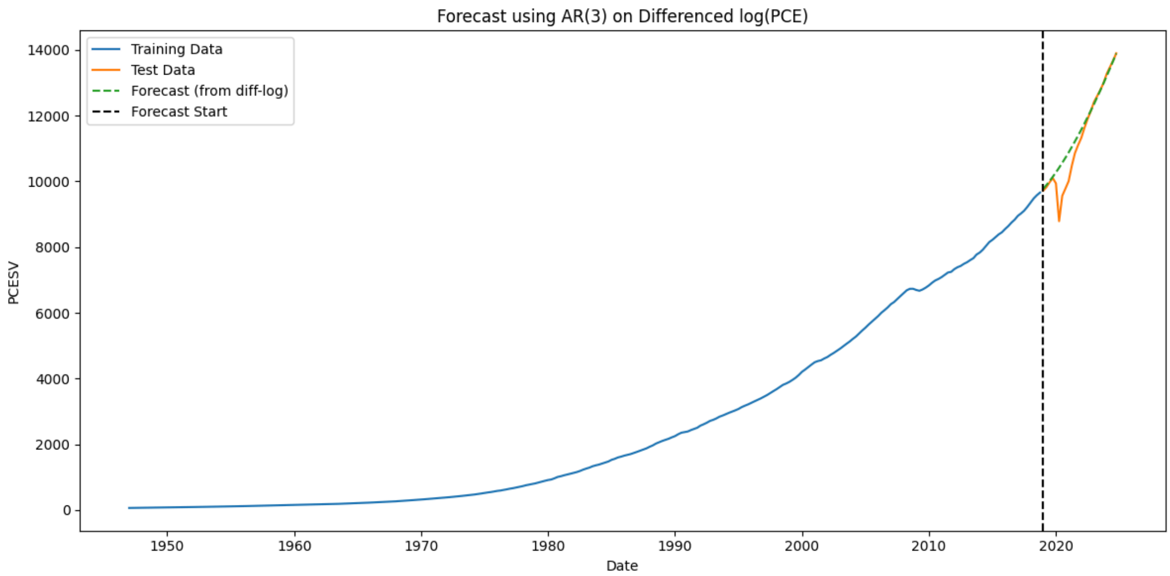
In [88]:
```
import numpy as np
import statsmodels.api as sm

yt = np.diff(log_y_train)
p = 3

yreg = yt[p:]
Xmat = np.ones((len(yt) - p, 1))
for j in range(1, p + 1):
    col = yt[p - j : -j].reshape(-1, 1)
    Xmat = np.column_stack([Xmat, col])

model_diff = sm.OLS(yreg, Xmat).fit()
phi0, phi1, phi2, phi3 = model_diff.params

alpha1 = phi0 + 1 - phi1
alpha2 = phi1 - phi2
alpha3 = phi2 - phi3
alpha4 = phi3

print("AR(4) coefficients for log(PCE_t) implied by differenced model:")
print(f"alpha1 (lag 1): {alpha1:.4f}")
print(f"alpha2 (lag 2): {alpha2:.4f}")
print(f"alpha3 (lag 3): {alpha3:.4f}")
print(f"alpha4 (lag 4): {alpha4:.4f}")
```

```
AR(4) coefficients for log(PCE_t) implied by differenced model:
alpha1 (lag 1): 0.5978
alpha2 (lag 2): 0.1543
alpha3 (lag 3): 0.0928
alpha4 (lag 4): 0.1583
```

The magnitudes and signs differ substantially. The direct model in (c) places most weight on lag 1, with some negative contributions from longer lags. In contrast, the differenced-log-implied model in (e) spreads out smaller positive weights more evenly across all four lags.

Part f

In [89]:
```
import matplotlib.pyplot as plt

k = 24
yt_hat = np.concatenate([yt, np.full(k, -9999.0)])
log_last = log_y_train[-1]

for i in range(1, k + 1):
    pred = model_diff.params[0]
    for j in range(1, 4):
        pred += model_diff.params[j] * yt_hat[len(yt) + i - j - 1]
    yt_hat[len(yt) + i - 1] = pred

log_forecast = np.zeros(k)
log_forecast[0] = log_last + yt_hat[len(yt)]
for i in range(1, k):
    log_forecast[i] = log_forecast[i - 1] + yt_hat[len(yt) + i]

forecast_pcesv = np.exp(log_forecast)
true_pcesv = np.exp(log_y_test)

plt.figure(figsize=(12, 6))
plt.plot(dates_train, np.exp(log_y_train), label="Training Data", color="C0")
plt.plot(dates_test, true_pcesv, label="Test Data", color="C1")
plt.plot(dates_test, forecast_pcesv, label="Forecast (from diff-log)", color="C2
plt.axvline(x=dates_test.iloc[0], color="black", linestyle="--", label="Forecast
plt.xlabel("Date")
plt.ylabel("PCESV")
plt.title("Forecast using AR(3) on Differenced log(PCE)")
plt.legend()
plt.tight_layout()
plt.show()
```



The AR(3) model fitted to the differenced log data performs quite well in forecasting future values of PCE. The forecasted trajectory closely matches the actual test data and successfully captures both the level and the upward trend. Unlike the raw log model in part (d), this model responds more effectively to sharp increases in the post-2020 period, including the recovery from the pandemic dip.

Part g

In [90]:

```
import numpy as np
import statsmodels.api as sm

log_y_diff2 = log_y_train[2:] - 2 * log_y_train[1:-1] + log_y_train[:-2]

p = 2
yreg = log_y_diff2[p:]
Xmat = np.ones((len(log_y_diff2) - p, 1))
for j in range(1, p + 1):
    col = log_y_diff2[p - j : -j].reshape(-1, 1)
    Xmat = np.column_stack([Xmat, col])

model_diff2 = sm.OLS(yreg, Xmat).fit()
theta0, theta1, theta2 = model_diff2.params

beta1 = 2 + theta0 - theta1
beta2 = -5 + 2*theta1 - theta2
beta3 = 4 - theta1 + theta2
beta4 = -1

print("AR(4) coefficients for log(PCE_t) implied by second-differenced model:")
print(f"beta1 (lag 1): {beta1:.4f}")
print(f"beta2 (lag 2): {beta2:.4f}")
print(f"beta3 (lag 3): {beta3:.4f}")
print(f"beta4 (lag 4): {beta4:.4f}")
```

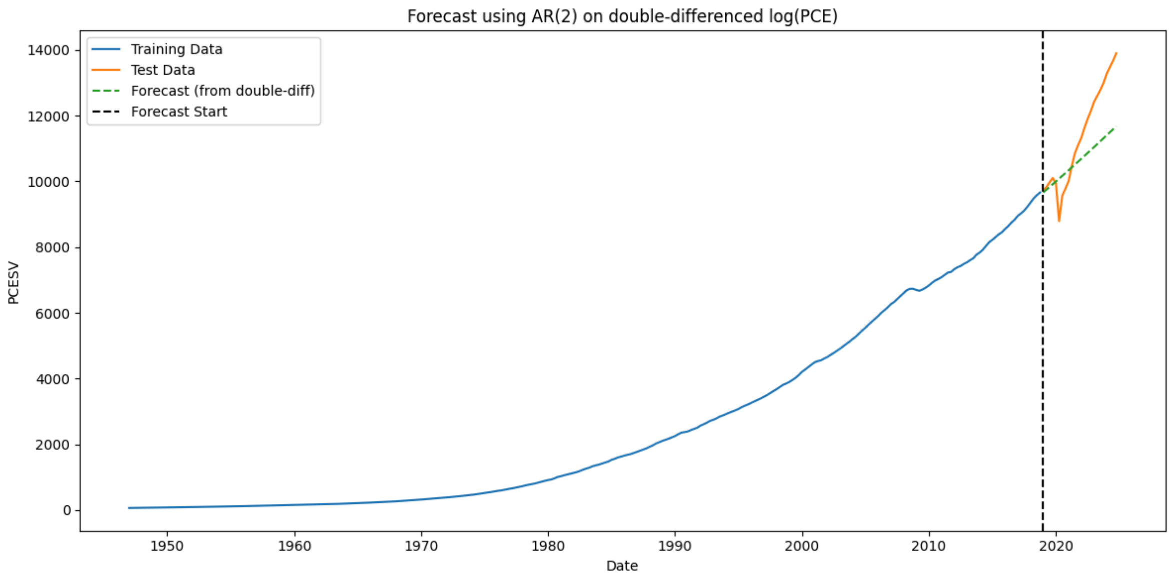
```
AR(4) coefficients for log(PCE_t) implied by second-differenced model:
beta1 (lag 1): 2.5305
beta2 (lag 2): -5.8379
beta3 (lag 3): 4.3074
beta4 (lag 4): -1.0000
```

Compared to parts (c) and (e), the coefficients in part (g) are much more extreme, especially for lag 2 and 3. The large magnitude and alternating signs indicate a more oscillatory structure. This is likely a result of over-differencing, which can introduce artificial dynamics.

Part h

In [91]:

```python
import matplotlib.pyplot as plt

log_prev2 = log_y_train[-2]
log_prev1 = log_y_train[-1]

log_forecast = np.zeros(24)
log_forecast[0] = log_prev1
log_forecast[1] = log_prev2 + 2*(log_prev1 - log_prev2)

diff2_preds = np.zeros(24)
for i in range(2, 24):
    pred = model_diff2.params[0]
    pred += model_diff2.params[1] * diff2_preds[i - 1]
    pred += model_diff2.params[2] * diff2_preds[i - 2]
    diff2_preds[i] = pred
    log_forecast[i] = 2 * log_forecast[i - 1] - log_forecast[i - 2] + pred

pcesv_forecast = np.exp(log_forecast)
pcesv_actual = np.exp(log_y_test)

plt.figure(figsize=(12, 6))
plt.plot(dates_train, np.exp(log_y_train), label="Training Data", color="C0")
plt.plot(dates_test, pcesv_actual, label="Test Data", color="C1")
plt.plot(dates_test, pcesv_forecast, label="Forecast (from double-diff)", color=
plt.axvline(x=dates_test.iloc[0], color="black", linestyle="--", label="Forecast
plt.xlabel("Date")
plt.ylabel("PCESV")
plt.title("Forecast using AR(2) on double-differenced log(PCE)")
plt.legend()
plt.tight_layout()
plt.show()
```



The AR(2) model fitted on double-differenced log data underestimates the actual test values by a substantial margin, especially in the later quarters. While the model captures the general upward trend, it lacks the flexibility to follow the sharp post-pandemic increase in PCE.

Part i

Among all the fitted models, the AR(3) model on the differenced log data (part f) provided the most accurate predictions for the 24 test observations. The predicted trajectory closely tracked the actual test data and successfully captured both the level and steep post-pandemic growth in PCE.

Problem 5

Part a

We are given the recurrence relation:

$$u_k - u_{k-1} + 0.5u_{k-2} = 0, \quad \text{for } k = 0, 1, 2, \ldots$$

This is a second-order linear homogeneous difference equation. The characteristic equation is:

$$r^2 - r + 0.5 = 0$$

Solving this gives the complex roots:

$$r = \frac{1 \pm i}{2} = \frac{1}{\sqrt{2}} e^{\pm i\pi/4}$$

Therefore, the general solution is:

$$u_k = C \cdot \left(\frac{1}{\sqrt{2}}\right)^k \cos\left(\frac{\pi k}{4} + \phi\right)$$

Or equivalently, using sine and cosine components:

$$u_k = 2^{-k/2}\left(a\cos\left(\frac{\pi k}{4}\right) + b\sin\left(\frac{\pi k}{4}\right)\right)$$

To match the form given in the problem:

$$u_k = 2^{-k/2}\left(u_0\cos\left(\frac{\pi k}{4}\right) + (2u_1 - u_0)\sin\left(\frac{\pi k}{4}\right)\right)$$

which satisfies the recurrence for any initial values (u_0) and (u_1).

Part b

We are given the AR(2) model:

$$y_t = 3 + y_{t-1} - 0.5y_{t-2} + \varepsilon_t$$

To simplify, define the deviation from steady state:

$$z_t = y_t - 6 \Rightarrow z_t = z_{t-1} - 0.5z_{t-2} + \varepsilon_t$$

We are asked to show the deterministic prediction (with no noise) has the form:

$$\hat{y}_{n+i} = 6 + 2^{-(i+1)/2}\left\{(y_{n-6})\cos\left(\frac{\pi(i+1)}{4}\right) + (2y_n - y_{n-6})\sin\left(\frac{\pi(i+1)}{4}\right)\right\}$$

This is exactly the homogeneous solution form from part (a), scaled and centered at 6. The terms (y_n) and (y_{n-6}) act as the initial values (u_1) and (u_0), respectively. Thus, the prediction formula holds by directly applying the solution form from part (a).

Part c

We are given the process:

$$y_t = 6 + \sum_{j=0}^{\infty} 2^{-j/2}\left(\cos\left(\frac{j\pi}{4}\right) + \sin\left(\frac{j\pi}{4}\right)\right)\varepsilon_{t-j}$$

Let the MA(∞) coefficients be:

$$\psi_j = 2^{-j/2}\left(\cos\left(\frac{j\pi}{4}\right) + \sin\left(\frac{j\pi}{4}\right)\right)$$

We want to verify that this series satisfies the AR(2) model:

$$y_t = 3 + y_{t-1} - 0.5y_{t-2} + \varepsilon_t$$

The summation form is a moving average representation of the **stationary solution** to the AR(2) model. Since the coefficients (ψ_j) are constructed using the same roots from the characteristic equation in part (a), the process indeed solves the AR(2) equation.

This can be checked by substituting the infinite sum into the AR(2) difference equation, confirming the relation holds term-by-term.

Problem 6

We are given:

$y_t = U_1 \cos(2\pi f_1 t) + V_1 \sin(2\pi f_1 t) + U_2 \cos(2\pi f_2 t) + V_2 \sin(2\pi f_2 t)$

where $(U_1, U_2, V_1, V_2 \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0, \sigma^2))$, and $(f_1 \neq f_2)$ are fixed real frequencies.

Part a

Since all random coefficients have mean zero:

$\mathbb{E}[y_t] = 0$

Part b

Using independence and the fact that $(\operatorname{Var}(aX) = a^2 \operatorname{Var}(X))$:

$\operatorname{Var}(y_t) = \sigma^2 \cos^2(2\pi f_1 t) + \sigma^2 \sin^2(2\pi f_1 t) + \sigma^2 \cos^2(2\pi f_2 t) + \sigma^2 \sin^2(2\pi f_2 t)$

$= \sigma^2(1 + 1) = 2\sigma^2$

Part c

We compute:

$\operatorname{Cov}(y_{t_1}, y_{t_2}) = \mathbb{E}[y_{t_1} y_{t_2}]$

Only terms involving the same random variable survive due to independence and zero-mean:

$ = \sigma^2 \cos(2\pi f_1 t_1) \cos(2\pi f_1 t_2)

- \sigma^2 \sin(2\pi f_1 t_1) \sin(2\pi f_1 t_2)
- \sigma^2 \cos(2\pi f_2 t_1) \cos(2\pi f_2 t_2)
- \sigma^2 \sin(2\pi f_2 t_1) \sin(2\pi f_2 t_2) $

Use the identity:

$\cos a \cos b + \sin a \sin b = \cos(a - b)$

$\Rightarrow \operatorname{Cov}(y_{t_1}, y_{t_2}) = \sigma^2 \cos(2\pi f_1 (t_1 - t_2)) + \sigma^2 \cos(2\pi f_2 (t_1 - t_2))$

Part d

Yes. The process has:

- Constant mean: $(\mathbb{E}[y_t] = 0)$

- Constant variance: $(\mathrm{Var}(y_t) = 2\sigma^2)$
- Autocovariance that depends only on the lag $(t_1 - t_2)$

Problem 7

Part a

In [92]:
```
import numpy as np
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt

df = pd.read_csv("PCEC.csv")
df["DATE"] = pd.to_datetime(df["observation_date"])
df.set_index("DATE", inplace=True)
df["logPCEC"] = np.log(df["PCEC"])

log_pcec = df["logPCEC"]
train = log_pcec.iloc[:-12].copy()
test = log_pcec.iloc[-12:]

y = train.values
y2 = y[2:]
X2 = np.column_stack([np.ones(len(y2)), y[1:-1], y[:-2]])

ols_model = sm.OLS(y2, X2).fit()
print(ols_model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       1.000
Model:                            OLS   Adj. R-squared:                  1.000
Method:                 Least Squares   F-statistic:                 2.017e+06
Date:                Fri, 11 Apr 2025   Prob (F-statistic):               0.00
Time:                        22:49:45   Log-Likelihood:                 883.39
No. Observations:                 298   AIC:                            -1761.
Df Residuals:                     295   BIC:                            -1750.
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0248      0.004      6.043      0.000       0.017       0.033
x1             1.0175      0.058     17.446      0.000       0.903       1.132
x2            -0.0187      0.058     -0.322      0.748      -0.133       0.096
==============================================================================
Omnibus:                      139.659   Durbin-Watson:                   2.005
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             8035.316
Skew:                          -1.046   Prob(JB):                         0.00
Kurtosis:                      28.353   Cond. No.                     1.24e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 1.24e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Part b

In [93]:
```
import scipy.stats as stats

beta_hat = ols_model.params
X = X2
y_target = y2
n, p = X.shape

resid = y_target - X @ beta_hat
RSS = np.sum(resid**2)

alpha_n = n / 2
beta_n = RSS / 2

N = 5000
theta_samples = []

XtX_inv = np.linalg.inv(X.T @ X)

for _ in range(N):
    sigma2_sample = stats.invgamma.rvs(a=alpha_n, scale=beta_n)

    cov_beta = sigma2_sample * XtX_inv
    beta_sample = np.random.multivariate_normal(mean=beta_hat, cov=cov_beta)

    theta_samples.append((*beta_sample, np.sqrt(sigma2_sample)))

theta_samples = np.array(theta_samples)  # shape: (N, 4)
```

Part c

In [94]:
```
forecast_samples = np.zeros((N, 12))

y_tm1 = train.iloc[-1]
y_tm2 = train.iloc[-2]

for i in range(N):
    phi_0, phi_1, phi_2, sigma = theta_samples[i]
    y1 = y_tm1
    y2 = y_tm2
    future_vals = []

    for _ in range(12):
        mean = phi_0 + phi_1 * y1 + phi_2 * y2
        y_new = np.random.normal(loc=mean, scale=sigma)

        future_vals.append(y_new)
        y2, y1 = y1, y_new

    forecast_samples[i, :] = future_vals
```

Part d

In [95]:
```
posterior_means = forecast_samples.mean(axis=0)
posterior_stds = forecast_samples.std(axis=0)
```

```
y_full = log_pcec.values
X_test = np.column_stack([
    np.ones(12),
    y_full[-13:-1],
    y_full[-14:-2]
])

frequentist_preds = ols_model.get_prediction(X_test)
frequentist_mean = frequentist_preds.predicted_mean
frequentist_std = frequentist_preds.se_mean

plt.figure(figsize=(12, 6))
x = np.arange(1, 13)

plt.plot(x, posterior_means, label="Bayesian Posterior Mean", color="blue")
plt.fill_between(x,
                 posterior_means - posterior_stds,
                 posterior_means + posterior_stds,
                 color="blue", alpha=0.2, label="Bayesian ±1 SD")

plt.plot(x, frequentist_mean, label="Frequentist Mean", color="orange")
plt.fill_between(x,
                 frequentist_mean - frequentist_std,
                 frequentist_mean + frequentist_std,
                 color="orange", alpha=0.2, label="Frequentist ±1 SE")

plt.title("12-step Ahead Forecast: Bayesian vs Frequentist")
plt.xlabel("Steps Ahead")
plt.ylabel("log(PCEC)")
plt.legend()
plt.tight_layout()
plt.show()
```