Module 6 : Bayesian Structural Time Series (BSTS)

Learning Objectives

By the end of this module, you will be able to:

1. Understand state-space models and their components
2. Implement local level and local linear trend models
3. Add seasonal components (daily, weekly, annual) to time series
4. Build regression components with external predictors
5. Model time-varying coefficients (dynamic regression)
6. Apply BSTS to gold prices with USD and inflation data
7. Decompose commodity prices into trend, seasonality, and noise

Why This Matters for Trading

Static linear regression assumes relationships don't change. But commodity markets are dynami

- **Structural Breaks**: OPEC policy changes, new technology (fracking), regime shifts
- **Time-Varying Relationships**: Oil-USD correlation weakens during crises
- **Complex Seasonality**: Natural gas has annual (winter heating), weekly (storage reports), a patterns
- **Trend Changes**: Mean-reversion vs momentum regimes alternate

Bayesian Structural Time Series (BSTS) models decompose prices into interpretable componen

- **Trend**: Long-term direction (is gold in a secular bull market?)
- **Seasonality**: Predictable cycles (harvest pressure on agricultural commodities)
- **Regression**: Impact of fundamentals (how much does USD drive gold today?)
- **Noise**: Unpredictable shocks

This decomposition enables:

- **Better forecasts**: Separate signal from noise
- **Regime detection**: Identify when relationships break down
- **Event impact**: Measure how geopolitical shocks affect trends
- **Seasonality trading**: Exploit recurring patterns with confidence

In short, BSTS gives you X-ray vision into market structure.

In [ ]:
```
# Standard imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy import stats
import pymc as pm
```

```
import arviz as az
import warnings
warnings.filterwarnings('ignore')

np.random.seed(42)
plt.style.use('seaborn-v0_8-whitegrid')

print(f"PyMC version: {pm.__version__}")
print(f"ArviZ version: {az.__version__}")
```

1. Introduction to State-Space Models

Theory

A **state-space model** separates observed data from hidden "state" variables:

Observation Equation (what we see): $$ y_t = Z_t \alpha_t + \epsilon_t, \quad \epsilon_t \sim \text{Normal}(0 , \sigma_y^ 2) $$

State Equation (hidden dynamics): $$ \alpha_t = T_t \alpha_{t- 1 } + R_t \eta_t, \quad \eta_t \sir \text{Normal}(0 , \sigma_\alpha^ 2) $$

Where:

- y_t: Observed price at time t
- α_t: Hidden state (e.g., "true" price level)
- ϵ_t: Observation noise (measurement error, bid-ask bounce)
- η_t: State innovation (real price changes)

Trading Interpretation

- **State α_t**: "Fundamental" or "fair" value
- **Observation noise ϵ_t**: Market microstructure, liquidity shocks
- **State innovation η_t**: New information arrival

Trading strategy: When y_t deviates from α_t (state estimate), mean-revert.

2. Local Level Model (Random Walk + Noise)

Theory

The simplest state-space model:

$$ \begin{align} y_t &= \mu_t + \epsilon_t, \quad \epsilon_t \sim \text{Normal}(0 , \sigma_y^ 2) &= \mu_{t- 1 } + \eta_t, \quad \eta_t \sim \text{Normal}(0 , \sigma_\mu^ 2) \end{align} $$

- μ_t: Local level (latent "true price")
- Evolves as a random walk
- $\sigma_\mu^ 2 / \sigma_y^ 2 $ ratio determines smoothness

Signal-to-Noise Ratio

- High $\sigma_\mu^ 2 $: Level changes rapidly (trending)
- Low $\sigma_\mu^ 2 $: Level nearly constant (mean-reverting)
- $\sigma_\mu^ 2 = 0 $: Reduces to $y_t = \mu + \epsilon_t$ (white noise around mear

In [ ]:
```
# Generate synthetic data: local level model
np.random.seed(42)
T = 200  # 200 days

# True parameters
true_sigma_mu = 0.5   # State innovation (trend changes)
true_sigma_y = 1.5    # Observation noise
true_mu0 = 50.0       # Initial level

# Generate latent level (random walk)
mu_true = np.zeros(T)
mu_true[0] = true_mu0
for t in range(1, T):
    mu_true[t] = mu_true[t-1] + np.random.normal(0, true_sigma_mu)

# Generate observations
y_obs = mu_true + np.random.normal(0, true_sigma_y, T)

# Visualize
fig, ax = plt.subplots(figsize=(14, 5))
ax.plot(y_obs, 'o-', alpha=0.5, markersize=3, label='Observed Price', col
ax.plot(mu_true, linewidth=2.5, label='True Latent Level (μ)', color='red
ax.fill_between(range(T), mu_true - 2*true_sigma_y, mu_true + 2*true_sigm
                alpha=0.2, color='red', label='±2σ_y (Observation Noise)'
ax.set_xlabel('Time (days)', fontsize=12)
ax.set_ylabel('Price', fontsize=12)
ax.set_title('Local Level Model: Latent State vs Observations', fontsize=
ax.legend(loc='upper left')
ax.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()

print(f"Signal-to-Noise Ratio: σ_μ² / σ_y² = {true_sigma_mu**2 / true_sig
print(f"Lower ratio → smoother level (mean-reversion)")
print(f"Higher ratio → rapidly changing level (trending)")
```

In [ ]:
```
# Build local level model in PyMC
with pm.Model() as local_level_model:
    # Priors on volatilities
    sigma_mu = pm.HalfNormal('sigma_mu', sigma=2)  # State innovation
    sigma_y = pm.HalfNormal('sigma_y', sigma=5)    # Observation noise

    # Initial state
    mu_init = pm.Normal('mu_init', mu=50, sigma=10)

    # State innovations
    innovations = pm.Normal('innovations', mu=0, sigma=sigma_mu, shape=T-

    # Build level via cumulative sum
    mu = pm.Deterministic('mu', pm.math.concatenate([[mu_init], mu_init +
```

```
    # Observations
    y = pm.Normal('y', mu=mu, sigma=sigma_y, observed=y_obs)

    # Sample
    trace_local_level = pm.sample(2000, tune=1000, return_inferencedata=T
                                  target_accept=0.95, random_seed=42)

print("\nPosterior Summary:")
print(az.summary(trace_local_level, var_names=['sigma_mu', 'sigma_y']))
```

In [ ]:

```
# Extract and visualize latent level estimates
mu_posterior = trace_local_level.posterior['mu'].values
mu_mean = mu_posterior.mean(axis=(0, 1))
mu_lower = np.percentile(mu_posterior, 2.5, axis=(0, 1))
mu_upper = np.percentile(mu_posterior, 97.5, axis=(0, 1))

fig, ax = plt.subplots(figsize=(14, 6))
ax.plot(y_obs, 'o', alpha=0.4, markersize=3, label='Observed Price', colo
ax.plot(mu_true, linewidth=2, label='True Level', color='red', linestyle=
ax.plot(mu_mean, linewidth=2.5, label='Estimated Level (Posterior Mean)',
ax.fill_between(range(T), mu_lower, mu_upper, alpha=0.3, color='blue',
                label='95% Credible Interval')
ax.set_xlabel('Time (days)', fontsize=12)
ax.set_ylabel('Price', fontsize=12)
ax.set_title('Local Level Model: Filtering Noise from Signal', fontsize=1
ax.legend(loc='upper left')
ax.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()

print("\nTrading Interpretation:")
print("Blue line = 'Fair value' after filtering out noise")
print("When observed price (gray) deviates from blue line → mean-reversio
print(f"\nEstimated σ_μ: {trace_local_level.posterior['sigma_mu'].values.
print(f"Estimated σ_y: {trace_local_level.posterior['sigma_y'].values.mea
```

3 . Local Linear Trend Model

Theory

The local level model has no persistent trend. Add a **slope** component:

$$ \begin{align} y_t &= \mu_t + \epsilon_t, \quad \epsilon_t \sim \text{Normal}(0 , \sigma_y^ 2) &= \mu_{t- 1 } + \beta_{t- 1 } + \eta_t, \quad \eta_t \sim \text{Normal}(0 , \sigma_\mu^ 2) \\ \bet \beta_{t- 1 } + \zeta_t, \quad \zeta_t \sim \text{Normal}(0 , \sigma_\beta^ 2) \end{align} $$

Where:

- μ_t: Level (intercept)
- β_t: Slope (trend)
- Both evolve as random walks

Trading Interpretation

- $\beta_t > 0 $: Uptrend (bullish)

- $\beta_t < 0$: Downtrend (bearish)
- β_t changing sign: Trend reversal
- σ_β^2 large: Frequently changing trends (regime switches)

In [ ]:
```
# Generate data with time-varying trend
np.random.seed(42)
T_trend = 150

# True parameters
true_sigma_mu_trend = 0.3
true_sigma_beta = 0.05  # Slope changes slowly
true_sigma_y_trend = 1.2

# Initialize
mu_trend_true = np.zeros(T_trend)
beta_trend_true = np.zeros(T_trend)
mu_trend_true[0] = 60.0
beta_trend_true[0] = 0.2  # Initial uptrend

# Evolve states
for t in range(1, T_trend):
    beta_trend_true[t] = beta_trend_true[t-1] + np.random.normal(0, true_
    mu_trend_true[t] = mu_trend_true[t-1] + beta_trend_true[t-1] + np.ran

# Generate observations
y_trend_obs = mu_trend_true + np.random.normal(0, true_sigma_y_trend, T_t

# Visualize
fig, axes = plt.subplots(2, 1, figsize=(14, 8), sharex=True)

# Observed vs Level
axes[0].plot(y_trend_obs, 'o-', alpha=0.5, markersize=3, label='Observed
axes[0].plot(mu_trend_true, linewidth=2.5, label='True Level (μ)', color=
axes[0].set_ylabel('Price', fontsize=12)
axes[0].set_title('Local Linear Trend: Level Component', fontsize=12, fon
axes[0].legend()
axes[0].grid(True, alpha=0.3)

# Slope over time
axes[1].plot(beta_trend_true, linewidth=2, color='red')
axes[1].axhline(0, color='black', linestyle='--', alpha=0.7, linewidth=1.
axes[1].fill_between(range(T_trend), 0, beta_trend_true, where=(beta_tren
                     alpha=0.3, color='green', label='Uptrend')
axes[1].fill_between(range(T_trend), 0, beta_trend_true, where=(beta_tren
                     alpha=0.3, color='red', label='Downtrend')
axes[1].set_xlabel('Time (days)', fontsize=12)
axes[1].set_ylabel('Slope (β)', fontsize=12)
axes[1].set_title('Time-Varying Trend (Slope Component)', fontsize=12, fo
axes[1].legend()
axes[1].grid(True, alpha=0.3)

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

In [ ]:
```
# Build local linear trend model
with pm.Model() as llt_model:
    # Priors
    sigma_mu = pm.HalfNormal('sigma_mu', sigma=2)
```

```
    sigma_beta = pm.HalfNormal('sigma_beta', sigma=0.5)
    sigma_y = pm.HalfNormal('sigma_y', sigma=5)

    # Initial states
    mu_init = pm.Normal('mu_init', mu=60, sigma=10)
    beta_init = pm.Normal('beta_init', mu=0, sigma=1)

    # Innovations
    innovations_mu = pm.Normal('innovations_mu', mu=0, sigma=sigma_mu, sh
    innovations_beta = pm.Normal('innovations_beta', mu=0, sigma=sigma_be

    # Build slope (random walk)
    beta = pm.Deterministic('beta', pm.math.concatenate(
        [[beta_init], beta_init + pm.math.cumsum(innovations_beta)]
    ))

    # Build level (random walk with drift = slope)
    # μ_t = μ_{t-1} + β_{t-1} + η_t
    # We need to construct this recursively
    # Using scan for sequential dependency

    def level_step(beta_lag, mu_lag, innovation):
        return mu_lag + beta_lag + innovation

    mu_rest, _ = pm.scan(
        fn=level_step,
        sequences=[beta[:-1], innovations_mu],
        outputs_info=[mu_init]
    )

    mu = pm.Deterministic('mu', pm.math.concatenate([[mu_init], mu_rest])

    # Observations
    y = pm.Normal('y', mu=mu, sigma=sigma_y, observed=y_trend_obs)

    # Sample
    trace_llt = pm.sample(2000, tune=1000, return_inferencedata=True,
                          target_accept=0.95, random_seed=42)

print("\nPosterior Summary:")
print(az.summary(trace_llt, var_names=['sigma_mu', 'sigma_beta', 'sigma_y
```

In [ ]:

```
# Visualize estimates
mu_llt_mean = trace_llt.posterior['mu'].mean(dim=['chain', 'draw']).value
beta_llt_mean = trace_llt.posterior['beta'].mean(dim=['chain', 'draw']).v
beta_llt_lower = np.percentile(trace_llt.posterior['beta'].values, 2.5, a
beta_llt_upper = np.percentile(trace_llt.posterior['beta'].values, 97.5,

fig, axes = plt.subplots(2, 1, figsize=(14, 8), sharex=True)

# Level
axes[0].plot(y_trend_obs, 'o', alpha=0.3, markersize=3, label='Observed',
axes[0].plot(mu_trend_true, linewidth=2, label='True Level', color='red',
axes[0].plot(mu_llt_mean, linewidth=2.5, label='Estimated Level', color='
axes[0].set_ylabel('Price', fontsize=12)
axes[0].set_title('Estimated Level (μ)', fontsize=12, fontweight='bold')
axes[0].legend()
axes[0].grid(True, alpha=0.3)

# Slope
```

```
axes[1].plot(beta_trend_true, linewidth=2, label='True Slope', color='red
axes[1].plot(beta_llt_mean, linewidth=2.5, label='Estimated Slope', color
axes[1].fill_between(range(T_trend), beta_llt_lower, beta_llt_upper,
                     alpha=0.3, color='green', label='95% CI')
axes[1].axhline(0, color='black', linestyle='--', alpha=0.7)
axes[1].set_xlabel('Time (days)', fontsize=12)
axes[1].set_ylabel('Slope (β)', fontsize=12)
axes[1].set_title('Estimated Slope (Trend)', fontsize=12, fontweight='bol
axes[1].legend()
axes[1].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

# Trading signals
current_slope = beta_llt_mean[-1]
prob_uptrend = np.mean(trace_llt.posterior['beta'].values[:, :, -1] > 0)

print(f"\nCurrent Slope Estimate: {current_slope:.4f}")
print(f"Probability of Uptrend: {prob_uptrend:.2%}")
if prob_uptrend > 0.75:
    print("→ BULLISH: Strong uptrend detected")
elif prob_uptrend < 0.25:
    print("→ BEARISH: Strong downtrend detected")
else:
    print("→ NEUTRAL: Trend direction uncertain")
```

4 . Seasonal Components

Theory

Commodities exhibit strong seasonality:

- **Natural gas**: Winter heating demand
- **Agriculture**: Planting and harvest cycles
- **Power**: Summer cooling load

Add a seasonal component with period S:

$$ \begin{align} y_t &= \mu_t + \gamma_t + \epsilon_t \\ \gamma_t &= -\sum_{s= 1 }^{S- 1 } \gai s} + \omega_t, \quad \omega_t \sim \text{Normal}(0 , \sigma_\gamma^ 2) \end{align} $$

The constraint ensures seasonal effects sum to zero over each cycle.

Trading Application

- Identify **high-seasonality months** to increase position size
- **Calendar spreads**: Long winter months, short summer months for nat gas
- **Harvest pressure**: Short agricultural commodities during harvest

In [ ]:
```
# Generate data with annual seasonality (12 periods)
np.random.seed(42)
T_seasonal = 120  # 10 years of monthly data
S = 12  # Annual seasonality
```

```
# Trend
trend_seasonal = 50 + 0.05 * np.arange(T_seasonal)

# Seasonal pattern (higher in winter: months 0, 1, 11)
seasonal_pattern = np.array([5, 4, 2, -1, -3, -4, -5, -4, -2, 1, 3, 4])
seasonal_component = np.tile(seasonal_pattern, T_seasonal // S)

# Observations
y_seasonal_obs = trend_seasonal + seasonal_component + np.random.normal(0

# Month labels
months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep',
month_indices = np.arange(T_seasonal) % 12

# Visualize
fig, axes = plt.subplots(2, 1, figsize=(14, 8), sharex=True)

# Time series
axes[0].plot(y_seasonal_obs, 'o-', alpha=0.6, markersize=3, label='Observ
axes[0].plot(trend_seasonal, linewidth=2, label='Trend', color='red', lin
axes[0].plot(trend_seasonal + seasonal_component, linewidth=2, label='Tre
axes[0].set_ylabel('Price', fontsize=12)
axes[0].set_title('Natural Gas Prices with Annual Seasonality', fontsize=
axes[0].legend()
axes[0].grid(True, alpha=0.3)

# Seasonal pattern
axes[1].bar(range(12), seasonal_pattern, color='steelblue', alpha=0.7)
axes[1].set_xticks(range(12))
axes[1].set_xticklabels(months)
axes[1].set_ylabel('Seasonal Effect', fontsize=12)
axes[1].set_title('Monthly Seasonal Pattern (Winter Premium)', fontsize=1
axes[1].axhline(0, color='black', linestyle='--')
axes[1].grid(True, alpha=0.3, axis='y')

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

In [ ]:

```
# Build model with trend + seasonality
# For simplicity, we'll use a fixed seasonal pattern approach
# (More sophisticated: stochastic seasonality with state evolution)

with pm.Model() as seasonal_model:
    # Trend component (local level)
    sigma_mu = pm.HalfNormal('sigma_mu', sigma=1)
    mu_init = pm.Normal('mu_init', mu=50, sigma=10)
    innovations_mu = pm.Normal('innovations_mu', mu=0, sigma=sigma_mu, sh
    mu = pm.Deterministic('mu', pm.math.concatenate(
        [[mu_init], mu_init + pm.math.cumsum(innovations_mu)]
    ))

    # Seasonal component (one parameter per month, sum-to-zero constraint
    # We'll use a centered parameterization
    seasonal_raw = pm.Normal('seasonal_raw', mu=0, sigma=5, shape=S-1)
    # Last seasonal component ensures sum = 0
    seasonal_effects = pm.Deterministic('seasonal_effects',
                                        pm.math.concatenate([seasonal_raw

    # Map seasonal effects to each time point
    seasonal_component = seasonal_effects[month_indices]
```

```
    # Observation noise
    sigma_y = pm.HalfNormal('sigma_y', sigma=5)

    # Combined mean
    mean = mu + seasonal_component

    # Likelihood
    y = pm.Normal('y', mu=mean, sigma=sigma_y, observed=y_seasonal_obs)

    # Sample
    trace_seasonal = pm.sample(2000, tune=1000, return_inferencedata=True
                               target_accept=0.95, random_seed=42)

print("\nPosterior Summary:")
print(az.summary(trace_seasonal, var_names=['sigma_mu', 'sigma_y', 'seaso
```

In [ ]:

```
# Extract estimates
mu_seasonal_mean = trace_seasonal.posterior['mu'].mean(dim=['chain', 'dra
seasonal_effects_mean = trace_seasonal.posterior['seasonal_effects'].mean
seasonal_component_fitted = seasonal_effects_mean[month_indices]

fig, axes = plt.subplots(2, 1, figsize=(14, 8))

# Decomposition
axes[0].plot(y_seasonal_obs, 'o', alpha=0.4, markersize=3, label='Observe
axes[0].plot(mu_seasonal_mean, linewidth=2, label='Estimated Trend', colo
axes[0].plot(mu_seasonal_mean + seasonal_component_fitted, linewidth=2,
             label='Trend + Seasonal', color='blue')
axes[0].set_ylabel('Price', fontsize=12)
axes[0].set_title('Decomposition: Trend + Seasonality', fontsize=12, font
axes[0].legend()
axes[0].grid(True, alpha=0.3)

# Seasonal effects comparison
x_pos = np.arange(12)
axes[1].bar(x_pos - 0.2, seasonal_pattern, width=0.4, label='True', alpha
axes[1].bar(x_pos + 0.2, seasonal_effects_mean, width=0.4, label='Estimat
axes[1].set_xticks(x_pos)
axes[1].set_xticklabels(months)
axes[1].set_ylabel('Seasonal Effect', fontsize=12)
axes[1].set_title('Estimated vs True Seasonal Pattern', fontsize=12, font
axes[1].axhline(0, color='black', linestyle='--')
axes[1].legend()
axes[1].grid(True, alpha=0.3, axis='y')

plt.tight_layout()
plt.show()

# Trading signals
print("\n=== SEASONAL TRADING STRATEGY ===")
sorted_months = np.argsort(seasonal_effects_mean)[::-1]
print("\nMonths with HIGHEST prices (go long ahead of these):")
for i in range(3):
    month_idx = sorted_months[i]
    print(f"  {months[month_idx]}: +${seasonal_effects_mean[month_idx]:.2

print("\nMonths with LOWEST prices (go short or avoid):")
for i in range(3):
```

```
    month_idx = sorted_months[-(i+1)]
    print(f"  {months[month_idx]}: ${seasonal_effects_mean[month_idx]:.2f
```

5 . Dynamic Regression with Time-Varying Coefficients

Theory

In static regression, β is constant. In **dynamic regression**, coefficients evolve:

$$ \begin{align} y_t &= \alpha_t + \beta_t x_t + \epsilon_t \\ \alpha_t &= \alpha_{t- 1 } + \eta_{\alp
\beta_t &= \beta_{t- 1 } + \eta_{\beta,t} \end{align} $$

Trading Application

Gold-USD relationship changes:

- **Normal times**: $\beta < 0 $ (inverse relationship)
- **Flight to safety**: $\beta \approx 0 $ (decoupling)
- **Currency crisis**: β very negative

Dynamic regression detects these regime changes automatically.

In [ ]:
```
# Generate data with time-varying coefficient
np.random.seed(42)
T_dyn = 200

# Predictor: USD index
usd_index = 95 + np.cumsum(np.random.normal(0, 0.3, T_dyn))
usd_standardized = (usd_index - usd_index.mean()) / usd_index.std()

# Time-varying coefficient (becomes more negative over time → stronger in
true_beta_dyn = np.zeros(T_dyn)
true_beta_dyn[0] = -0.5
for t in range(1, T_dyn):
    true_beta_dyn[t] = true_beta_dyn[t-1] + np.random.normal(0, 0.02)

# Gold price
true_alpha_dyn = 1500
gold_price = true_alpha_dyn + true_beta_dyn * usd_standardized * 100 + np

# Visualize
fig, axes = plt.subplots(3, 1, figsize=(14, 10), sharex=True)

# Gold price
axes[0].plot(gold_price, linewidth=1.5, color='gold')
axes[0].set_ylabel('Gold Price ($/oz)', fontsize=12)
axes[0].set_title('Gold Prices', fontsize=12, fontweight='bold')
axes[0].grid(True, alpha=0.3)

# USD Index
axes[1].plot(usd_index, linewidth=1.5, color='green')
axes[1].set_ylabel('USD Index', fontsize=12)
axes[1].set_title('USD Index (Predictor)', fontsize=12, fontweight='bold'
axes[1].grid(True, alpha=0.3)

# Time-varying coefficient
```

```
axes[2].plot(true_beta_dyn, linewidth=2, color='red')
axes[2].axhline(0, color='black', linestyle='--')
axes[2].fill_between(range(T_dyn), 0, true_beta_dyn, where=(true_beta_dyn
                     alpha=0.3, color='red', label='Inverse Relationship'
axes[2].set_xlabel('Time (days)', fontsize=12)
axes[2].set_ylabel('β (Gold-USD)', fontsize=12)
axes[2].set_title('Time-Varying Coefficient: Gold vs USD', fontsize=12, f
axes[2].legend()
axes[2].grid(True, alpha=0.3)

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

In [ ]:
```
# Build dynamic regression model
with pm.Model() as dynamic_reg_model:
    # Time-varying intercept
    sigma_alpha = pm.HalfNormal('sigma_alpha', sigma=10)
    alpha_init = pm.Normal('alpha_init', mu=1500, sigma=100)
    innovations_alpha = pm.Normal('innovations_alpha', mu=0, sigma=sigma_
    alpha = pm.Deterministic('alpha', pm.math.concatenate(
        [[alpha_init], alpha_init + pm.math.cumsum(innovations_alpha)]
    ))

    # Time-varying coefficient on USD
    sigma_beta = pm.HalfNormal('sigma_beta', sigma=0.5)
    beta_init = pm.Normal('beta_init', mu=-0.5, sigma=1)
    innovations_beta = pm.Normal('innovations_beta', mu=0, sigma=sigma_be
    beta = pm.Deterministic('beta', pm.math.concatenate(
        [[beta_init], beta_init + pm.math.cumsum(innovations_beta)]
    ))

    # Observation noise
    sigma_y = pm.HalfNormal('sigma_y', sigma=50)

    # Regression equation
    mu = alpha + beta * usd_standardized * 100

    # Likelihood
    y = pm.Normal('y', mu=mu, sigma=sigma_y, observed=gold_price)

    # Sample
    trace_dyn_reg = pm.sample(2000, tune=1000, return_inferencedata=True,
                              target_accept=0.95, random_seed=42)

print("\nPosterior Summary:")
print(az.summary(trace_dyn_reg, var_names=['sigma_alpha', 'sigma_beta', '
```

In [ ]:
```
# Visualize time-varying coefficient
beta_dyn_mean = trace_dyn_reg.posterior['beta'].mean(dim=['chain', 'draw'
beta_dyn_lower = np.percentile(trace_dyn_reg.posterior['beta'].values, 2.
beta_dyn_upper = np.percentile(trace_dyn_reg.posterior['beta'].values, 97

fig, axes = plt.subplots(2, 1, figsize=(14, 8), sharex=True)

# Fitted vs Observed
alpha_dyn_mean = trace_dyn_reg.posterior['alpha'].mean(dim=['chain', 'dra
fitted = alpha_dyn_mean + beta_dyn_mean * usd_standardized * 100

axes[0].plot(gold_price, 'o', alpha=0.4, markersize=3, label='Observed',
```

```
axes[0].plot(fitted, linewidth=2, label='Fitted', color='gold')
axes[0].set_ylabel('Gold Price ($/oz)', fontsize=12)
axes[0].set_title('Dynamic Regression Fit', fontsize=12, fontweight='bold
axes[0].legend()
axes[0].grid(True, alpha=0.3)

# Time-varying coefficient
axes[1].plot(true_beta_dyn, linewidth=2, label='True β', color='red', lin
axes[1].plot(beta_dyn_mean, linewidth=2.5, label='Estimated β', color='bl
axes[1].fill_between(range(T_dyn), beta_dyn_lower, beta_dyn_upper,
                     alpha=0.3, color='blue', label='95% CI')
axes[1].axhline(0, color='black', linestyle='--', alpha=0.7)
axes[1].set_xlabel('Time (days)', fontsize=12)
axes[1].set_ylabel('β (Gold-USD)', fontsize=12)
axes[1].set_title('Time-Varying Coefficient: Gold vs USD', fontsize=12, f
axes[1].legend()
axes[1].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

# Trading signals
current_beta = beta_dyn_mean[-1]
recent_beta_change = beta_dyn_mean[-1] - beta_dyn_mean[-20]

print(f"\n=== GOLD-USD RELATIONSHIP ===")
print(f"Current β: {current_beta:.3f}")
print(f"20-day change in β: {recent_beta_change:.3f}")
if current_beta < -0.5:
    print("→ STRONG INVERSE: USD strength heavily pressures gold")
    print("   Strategy: Short gold when USD rallies")
elif current_beta > -0.2:
    print("→ WEAK INVERSE or DECOUPLED: Relationship breakdown")
    print("   Strategy: USD less reliable predictor, watch other drivers"
else:
    print("→ MODERATE INVERSE: Typical negative correlation")
    print("   Strategy: Use USD as hedge factor")
```

6 . Practical Application: Gold Prices with USD and Infl

Trading Context

Gold is driven by:

1 . **USD strength**: Inverse (commodities priced in USD)
2 . **Inflation expectations**: Positive (inflation hedge)
3 . **Real rates**: Inverse (opportunity cost)

We'll build a full BSTS model with:

- Local linear trend
- Dynamic regression on USD and inflation
- Student-t likelihood (robust to outliers)

In [ ]:
```
# Generate realistic gold price data
np.random.seed(42)
T_gold = 250  # ~1 year of daily data
```

```python

# Predictors
usd = 95 + np.cumsum(np.random.normal(0, 0.2, T_gold))  # USD index
inflation = 2.5 + np.cumsum(np.random.normal(0, 0.05, T_gold))  # Inflati

# Standardize
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
predictors = scaler.fit_transform(np.column_stack([usd, inflation]))
usd_std = predictors[:, 0]
inflation_std = predictors[:, 1]

# Time-varying coefficients
beta_usd_true = np.zeros(T_gold)
beta_inflation_true = np.zeros(T_gold)
beta_usd_true[0] = -15  # Negative: strong USD → lower gold
beta_inflation_true[0] = 25  # Positive: inflation → higher gold

for t in range(1, T_gold):
    beta_usd_true[t] = beta_usd_true[t-1] + np.random.normal(0, 0.5)
    beta_inflation_true[t] = beta_inflation_true[t-1] + np.random.normal(

# Trend (local linear trend)
mu_gold = np.zeros(T_gold)
beta_trend = np.zeros(T_gold)
mu_gold[0] = 1800
beta_trend[0] = 0.3

for t in range(1, T_gold):
    beta_trend[t] = beta_trend[t-1] + np.random.normal(0, 0.05)
    mu_gold[t] = mu_gold[t-1] + beta_trend[t-1] + np.random.normal(0, 2)

# Gold price
gold_full = (mu_gold +
             beta_usd_true * usd_std +
             beta_inflation_true * inflation_std +
             np.random.standard_t(df=5, size=T_gold) * 8)  # Fat tails

# Visualize
fig, axes = plt.subplots(4, 1, figsize=(14, 12), sharex=True)

axes[0].plot(gold_full, linewidth=1.5, color='gold')
axes[0].set_ylabel('Gold ($/oz)', fontsize=11)
axes[0].set_title('Gold Prices', fontsize=12, fontweight='bold')
axes[0].grid(True, alpha=0.3)

axes[1].plot(usd, linewidth=1.5, color='green')
axes[1].set_ylabel('USD Index', fontsize=11)
axes[1].set_title('USD Index', fontsize=12, fontweight='bold')
axes[1].grid(True, alpha=0.3)

axes[2].plot(inflation, linewidth=1.5, color='red')
axes[2].set_ylabel('Inflation (%)', fontsize=11)
axes[2].set_title('Inflation Expectations', fontsize=12, fontweight='bold
axes[2].grid(True, alpha=0.3)

axes[3].plot(beta_usd_true, linewidth=2, label='β (USD)', color='green')
axes[3].plot(beta_inflation_true, linewidth=2, label='β (Inflation)', col
axes[3].axhline(0, color='black', linestyle='--')
axes[3].set_xlabel('Time (days)', fontsize=11)
```

```
axes[3].set_ylabel('Coefficient', fontsize=11)
axes[3].set_title('Time-Varying Coefficients (True)', fontsize=12, fontwe
axes[3].legend()
axes[3].grid(True, alpha=0.3)

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

In [ ]:
```
# Build full BSTS model
# This is computationally intensive, so we'll use a simplified version

with pm.Model() as bsts_full:
    # Trend component (local level only for speed)
    sigma_mu = pm.HalfNormal('sigma_mu', sigma=5)
    mu_init = pm.Normal('mu_init', mu=1800, sigma=100)
    innovations_mu = pm.Normal('innovations_mu', mu=0, sigma=sigma_mu, sh
    mu = pm.Deterministic('mu', pm.math.concatenate(
        [[mu_init], mu_init + pm.math.cumsum(innovations_mu)]
    ))

    # Time-varying USD coefficient
    sigma_beta_usd = pm.HalfNormal('sigma_beta_usd', sigma=2)
    beta_usd_init = pm.Normal('beta_usd_init', mu=-15, sigma=10)
    innovations_beta_usd = pm.Normal('innovations_beta_usd', mu=0, sigma=
    beta_usd = pm.Deterministic('beta_usd', pm.math.concatenate(
        [[beta_usd_init], beta_usd_init + pm.math.cumsum(innovations_beta
    ))

    # Time-varying inflation coefficient
    sigma_beta_inf = pm.HalfNormal('sigma_beta_inf', sigma=2)
    beta_inf_init = pm.Normal('beta_inf_init', mu=25, sigma=10)
    innovations_beta_inf = pm.Normal('innovations_beta_inf', mu=0, sigma=
    beta_inf = pm.Deterministic('beta_inf', pm.math.concatenate(
        [[beta_inf_init], beta_inf_init + pm.math.cumsum(innovations_beta
    ))

    # Robust likelihood (Student-t)
    nu = pm.Gamma('nu', alpha=2, beta=0.1)
    sigma_y = pm.HalfNormal('sigma_y', sigma=20)

    # Combined mean
    mean = mu + beta_usd * usd_std + beta_inf * inflation_std

    # Likelihood
    y = pm.StudentT('y', nu=nu, mu=mean, sigma=sigma_y, observed=gold_ful

    # Sample
    trace_bsts_full = pm.sample(1500, tune=1000, return_inferencedata=Tru
                                target_accept=0.95, random_seed=42, chain

print("\nPosterior Summary:")
print(az.summary(trace_bsts_full, var_names=['sigma_mu', 'sigma_beta_usd'
                                             'sigma_y', 'nu']))
```

In [ ]:
```
# Decompose gold prices
mu_est = trace_bsts_full.posterior['mu'].mean(dim=['chain', 'draw']).valu
beta_usd_est = trace_bsts_full.posterior['beta_usd'].mean(dim=['chain', '
beta_inf_est = trace_bsts_full.posterior['beta_inf'].mean(dim=['chain', '
```

```
# Components
usd_effect = beta_usd_est * usd_std
inflation_effect = beta_inf_est * inflation_std
fitted = mu_est + usd_effect + inflation_effect

fig, axes = plt.subplots(3, 1, figsize=(14, 10), sharex=True)

# Observed vs Fitted
axes[0].plot(gold_full, 'o', alpha=0.4, markersize=3, label='Observed', c
axes[0].plot(fitted, linewidth=2, label='Fitted (BSTS)', color='gold')
axes[0].set_ylabel('Gold Price ($/oz)', fontsize=11)
axes[0].set_title('BSTS Model: Fitted vs Observed', fontsize=12, fontweig
axes[0].legend()
axes[0].grid(True, alpha=0.3)

# Component decomposition
axes[1].plot(mu_est, linewidth=2, label='Trend (μ)', color='black')
axes[1].plot(mu_est + usd_effect, linewidth=2, label='Trend + USD', color
axes[1].plot(fitted, linewidth=2, label='Trend + USD + Inflation', color=
axes[1].set_ylabel('Price Components ($/oz)', fontsize=11)
axes[1].set_title('Price Decomposition', fontsize=12, fontweight='bold')
axes[1].legend()
axes[1].grid(True, alpha=0.3)

# Time-varying coefficients
axes[2].plot(beta_usd_true, linewidth=2, label='True β (USD)', color='gre
axes[2].plot(beta_usd_est, linewidth=2, label='Est β (USD)', color='green
axes[2].plot(beta_inflation_true, linewidth=2, label='True β (Inflation)'
axes[2].plot(beta_inf_est, linewidth=2, label='Est β (Inflation)', color=
axes[2].axhline(0, color='black', linestyle='--', alpha=0.5)
axes[2].set_xlabel('Time (days)', fontsize=11)
axes[2].set_ylabel('Coefficient', fontsize=11)
axes[2].set_title('Time-Varying Coefficients', fontsize=12, fontweight='b
axes[2].legend(loc='best', fontsize=9)
axes[2].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

print(f"\nModel RMSE: ${np.sqrt(np.mean((gold_full - fitted)**2)):.2f}")
```

In [ ]:

```
# Trading signals based on decomposition
current_trend = mu_est[-1]
current_beta_usd = beta_usd_est[-1]
current_beta_inf = beta_inf_est[-1]
current_usd_effect = usd_effect[-1]
current_inf_effect = inflation_effect[-1]
current_fair_value = fitted[-1]
current_price = gold_full[-1]
mispricing = current_price - current_fair_value

print("\n" + "="*60)
print("GOLD TRADING DASHBOARD")
print("="*60)

print(f"\nCurrent Price: ${current_price:.2f}")
print(f"Fair Value (BSTS): ${current_fair_value:.2f}")
print(f"Mispricing: ${mispricing:+.2f} ({mispricing/current_fair_value*10

if abs(mispricing) > 20:
```

```
    if mispricing > 0:
        print("→ OVERVALUED: Consider short position or tighten stops on
    else:
        print("→ UNDERVALUED: Consider long position or add to existing l
else:
    print("→ FAIRLY VALUED: No mispricing signal")

print(f"\n--- PRICE DECOMPOSITION ---")
print(f"Trend component: ${current_trend:.2f}")
print(f"USD effect: ${current_usd_effect:+.2f} (β = {current_beta_usd:.2f
print(f"Inflation effect: ${current_inf_effect:+.2f} (β = {current_beta_i

print(f"\n--- FACTOR SENSITIVITIES ---")
print(f"β (USD): {current_beta_usd:.2f}")
if current_beta_usd < -10:
    print("  → STRONG INVERSE: USD rallies hurt gold significantly")
    print("     Strategy: Hedge gold positions with long USD")
elif current_beta_usd > -5:
    print("  → WEAK INVERSE: Gold-USD correlation breaking down")
    print("     Strategy: USD less reliable hedge, watch other factors")

print(f"\nβ (Inflation): {current_beta_inf:.2f}")
if current_beta_inf > 20:
    print("  → STRONG INFLATION HEDGE: Rising inflation supports gold")
    print("     Strategy: Increase gold allocation in inflationary enviro
elif current_beta_inf < 10:
    print("  → WEAK INFLATION HEDGE: Gold not responding to inflation")
    print("     Strategy: Consider other inflation hedges (TIPS, commodit

print("\n" + "="*60)
```

Knowledge Check Quiz

Question 1

What is the key advantage of state-space models over standard regression?

A) Faster computation
B) Separate latent "true" state from noisy observations
C) Require fewer data points
D) Always more accurate

Answer: B - State-space models explicitly model hidden states (e.g., "fair value") that evolve ov separating signal from noise.

Question 2

In a local level model, what does the ratio $\sigma^2_\mu / \sigma^2_y$ control?

A) Model accuracy
B) Smoothness of the latent level
C) Number of parameters
D) Forecast horizon

Answer: B - Higher ratio → level changes rapidly (trending). Lower ratio → level nearly constan reverting).

Question 3

Why are seasonal components important for commodity trading?

A) They increase model complexity
B) They capture predictable price patterns (harvest, weather)
C) They are required by PyMC
D) They replace the need for fundamentals

Answer: B - Commodities have strong seasonal patterns (e.g., natural gas winter demand, agric harvest pressure) that create tradable opportunities.

Question 4

What does a time-varying coefficient (dynamic regression) detect?

A) Data errors
B) Regime changes in relationships between variables
C) Missing data
D) Outliers

Answer: B - Dynamic regression allows coefficients to evolve, detecting when relationships stre weaken, or reverse (e.g., Gold-USD correlation during crises).

Question 5

In the gold BSTS model, what does it mean if β (USD) becomes less negative over time?

A) USD is getting stronger
B) The inverse Gold-USD relationship is weakening
C) Gold is becoming more volatile
D) Model is overfitting

Answer: B - A coefficient moving toward zero indicates the relationship is weakening (decouplin possibly due to regime change or other dominant factors.

Exercises

Exercise 1 : Crude Oil Seasonality

Build a BSTS model for crude oil with:

- Local linear trend

- Weekly seasonality (S= 5 for trading days)
- Compare model with vs without seasonality using WAIC
- Identify which day of the week has highest/lowest prices

Exercise 2 : Natural Gas Multi-Seasonality

Natural gas has multiple seasonal patterns:

- Annual (S= 1 2 months): Winter heating
- Weekly (S= 7 days): Storage report Thursdays

Build a model with both. Which is more important?

Exercise 3 : Corn Prices with Weather

Generate synthetic corn data with:

- Annual harvest seasonality (low prices in fall)
- Dynamic regression on rainfall (time-varying coefficient)
- Simulate drought year where β (rainfall) spikes
- Detect the regime change automatically

Exercise 4 : Regime Detection

Using the gold BSTS model:

1. Define a regime as "when β (USD) < - 2 0 "
2. Calculate P(regime) at each time point
3. Visualize regime probabilities over time
4. Design a trading rule: increase position size when P(high sensitivity regime) > 0 . 8

Exercise 5 : Forecast Decomposition

For the gold model:

1. Forecast 3 0 days ahead
2. Decompose forecast into: trend + USD effect + inflation effect
3. Scenario analysis:
    - If USD rises 2 %, how much does gold forecast drop?
    - If inflation rises 0 . 5 %, how much does gold forecast rise?
4. Calculate forecast credible intervals

Summary

In this module, you learned:

1. **State-Space Models**: Separate latent states from noisy observations
2. **Local Level Model**: Random walk for slowly changing "fair value"

3. **Local Linear Trend**: Add time-varying slope for trend detection
4. **Seasonal Components**: Model predictable cycles (harvest, weather)
5. **Dynamic Regression**: Time-varying coefficients detect regime changes
6. **Gold Application**: Full BSTS with trend, USD, and inflation factors

Key Takeaways for Trading

- **Decomposition**: Break prices into trend + seasonal + fundamental + noise
- **Fair Value**: Use latent level estimate for mean-reversion trades
- **Regime Detection**: Time-varying coefficients automatically identify structural breaks
- **Seasonality**: Exploit recurring patterns with statistical confidence
- **Factor Sensitivity**: Track how responsive commodities are to drivers (USD, inflation)
- **Robust Estimation**: Student-t likelihood handles price shocks without bias

BSTS models provide a principled framework for understanding commodity price dynamics in nc stationary, regime-switching markets.

Preview of Next Module

Module 7: Hierarchical Models for Multiple Commodities

So far, we've modeled one commodity at a time. But markets are interconnected:

- **Energy complex**: WTI, Brent, Natural Gas share supply shocks
- **Grains**: Corn, Wheat, Soybeans compete for farmland
- **Metals**: Gold, Silver, Copper respond to economic cycles

Hierarchical models pool information across related commodities:

- Estimate **group-level parameters** (e.g., average energy seasonality)
- Allow **commodity-specific deviations** (Brent vs WTI spreads)
- **Shrinkage**: Regularize estimates for thinly-traded commodities

You'll learn:

- Partial pooling vs complete pooling vs no pooling
- Varying intercepts and slopes
- Cross-commodity correlation structures
- Pairs trading with hierarchical models

See you in Module 7!