Term Project 3

In [20]:
```
import numpy as np
import math
from math import log
from scipy.optimize import minimize
import numpy as np
import pandas as pd
import pandas_datareader.data as web
from datetime import datetime
import matplotlib.pyplot as plt
```

In [21]:
```
# Set random seed for reproducibility
np.random.seed(1234)
T = 100  # length of time series
phi_true = 0.9                # true AR(1) coefficient
alpha_true = 1.5              # true measurement coefficient
sigma2_true = 0.1             # true process noise variance
tau2_true = 0.2               # true measurement noise variance

# Initialize arrays
x_true = np.zeros(T)
y_obs  = np.zeros(T)

# Initial state x_0
x_true[0] = np.random.normal(0, np.sqrt(sigma2_true/(1-phi_true**2)))
y_obs[0]  = alpha_true * x_true[0] + np.random.normal(0, np.sqrt(tau2_true))

# Simulate forward
for t in range(1, T):
    x_true[t] = phi_true * x_true[t-1] + np.random.normal(0, np.sqrt(sigma2_true
    y_obs[t]  = alpha_true * x_true[t] + np.random.normal(0, np.sqrt(tau2_true))

# Display first few data points
print("First 5 true states:", x_true[:5])
print("First 5 observations:", y_obs[:5])
```

```
First 5 true states: [0.34201501 0.76087524 0.45691755 0.68305152 0.61970999]
First 5 observations: [-0.019598    1.00149068  1.08212765  0.73991531 -0.0733942
1]
```

In [22]:
```
def particle_filter(y_observations, phi=0.9, alpha=1.5, sigma2=0.1, tau2=0.2, N=
    T = len(y_observations)
    # Initialize particles from prior (assume prior ~ N(0, steady-state variance
    if phi**2 < 1:
        init_var = sigma2 / (1 - phi**2)
    else:
        init_var = sigma2 * 10  # arbitrary large variance if non-stationary
    particles = np.random.normal(0, np.sqrt(init_var), size=N)
    weights = np.ones(N) / N  # start with equal weights

    state_estimates = np.zeros(T)
    # Update weights with first observation y0
    diff0 = y_observations[0] - alpha * particles       # innovation for each pa
    likelihoods = (1/np.sqrt(2*np.pi*tau2)) * np.exp(-0.5 * (diff0**2) / tau2)
    weights *= likelihoods
```

```python
    weights /= np.sum(weights)                          # normalize weights
    state_estimates[0] = np.sum(weights * particles)    # weighted mean estimate

    # Resample particles based on weights
    indices = np.random.choice(np.arange(N), size=N, p=weights)
    particles = particles[indices]
    weights.fill(1.0/N)  # reset weights to uniform after resampling

    # Iterate through time steps 1 to T-1
    for t in range(1, T):
        # Prediction: propagate each particle according to state model
        particles = phi * particles + np.random.normal(0, np.sqrt(sigma2), size=
        # Update: weight particles by likelihood of observation y[t]
        diff = y_observations[t] - alpha * particles
        likelihoods = (1/np.sqrt(2*np.pi*tau2)) * np.exp(-0.5 * (diff**2) / tau2
        weights *= likelihoods
        weights /= np.sum(weights)                      # normalize
        state_estimates[t] = np.sum(weights * particles)
        # Resample
        indices = np.random.choice(np.arange(N), size=N, p=weights)
        particles = particles[indices]
        weights.fill(1.0/N)
    return state_estimates

# Run particle filter on the simulated data
x_pf_est = particle_filter(y_obs, phi=phi_true, alpha=alpha_true, sigma2=sigma2_
```

In [23]:
```python
# Compute PF estimation error
pf_errors = x_pf_est - x_true
print("Particle Filter RMSE:", np.sqrt(np.mean(pf_errors**2)))
```

```
Particle Filter RMSE: 0.26306123758589933
```

In [24]:
```python
def kalman_filter(y_observations, phi=0.9, alpha=1.5, sigma2=0.1, tau2=0.2):
    """
    Kalman Filter for state estimation in a linear Gaussian model.
    Returns arrays of filtered state means and variances.
    """
    T = len(y_observations)
    x_est = np.zeros(T)
    P_est = np.zeros(T)
    # Initialize prior for x_0
    x_est[0] = 0.0
    P_est[0] = sigma2/(1-phi**2) if phi**2 < 1 else sigma2 * 10  # large prior v
    K0 = P_est[0] * alpha / (alpha**2 * P_est[0] + tau2)
    x_est[0] = x_est[0] + K0 * (y_observations[0] - alpha * x_est[0])
    P_est[0] = (1 - K0 * alpha) * P_est[0]
    # Iterate for t=1...T-1
    for t in range(1, T):
        # Prediction
        x_pred = phi * x_est[t-1]
        P_pred = phi**2 * P_est[t-1] + sigma2
        # Update
        Kt = P_pred * alpha / (alpha**2 * P_pred + tau2)
        x_est[t] = x_pred + Kt * (y_observations[t] - alpha * x_pred)
        P_est[t] = (1 - Kt * alpha) * P_pred
    return x_est, P_est

x_kf_est, P_kf_est = kalman_filter(y_obs, phi=phi_true, alpha=alpha_true, sigma2
print("Kalman Filter RMSE:", np.sqrt(np.mean((x_kf_est - x_true)**2)))
```

```
Kalman Filter RMSE: 0.2611220924878127
```

In [25]:
```
def neg_log_likelihood(params, observations):
    phi, alpha, log_sigma2, log_tau2 = params
    sigma2 = math.exp(log_sigma2)
    tau2   = math.exp(log_tau2)
    T = len(observations)
    # initialize
    x_mean = 0.0
    P_var = sigma2/(1-phi**2) if phi**2 < 1 else sigma2*100  # large prior var i
    logL = 0.0
    for t in range(T):
        # Prediction (for t=0, prediction is prior)
        # Compute innovation and its variance
        S = alpha**2 * P_var + tau2
        nu = observations[t] - alpha * x_mean
        # Update log-likelihood
        logL += -0.5*(math.log(2*math.pi*S) + (nu**2)/S)
        # Kalman update
        K = (P_var * alpha) / S
        x_mean = x_mean + K * nu
        P_var = (1 - K*alpha) * P_var
        # Predict next (except at last step)
        if t < T-1:
            x_mean = phi * x_mean
            P_var = phi**2 * P_var + sigma2
    return -logL  # return negative log-likelihood for min

# Initial guess for parameters
initial_guess = [0.5, 1.0, log(0.1), log(0.2)]
result = minimize(neg_log_likelihood, initial_guess, args=(y_obs,))
phi_est, alpha_est, log_sigma2_est, log_tau2_est = result.x
sigma2_est = math.exp(log_sigma2_est)
tau2_est   = math.exp(log_tau2_est)

print("Estimated phi =", round(phi_est, 3))
print("Estimated alpha =", round(alpha_est, 3))
print("Estimated sigma^2 =", round(sigma2_est, 3))
print("Estimated tau^2 =", round(tau2_est, 3))
```

```
Estimated phi = 0.959
Estimated alpha = 4.346
Estimated sigma^2 = 0.004
Estimated tau^2 = 0.298
```

In [26]:
```
# State estimation using Kalman smoother with estimated parameters
x_kf_est_MLE, P_kf_est_MLE = kalman_filter(y_obs, phi=phi_est, alpha=alpha_est,
# (We reuse the kalman_filter function; it returns the filtered estimates.)
# Now perform Rauch-Tung-Striebel smoothing on the filtered results:
def kalman_smoother(x_filt, P_filt, phi, alpha, sigma2, tau2):
    T = len(x_filt)
    x_smooth = np.copy(x_filt)
    # Backward smoothing
    for t in range(T-2, -1, -1):
        # Compute smoothing gain
        P_pred = phi**2 * P_filt[t] + sigma2    # prior var at t+1
        G = P_filt[t] * phi / P_pred
        # Smooth the state estimate
        x_smooth[t] += G * (x_smooth[t+1] - phi * x_filt[t])
    return x_smooth
```

```python
x_mle_smooth = kalman_smoother(x_kf_est_MLE, P_kf_est_MLE, phi=phi_est, alpha=al
print("MLE-based method RMSE:", np.sqrt(np.mean((x_mle_smooth - x_true)**2)))
```

MLE-based method RMSE: 0.5326021386548989

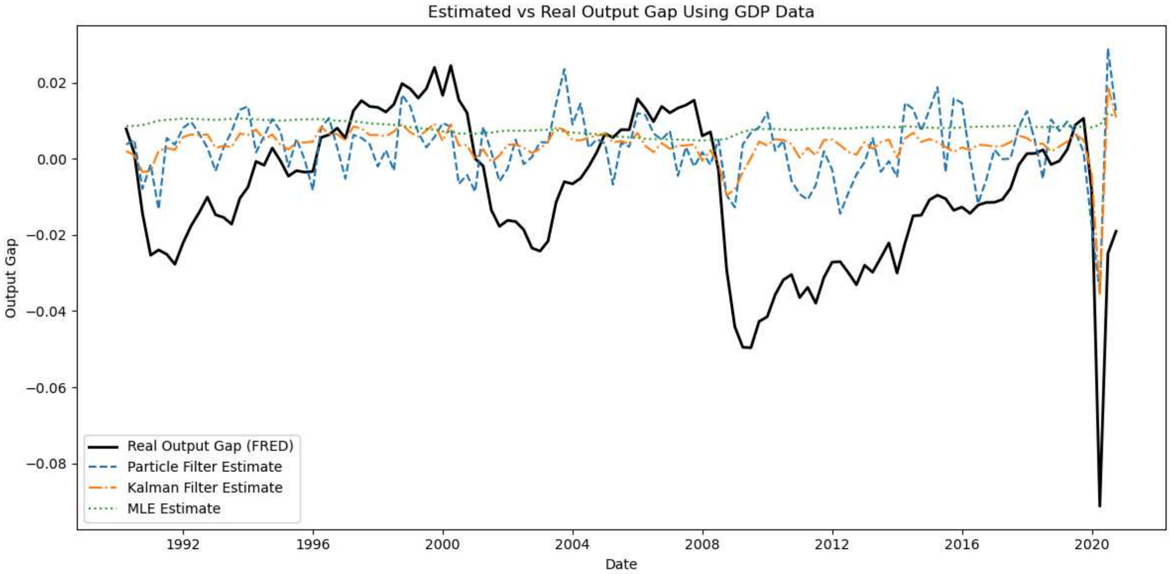
In [ ]:

Estimation of the GDP gap using real data

In [27]:

```python
start = datetime(1990, 1, 1)
end   = datetime(2020, 12, 31)

# Download Real GDP (GDPC1) and Potential GDP (GDPPOT) series
gdpc1  = web.DataReader('GDPC1', 'fred', start, end)
gdppot = web.DataReader('GDPPOT', 'fred', start, end)

# Merge and compute the real output gap (as percentage deviation)
df = pd.concat([gdpc1.rename(columns={'GDPC1':'RealGDP'}),
                gdppot.rename(columns={'GDPPOT':'PotentialGDP'})], axis=1).dropn
df['OutputGap'] = (df['RealGDP'] - df['PotentialGDP']) / df['PotentialGDP']

# Compute GDP growth (log differences)
df['GDP_Growth'] = np.log(df['RealGDP']).diff()
# Drop initial NaN from diff
df = df.dropna()

# Extract observation and "true" state arrays
y_real      = df['GDP_Growth'].values
nx          = len(y_real)
x_true_real = df['OutputGap'].values

# ## Estimate Latent Output Gap using Particle Filter and Kalman Filter
# (Assuming particle_filter and kalman_filter functions and true parameters are
x_pf_real = particle_filter(y_real,
                              phi=phi_true,
                              alpha=alpha_true,
                              sigma2=sigma2_true,
                              tau2=tau2_true,
                              N=1000)
x_kf_real, p_kf_est = kalman_filter(y_real,
                              phi=phi_true,
                              alpha=alpha_true,
                              sigma2=sigma2_true,
                              tau2=tau2_true)

x_mle_real = kalman_smoother(x_kf_real,
                             P_filt = p_kf_est,
                             phi=phi_est,
                             alpha=alpha_est,
                             sigma2=sigma2_est,
                             tau2=tau2_est)
# ## Plot Estimated vs Real Output Gap

plt.figure(figsize=(12, 6))
plt.plot(df.index, x_true_real, label='Real Output Gap (FRED)', color='black', l
plt.plot(df.index, x_pf_real,   label='Particle Filter Estimate', linestyle='--'
plt.plot(df.index, x_kf_real,   label='Kalman Filter Estimate', linestyle='-.')
plt.plot(df.index, x_mle_real,  label='MLE Estimate', linestyle=':')
```

```
plt.legend()
plt.xlabel('Date')
plt.ylabel('Output Gap')
plt.title('Estimated vs Real Output Gap Using GDP Data')
plt.tight_layout()
plt.show()
```



Results

The graph illustrates the estimated and real output gap using GDP data over the period from the early 1990s to around 2020. Three lines are presented:

1. Real Output Gap (FRED) - shown as a solid black line.
2. Particle Filter Estimate - shown as a blue dashed line.
3. Kalman Filter Estimate - shown as an orange dash-dot line.

The real output gap exhibits notable fluctuations, especially during significant economic downturns (e.g., the 2008 financial crisis and the COVID-19 pandemic in 2020). The particle and Kalman filter estimates generally track the trend of the real output gap, though with varying levels of volatility.

Findings

1. **Consistency of Estimates:** The Kalman filter estimate appears smoother and more consistent with long-term trends, while the particle filter estimate shows higher volatility, particularly in periods of economic uncertainty.
2. **Crisis Response:** During major economic shocks, the real output gap diverges significantly from both estimates, particularly during the 2008 financial crisis and the COVID-19 recession. The particle filter estimate shows more rapid and pronounced deviations compared to the Kalman filter.
3. **Post-Crisis Recovery:** Both filter estimates tend to realign with the real output gap after the shock periods, with the Kalman filter demonstrating a more gradual transition compared to the particle filter.

Discussion

The comparison between the real output gap and the estimated values highlights the trade-offs between filtering methods in terms of volatility and trend alignment. The Kalman filter's smoothness makes it more suitable for long-term economic analysis, while the particle filter's sensitivity may be beneficial for capturing rapid economic shifts. However, the particle filter's heightened volatility may lead to overreacting to short-term variations. Further analysis could involve evaluating the root mean square error (RMSE) of each method against the real gap to quantitatively assess accuracy.