Module 9: Volatility Modeling and Uncertai Quantification

Course: Bayesian Regression and Time Series Forecasting for Commodities Trading

---

Learning Objectives

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

1. **Distinguish** between epistemic (model) and aleatoric (irreducible) uncertainty
2. **Implement** GARCH models in a Bayesian framework for volatility clustering
3. **Build** stochastic volatility models with PyMC for time-varying risk
4. **Calculate** Value at Risk (VaR) and Conditional VaR (CVaR) from posterior predictive distribu
5. **Forecast** volatility regimes for risk management and position sizing
6. **Apply** Bayesian volatility models to crude oil for trading decisions

---

Why This Matters for Trading

Volatility is not constant—and ignoring this can be catastrophic for traders:

Real-World Examples

- **Crude oil (2020)**: Volatility spiked 10x during COVID, wiping out strategies calib "normal" volatility
- **Natural gas (Winter Storm Uri, 2021)**: Prices jumped 100x in days—fixed v models failed completely
- **Copper (2008)**: Volatility doubled during financial crisis; traders with constant-vol ass lost fortunes

Why Bayesian Volatility Modeling?

1. **Time-varying risk**: Volatility clusters (high vol follows high vol)
2. **Regime detection**: Identify when markets shift from calm to turbulent
3. **Option pricing**: Implied vol forecasts drive delta hedging and gamma trading
4. **Position sizing**: Scale positions by forecasted volatility (Kelly criterion)
5. **Risk management**: VaR/CVaR for regulatory compliance and internal limits
6. **Uncertainty decomposition**: Separate "we don't know the model" from "markets are rando

Bottom line: Accurate volatility forecasts = better risk-adjusted returns.

---

1. Two Types of Uncertainty

Before modeling volatility, we must understand **what we're uncertain about**.

1.1 Aleatoric Uncertainty (Irreducible)

Definition: Randomness inherent in the system.

- **Also called**: Statistical uncertainty, data uncertainty
- **Source**: Markets are fundamentally stochastic
- **Cannot be reduced**: Even with infinite data, prices are still random

Example:

- Crude oil price tomorrow is uncertain because of random supply/demand shocks
- No amount of historical data will make tomorrow's price deterministic

Mathematical form: $$y_t = \mu_t + \epsilon_t, \quad \epsilon_t \sim \mathcal{N}(0, \sigma_t^

The ϵ_t term captures aleatoric uncertainty.

1.2 Epistemic Uncertainty (Reducible)

Definition: Uncertainty about the model parameters.

- **Also called**: Model uncertainty, parameter uncertainty
- **Source**: Limited data, model misspecification
- **Can be reduced**: More data → tighter posterior → less epistemic uncertainty

Example:

- We're uncertain whether crude oil volatility is 20% or 30% annualized
- More historical data narrows our belief about true volatility

Mathematical form: $$\sigma \sim p(\sigma | \text{data})$$

The posterior distribution $p(\sigma | \text{data})$ captures epistemic uncertainty.

1.3 Total Predictive Uncertainty

Bayesian forecasts combine both:

$$p(y_{\text{future}} | \text{data}) = \int p(y_{\text{future}} | \theta) \cdot p(\theta | \text{data}) d\th

- **Inner term** $p(y_{\text{future}} | \theta)$: Aleatoric (data randomness given parameters)
- **Outer term** $p(\theta | \text{data})$: Epistemic (parameter uncertainty)
- **Integral**: Marginalizes over parameter uncertainty

Trading implication:

- **Short-term forecasts**: Dominated by aleatoric uncertainty (market randomness)
- **Long-term forecasts**: Dominated by epistemic uncertainty (don't know true parameters)
- **Position sizing**: Use total uncertainty (both sources matter)

In [ ]:
```
# Setup
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')
plt.rcParams['figure.figsize'] = (14, 6)
plt.rcParams['font.size'] = 11

print("Libraries loaded successfully!")
print(f"PyMC version: {pm.__version__}")
```

In [ ]:
```
# Demonstrate epistemic vs aleatoric uncertainty
def demonstrate_uncertainty_types():
    """
    Visualize epistemic vs aleatoric uncertainty with simple example.
    """
    # Generate data from known process
    true_mu = 100
    true_sigma = 10

    # Two scenarios: small vs large dataset
    n_small = 10
    n_large = 500

    np.random.seed(42)
    data_small = np.random.normal(true_mu, true_sigma, n_small)
    data_large = np.random.normal(true_mu, true_sigma, n_large)

    # Bayesian inference on mean and std
    def infer_params(data):
        with pm.Model() as model:
            mu = pm.Normal('mu', mu=100, sigma=20)
            sigma = pm.HalfNormal('sigma', sigma=15)
            y = pm.Normal('y', mu=mu, sigma=sigma, observed=data)
            trace = pm.sample(1000, tune=1000, chains=2, random_seed=42,
        return trace

    trace_small = infer_params(data_small)
    trace_large = infer_params(data_large)

    # Posterior predictive
    def get_predictions(trace, n_pred=1000):
        mu_samples = trace.posterior['mu'].values.flatten()
        sigma_samples = trace.posterior['sigma'].values.flatten()

        # Sample predictions
        n_samples = len(mu_samples)
        predictions = np.zeros((n_samples, n_pred))
        for i in range(n_samples):
            predictions[i, :] = np.random.normal(mu_samples[i], sigma_sam

        return predictions.flatten()
```

```python
pred_small = get_predictions(trace_small)
pred_large = get_predictions(trace_large)

# Plot
fig, axes = plt.subplots(2, 2, figsize=(15, 10))

# Small data: parameter uncertainty
ax = axes[0, 0]
ax.hist(trace_small.posterior['mu'].values.flatten(), bins=30, alpha=
        density=True, color='orange', label=f'n={n_small}')
ax.axvline(true_mu, color='red', linestyle='--', linewidth=2, label='
ax.set_xlabel('Mean (μ)')
ax.set_ylabel('Density')
ax.set_title('Epistemic Uncertainty: Small Dataset', fontweight='bold
ax.legend()
ax.grid(True, alpha=0.3)

# Large data: parameter uncertainty
ax = axes[0, 1]
ax.hist(trace_large.posterior['mu'].values.flatten(), bins=30, alpha=
        density=True, color='green', label=f'n={n_large}')
ax.axvline(true_mu, color='red', linestyle='--', linewidth=2, label='
ax.set_xlabel('Mean (μ)')
ax.set_ylabel('Density')
ax.set_title('Epistemic Uncertainty: Large Dataset', fontweight='bold
ax.legend()
ax.grid(True, alpha=0.3)

# Small data: predictive distribution
ax = axes[1, 0]
ax.hist(pred_small, bins=50, alpha=0.7, density=True, color='orange',
ax.hist(np.random.normal(true_mu, true_sigma, 10000), bins=50, alpha=
        density=True, color='blue', label='True distribution')
ax.set_xlabel('Predicted Value')
ax.set_ylabel('Density')
ax.set_title('Predictive Uncertainty: Small Dataset', fontweight='bol
ax.legend()
ax.grid(True, alpha=0.3)

# Large data: predictive distribution
ax = axes[1, 1]
ax.hist(pred_large, bins=50, alpha=0.7, density=True, color='green',
ax.hist(np.random.normal(true_mu, true_sigma, 10000), bins=50, alpha=
        density=True, color='blue', label='True distribution')
ax.set_xlabel('Predicted Value')
ax.set_ylabel('Density')
ax.set_title('Predictive Uncertainty: Large Dataset', fontweight='bol
ax.legend()
ax.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

# Quantify uncertainties
epistemic_small = np.std(trace_small.posterior['mu'].values)
epistemic_large = np.std(trace_large.posterior['mu'].values)
aleatoric_small = np.mean(trace_small.posterior['sigma'].values)
aleatoric_large = np.mean(trace_large.posterior['sigma'].values)
```

```
    print("\n" + "="*70)
    print("UNCERTAINTY QUANTIFICATION")
    print("="*70)
    print(f"\nSmall Dataset (n={n_small}):")
    print(f"  Epistemic (parameter uncertainty): σ_μ = {epistemic_small:.
    print(f"  Aleatoric (data randomness):        σ = {aleatoric_small:.2
    print(f"  Total predictive std:               {np.std(pred_small):.2f
    print(f"\nLarge Dataset (n={n_large}):")
    print(f"  Epistemic (parameter uncertainty): σ_μ = {epistemic_large:.
    print(f"  Aleatoric (data randomness):        σ = {aleatoric_large:.2
    print(f"  Total predictive std:               {np.std(pred_large):.2f

    print("\n" + "="*70)
    print("KEY INSIGHTS")
    print("="*70)
    print(f"""
1. Epistemic uncertainty DECREASES with more data:
   - Small dataset: σ_μ = {epistemic_small:.2f}
   - Large dataset: σ_μ = {epistemic_large:.2f}
   - Reduction: {epistemic_small/epistemic_large:.1f}x

2. Aleatoric uncertainty STAYS CONSTANT:
   - Small dataset: σ = {aleatoric_small:.2f}
   - Large dataset: σ = {aleatoric_large:.2f}
   - This is irreducible market randomness!

3. Predictive uncertainty converges to aleatoric:
   - Small data: wider (epistemic + aleatoric)
   - Large data: narrower (mostly aleatoric)

**Trading Application**:
- New commodity with limited history → High epistemic uncertainty
  → Use wider stop-losses, smaller positions
- Mature commodity with decades of data → Low epistemic uncertainty
  → Can size positions more aggressively
    """)

demonstrate_uncertainty_types()
```

2 . GARCH Models: Volatility Clustering

2 . 1 The Stylized Fact: Volatility Clusters

Observation: Large price changes tend to follow large price changes.

- Mandelbrot (1 9 6 3): "Large changes tend to be followed by large changes—of either s small changes tend to be followed by small changes."
- This violates the constant variance assumption of standard regression

2 . 2 GARCH(1 , 1) Model

Generalized Autoregressive Conditional Heteroskedasticity

$$\begin{align} r_t &= \mu + \epsilon_t \quad \text{(return equation)} \\ \epsilon_t &= \sigma_t z_t z_t \sim \mathcal{N}(0 , 1) \quad \text{(standardized shock)} \\ \sigma_t^ 2 &= \omega + \epsilon_{t- 1 }^ 2 + \beta \sigma_{t- 1 }^ 2 \quad \text{(variance equation)} \end{align}$$

Parameters:

- $\omega > 0$: Baseline variance
- $\alpha \geq 0$: Reaction to shocks (ARCH effect)
- $\beta \geq 0$: Persistence of volatility (GARCH effect)
- **Stationarity condition**: $\alpha + \beta < 1$

Interpretation:

- $\alpha$ high: Volatility reacts strongly to recent shocks
- $\beta$ high: Volatility shocks persist for long time
- $\alpha + \beta \approx 1$: Volatility shocks are nearly permanent (integrated GARCH)

2.3 Bayesian GARCH vs Frequentist MLE

Frequentist GARCH:

- Maximum likelihood estimation
- Point estimates for $\alpha, \beta, \omega$
- No parameter uncertainty

Bayesian GARCH:

- Full posterior distributions for parameters
- Propagate parameter uncertainty to volatility forecasts
- Natural shrinkage through priors
- Can incorporate expert beliefs about volatility persistence

In [ ]:
```
# Simulate GARCH(1,1) process
def simulate_garch(n=500, omega=0.1, alpha=0.15, beta=0.8, mu=0.0):
    """
    Simulate GARCH(1,1) returns.
    """
    returns = np.zeros(n)
    sigma2 = np.zeros(n)
    sigma2[0] = omega / (1 - alpha - beta)  # Unconditional variance

    for t in range(1, n):
        # Variance equation
        sigma2[t] = omega + alpha * returns[t-1]**2 + beta * sigma2[t-1]

        # Return equation
        returns[t] = mu + np.sqrt(sigma2[t]) * np.random.randn()

    return returns, np.sqrt(sigma2)

# Generate synthetic crude oil returns
np.random.seed(42)
returns, true_vol = simulate_garch(n=500, omega=0.05, alpha=0.12, beta=0.

# Convert to prices
prices = 70 * np.exp(np.cumsum(returns))

# Visualize
```

```python
fig, axes = plt.subplots(3, 1, figsize=(14, 10))

# Prices
ax = axes[0]
ax.plot(prices, linewidth=1.5, color='black')
ax.set_ylabel('Price ($/barrel)', fontsize=11)
ax.set_title('Simulated Crude Oil Prices (GARCH volatility)', fontsize=12
ax.grid(True, alpha=0.3)

# Returns
ax = axes[1]
ax.plot(returns, linewidth=1, color='blue', alpha=0.7)
ax.axhline(0, color='red', linestyle='--', linewidth=1)
ax.set_ylabel('Returns', fontsize=11)
ax.set_title('Returns (showing volatility clustering)', fontsize=12, font
ax.grid(True, alpha=0.3)

# Volatility
ax = axes[2]
ax.plot(true_vol, linewidth=2, color='red', label='True volatility (σ_t)'
# Rolling realized volatility (for comparison)
rolling_vol = pd.Series(returns).rolling(window=20).std() * np.sqrt(252)
ax.plot(rolling_vol, linewidth=1.5, color='green', alpha=0.7, label='Real
ax.set_xlabel('Time', fontsize=11)
ax.set_ylabel('Volatility', fontsize=11)
ax.set_title('GARCH Volatility (time-varying)', fontsize=12, fontweight='
ax.legend()
ax.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

print("\n" + "="*70)
print("VOLATILITY CLUSTERING EVIDENCE")
print("="*70)
print(f"""
Notice:
1. Periods of HIGH volatility (wide swings) cluster together
2. Periods of LOW volatility (calm) also cluster together
3. This is NOT captured by constant variance models!

Autocorrelation of squared returns (test for ARCH effects):
  lag-1: {np.corrcoef(returns[:-1]**2, returns[1:]**2)[0,1]:.3f}
  lag-5: {np.corrcoef(returns[:-5]**2, returns[5:]**2)[0,1]:.3f}

Positive autocorrelation in squared returns = volatility clustering!
""")
```

In [ ]:
```python
# Fit Bayesian GARCH(1,1)
# Note: Full Bayesian GARCH is computationally intensive in PyMC
# We'll use a simplified approach for demonstration

def fit_bayesian_garch_simple(returns, n_samples=1000):
    """
    Simplified Bayesian GARCH using PyMC.

    For production, consider using specialized packages like
    arch (Python) with Bayesian extensions.
    """
    # Use first portion of data
    returns_train = returns[:400]

    with pm.Model() as garch_model:
        # Priors for GARCH parameters
        omega = pm.HalfNormal('omega', sigma=0.1)
        alpha = pm.Beta('alpha', alpha=2, beta=8)  # Concentrated near 0.
        beta = pm.Beta('beta', alpha=8, beta=2)    # Concentrated near 0.

        # Initialize variance
        initial_vol = pm.HalfNormal('initial_vol', sigma=0.5)

        # GARCH recursion (using scan for efficiency)
        def garch_step(ret_prev, sigma2_prev, omega, alpha, beta):
            sigma2_new = omega + alpha * ret_prev**2 + beta * sigma2_prev
            return sigma2_new

        # Compute variance series
        sigma2_series, _ = pm.scan(
            fn=garch_step,
            sequences=[returns_train[:-1]],
            outputs_info=[initial_vol**2],
            non_sequences=[omega, alpha, beta],
            n_steps=len(returns_train)-1
        )

        # Prepend initial variance
        sigma2_all = pm.math.concatenate([[initial_vol**2], sigma2_series

        # Likelihood
        returns_obs = pm.Normal('returns_obs',
                                mu=0,
                                sigma=pm.math.sqrt(sigma2_all),
                                observed=returns_train)

        # Sample posterior
        trace = pm.sample(n_samples, tune=1000, chains=2, random_seed=42,
                          progressbar=True, target_accept=0.95)

    return trace, garch_model

print("Fitting Bayesian GARCH(1,1)...")
print("(This may take a few minutes)\n")

trace_garch, model_garch = fit_bayesian_garch_simple(returns, n_samples=5

print("\nGARCH model fitted successfully!")
```

In [ ]:

```
# Analyze GARCH parameter posteriors
fig, axes = plt.subplots(1, 3, figsize=(15, 4))

params = ['omega', 'alpha', 'beta']
true_values = [0.05, 0.12, 0.85]
param_names = ['ω (baseline var)', 'α (ARCH)', 'β (GARCH)']

for ax, param, true_val, name in zip(axes, params, true_values, param_nam
    samples = trace_garch.posterior[param].values.flatten()

    ax.hist(samples, bins=30, alpha=0.7, density=True, color='steelblue',
    ax.axvline(true_val, color='red', linestyle='--', linewidth=2, label=
    ax.axvline(np.mean(samples), color='green', linestyle='-', linewidth=
                label=f'Post mean = {np.mean(samples):.3f}')
    ax.set_xlabel('Value', fontsize=11)
    ax.set_ylabel('Density', fontsize=11)
    ax.set_title(name, fontsize=12, fontweight='bold')
    ax.legend(fontsize=9)
    ax.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

# Summary statistics
print("\n" + "="*70)
print("GARCH PARAMETER POSTERIORS")
print("="*70)
print(f"{'Parameter':<15} {'True':>10} {'Post Mean':>12} {'Post Std':>12}
print("-"*70)

for param, true_val, name in zip(params, true_values, param_names):
    samples = trace_garch.posterior[param].values.flatten()
    mean_est = np.mean(samples)
    std_est = np.std(samples)
    ci = np.percentile(samples, [2.5, 97.5])
    print(f"{name:<15} {true_val:>10.3f} {mean_est:>12.3f} {std_est:>12.3

# Check persistence
alpha_samples = trace_garch.posterior['alpha'].values.flatten()
beta_samples = trace_garch.posterior['beta'].values.flatten()
persistence = alpha_samples + beta_samples

print("\n" + "="*70)
print("VOLATILITY PERSISTENCE")
print("="*70)
print(f"α + β (persistence):")
print(f"  Mean: {np.mean(persistence):.3f}")
print(f"  95% CI: [{np.percentile(persistence, 2.5):.3f}, {np.percentile(
print(f"\nInterpretation:")
print(f"  α + β = {np.mean(persistence):.3f} means volatility shocks have
print(f"  ~{-1/np.log(np.mean(persistence)):.1f} periods (days in this ca
print(f"\n  Close to 1 → volatility shocks are very persistent (typical f
print(f"  Far from 1 → volatility mean-reverts quickly")
```

3 . Stochastic Volatility Models

3 . 1 Limitations of GARCH

GARCH models have **deterministic volatility dynamics**:

- Volatility is a deterministic function of past returns
- No separate shock term for volatility itself

3 . 2 Stochastic Volatility (SV) Model

Idea: Volatility has its own random shocks.

$$\begin{align} r_t &= \exp(h_t/ 2) \epsilon_t, \quad \epsilon_t \sim \mathcal{N}(0 , 1) \quad \text{(return equation)} \\ h_t &= \mu_h + \phi (h_{t- 1 } - \mu_h) + \sigma_h \eta_t, \quad \eta_t \sim \mathcal{N}(0 , 1) \quad \text{(log-vol equation)} \end{align}$$

Parameters:

- $h_t = \log(\sigma_t^ 2)$: Log-variance (ensures positivity)
- $\mu_h$: Mean log-variance
- $\phi \in (- 1 , 1)$: Persistence of volatility
- $\sigma_h$: Volatility of volatility (vol-of-vol)

Advantages over GARCH:

1 . **Separate volatility shocks**: $\eta_t$ drives volatility changes
2 . **Leverage effect**: Can model correlation between $\epsilon_t$ and $\eta_t$ (negative correl leverage)
3 . **Better option pricing**: Matches implied volatility smiles
4 . **More flexible**: Captures volatility spikes not driven by returns

3 . 3 Bayesian SV Estimation

Challenge: Latent volatility $h_t$ must be inferred.

Solution: MCMC samples both parameters $(\mu_h, \phi, \sigma_h)$ and latent states $\{h_ 1 ,
$ jointly.

In [ ]:
```
# Fit Stochastic Volatility model
def fit_stochastic_volatility(returns, n_samples=1000):
    """
    Fit stochastic volatility model using PyMC.
    """
    returns_train = returns[:400]

    with pm.Model() as sv_model:
        # Hyperparameters
        mu_h = pm.Normal('mu_h', mu=-3, sigma=2)  # Mean log-volatility
        phi = pm.Uniform('phi', lower=-0.999, upper=0.999)  # AR(1) persi
        sigma_h = pm.HalfNormal('sigma_h', sigma=0.5)  # Vol-of-vol

        # Initial log-volatility
        h_init = pm.Normal('h_init', mu=mu_h, sigma=sigma_h / pm.math.sqr

        # Log-volatility random walk (AR(1))
        h = pm.GaussianRandomWalk('h',
                                  mu=mu_h * (1 - phi),
                                  sigma=sigma_h * pm.math.sqrt(1 - phi**2
                                  init_dist=pm.Normal.dist(mu_h, sigma_h)
                                  steps=len(returns_train)-1)

        # Return likelihood
        returns_obs = pm.Normal('returns_obs',
                                mu=0,
                                sigma=pm.math.exp(h/2),
                                observed=returns_train)

        # Sample
        trace = pm.sample(n_samples, tune=1000, chains=2, random_seed=42,
                          progressbar=True, target_accept=0.9)
    return trace, sv_model

print("Fitting Stochastic Volatility model...")
print("(This may take several minutes)\n")

trace_sv, model_sv = fit_stochastic_volatility(returns, n_samples=500)

print("\nStochastic Volatility model fitted!")
```

In [ ]:

```
# Extract and visualize latent volatility
h_samples = trace_sv.posterior['h'].values  # Shape: (chains, draws, time
h_mean = h_samples.mean(axis=(0, 1))
h_std = h_samples.std(axis=(0, 1))

# Convert log-vol to vol
vol_mean = np.exp(h_mean / 2)
vol_lower = np.exp((h_mean - 1.96*h_std) / 2)
vol_upper = np.exp((h_mean + 1.96*h_std) / 2)

# Plot
fig, axes = plt.subplots(2, 1, figsize=(14, 9))

# Returns
ax = axes[0]
ax.plot(returns[:400], linewidth=1, color='black', alpha=0.7)
ax.axhline(0, color='red', linestyle='--', linewidth=1)
ax.set_ylabel('Returns', fontsize=11)
ax.set_title('Crude Oil Returns', fontsize=12, fontweight='bold')
ax.grid(True, alpha=0.3)

# Estimated volatility
ax = axes[1]
ax.plot(vol_mean, linewidth=2, color='blue', label='Estimated volatility
ax.fill_between(range(len(vol_mean)), vol_lower, vol_upper,
                alpha=0.3, color='blue', label='95% credible interval')
ax.plot(true_vol[:400], linewidth=2, color='red', linestyle='--',
        alpha=0.6, label='True volatility')
ax.set_xlabel('Time', fontsize=11)
ax.set_ylabel('Volatility (σ_t)', fontsize=11)
ax.set_title('Stochastic Volatility Estimates', fontsize=12, fontweight='
ax.legend()
ax.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

# Parameter summary
print("\n" + "="*70)
print("STOCHASTIC VOLATILITY PARAMETER POSTERIORS")
print("="*70)

sv_params = ['mu_h', 'phi', 'sigma_h']
sv_names = ['Mean log-vol (μ_h)', 'Persistence (φ)', 'Vol-of-vol (σ_h)']

print(f"{'Parameter':<25} {'Mean':>10} {'Std':>10} {'95% CI':>25}")
print("-"*70)

for param, name in zip(sv_params, sv_names):
    samples = trace_sv.posterior[param].values.flatten()
    mean_val = np.mean(samples)
    std_val = np.std(samples)
    ci = np.percentile(samples, [2.5, 97.5])
    print(f"{name:<25} {mean_val:>10.3f} {std_val:>10.3f} [{ci[0]:>8.3f},

phi_mean = np.mean(trace_sv.posterior['phi'].values)
sigma_h_mean = np.mean(trace_sv.posterior['sigma_h'].values)

print("\n" + "="*70)
print("INTERPRETATION")
print("="*70)
print(f"""
Persistence (φ): {phi_mean:.3f}
  → High persistence means volatility shocks last long
  → Half-life: ~{-np.log(2)/np.log(phi_mean):.1f} periods
  → Typical for commodities: φ ∈ [0.9, 0.99]

Vol-of-vol (σ_h): {sigma_h_mean:.3f}
  → Volatility itself is volatile!
  → Higher σ_h = more abrupt volatility regime changes
  → Important for option pricing (vega risk)

**Trading Application**:
- High φ → Volatility regimes are sticky
  → When vol spikes, it stays high for a while
  → Adjust position sizes for extended periods

- High σ_h → Volatility can change quickly
  → Need frequent rebalancing
  → Options may be mispriced (underestimate vol-of-vol)
""")
```

4 . Value at Risk (VaR) and Conditional VaR (CVaR)

4 . 1 What is VaR?

Value at Risk (VaR): The maximum loss expected over a time horizon at a given confidence lev

Mathematical definition: $$\text{VaR}_{\alpha}(X) = \inf\{x : P(X \leq x) \geq \alpha\}$$

Example:

- VaR at 9 5 % confidence = 5 th percentile of loss distribution
- "With 9 5 % probability, losses won't exceed $X"

4 . 2 What is CVaR?

Conditional Value at Risk (CVaR) / **Expected Shortfall (ES)**: Average loss **beyond** VaR.

$$\text{CVaR}_{\alpha}(X) = \mathbb{E}[X | X \leq \text{VaR}_{\alpha}(X)]$$

Why CVaR > VaR:

- **VaR ignores tail shape**: Only cares about threshold
- **CVaR captures tail risk**: Average of all extreme losses
- **CVaR is coherent**: Satisfies desirable mathematical properties (subadditivity)

4 . 3 Bayesian VaR/CVaR

Standard approach (frequentist):

1 . Estimate volatility $\hat{\sigma}$ (point estimate)
2 . Assume normality: $\text{VaR}_{ 0 . 0 5 } = \mu + \Phi^{- 1 }(0 . 0 5) \hat{\sigma}$
3 . No uncertainty about $\sigma$

Bayesian approach:

1 . Sample volatility from posterior: $\sigma \sim p(\sigma | \text{data})$
2 . For each $\sigma$ sample, generate future returns
3 . VaR/CVaR from posterior predictive distribution
4 . **Accounts for parameter uncertainty** → More conservative risk estimates

In [ ]:
```python
# Calculate Bayesian VaR and CVaR from SV model
def calculate_bayesian_var_cvar(trace_sv, horizon=1, n_simulations=10000,
    """
    Calculate VaR and CVaR from stochastic volatility model.

    Accounts for:
    1. Parameter uncertainty (from posterior)
    2. Future volatility uncertainty (from SV dynamics)
    3. Return randomness (aleatoric)
    """
    # Extract posterior samples
    mu_h_samples = trace_sv.posterior['mu_h'].values.flatten()
    phi_samples = trace_sv.posterior['phi'].values.flatten()
    sigma_h_samples = trace_sv.posterior['sigma_h'].values.flatten()
    h_last = trace_sv.posterior['h'].values[:, :, -1].flatten()  # Last l

    n_param_samples = len(mu_h_samples)
    future_returns = np.zeros(n_simulations)

    for i in range(n_simulations):
        # Sample parameters from posterior
        idx = np.random.randint(0, n_param_samples)
        mu_h = mu_h_samples[idx]
        phi = phi_samples[idx]
        sigma_h = sigma_h_samples[idx]
        h_t = h_last[idx]

        # Simulate future volatility path
        cumulative_return = 0
        for t in range(horizon):
            # Update log-volatility
            h_t = mu_h + phi * (h_t - mu_h) + sigma_h * np.random.randn()

            # Generate return
            vol_t = np.exp(h_t / 2)
            ret_t = vol_t * np.random.randn()
            cumulative_return += ret_t

        future_returns[i] = cumulative_return

    # Calculate VaR and CVaR
    alpha = 1 - confidence
    var = np.percentile(future_returns, alpha * 100)
    cvar = future_returns[future_returns <= var].mean()

    return future_returns, var, cvar

# Calculate for different horizons
horizons = [1, 5, 10, 20]  # 1-day, 1-week, 2-week, 1-month
confidence = 0.95

results = {}
for h in horizons:
    returns_sim, var, cvar = calculate_bayesian_var_cvar(trace_sv, horizo
                                                           n_simulations=5
                                                           confidence=conf
    results[h] = {'returns': returns_sim, 'var': var, 'cvar': cvar}
    print(f"Horizon {h:2d} days: VaR = {var:.4f}, CVaR = {cvar:.4f}")

print("\nRisk calculations complete!")
```

In [ ]:
```
# Visualize VaR and CVaR
fig, axes = plt.subplots(2, 2, figsize=(15, 10))

for ax, h in zip(axes.flatten(), horizons):
    returns_sim = results[h]['returns']
    var = results[h]['var']
    cvar = results[h]['cvar']

    # Histogram of simulated returns
    ax.hist(returns_sim, bins=50, alpha=0.7, density=True, color='lightbl
            edgecolor='black', label='Posterior predictive')

    # VaR line
    ax.axvline(var, color='orange', linestyle='--', linewidth=2.5,
               label=f'VaR (95%) = {var:.4f}')

    # CVaR line
    ax.axvline(cvar, color='red', linestyle='-', linewidth=2.5,
               label=f'CVaR (95%) = {cvar:.4f}')

    # Shade tail
    tail_returns = returns_sim[returns_sim <= var]
    ax.hist(tail_returns, bins=20, alpha=0.5, density=True, color='red',
            edgecolor='darkred', label='Tail (losses > VaR)')

    ax.set_xlabel('Cumulative Return', fontsize=11)
    ax.set_ylabel('Density', fontsize=11)
    ax.set_title(f'{h}-Day Horizon', fontsize=12, fontweight='bold')
    ax.legend(fontsize=9)
    ax.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

# Summary table
print("\n" + "="*70)
print("BAYESIAN VaR AND CVaR SUMMARY (95% Confidence)")
print("="*70)
print(f"{'Horizon':<10} {'VaR':>12} {'CVaR':>12} {'CVaR/VaR':>12} {'$ on
print("-"*70)
```

```
portfolio_value = 100000

for h in horizons:
    var = results[h]['var']
    cvar = results[h]['cvar']
    ratio = cvar / var if var != 0 else np.nan
    dollar_cvar = portfolio_value * abs(cvar)

    print(f"{h:2d} days    {var:>12.4f} {cvar:>12.4f} {ratio:>12.2f} ${do

print("\n" + "="*70)
print("INTERPRETATION FOR TRADERS")
print("="*70)
print(f"""
VaR (Value at Risk):
  "With 95% confidence, losses won't exceed VaR"
  Example: 1-day VaR = {results[1]['var']:.4f}
  → On a $100k position, max 1-day loss is ${portfolio_value * abs(result

CVaR (Conditional VaR / Expected Shortfall):
  "Average loss in the worst 5% of cases"
  Example: 1-day CVaR = {results[1]['cvar']:.4f}
  → When things go bad (worst 5%), expect to lose ${portfolio_value * abs

CVaR/VaR Ratio:
  > 1.0 means tail is fat (extreme losses are much worse than VaR)
  = 1.0 would mean no tail risk beyond VaR

  Our ratio: ~{results[1]['cvar']/results[1]['var']:.2f}
  → Tail risk is significant! Don't just rely on VaR.

**Risk Management Actions**:
1. Set position limits using CVaR (more conservative than VaR)
2. Increase margin requirements in high-volatility regimes
3. Use options to cap tail risk when CVaR/VaR ratio is high
4. Scale positions: Position size ∝ 1/CVaR
""")
```

5 . Volatility Forecasting for Risk Management

5 . 1 Why Forecast Volatility?

Trading applications:

1. **Position sizing**: Higher forecast vol → smaller positions
2. **Stop-loss placement**: Wider stops in high-vol regimes
3. **Option strategies**: Sell vol when forecast < implied, buy when forecast > implied
4. **Risk budgeting**: Allocate more risk capital to low-vol assets
5. **Margin requirements**: Exchanges use vol forecasts for margin

5 . 2 Multi-Step Ahead Volatility Forecasts

From SV model: $$h_{t+k} = \mu_h + \phi^k (h_t - \mu_h) + \text{noise}$$

Key insight: As $k \to \infty$, $h_{t+k} \to \mu_h$ (mean reversion).

Forecast variance: $$\text{Var}(h_{t+k}) = \sigma_h^ 2 \frac{ 1 - \phi^{ 2 k}}{ 1 - \phi^

- Short horizon: Low variance (know recent vol)
- Long horizon: Converges to unconditional variance

In [ ]:
```python
# Generate volatility forecasts from SV model
def forecast_volatility(trace_sv, n_ahead=30, n_samples=1000):
    """
    Generate multi-step ahead volatility forecasts.
    """
    # Extract parameters
    mu_h_samples = trace_sv.posterior['mu_h'].values.flatten()
    phi_samples = trace_sv.posterior['phi'].values.flatten()
    sigma_h_samples = trace_sv.posterior['sigma_h'].values.flatten()
    h_last = trace_sv.posterior['h'].values[:, :, -1].flatten()

    # Storage for forecasts
    vol_forecasts = np.zeros((n_samples, n_ahead))

    for i in range(n_samples):
        # Sample parameters
        idx = np.random.randint(0, len(mu_h_samples))
        mu_h = mu_h_samples[idx]
        phi = phi_samples[idx]
        sigma_h = sigma_h_samples[idx]
        h_t = h_last[idx]

        # Simulate forward
        for t in range(n_ahead):
            h_t = mu_h + phi * (h_t - mu_h) + sigma_h * np.random.randn()
            vol_forecasts[i, t] = np.exp(h_t / 2)

    return vol_forecasts

# Generate forecasts
n_ahead = 60  # 2 months ahead
vol_forecasts = forecast_volatility(trace_sv, n_ahead=n_ahead, n_samples=

# Summary statistics
vol_mean = vol_forecasts.mean(axis=0)
vol_median = np.median(vol_forecasts, axis=0)
vol_lower = np.percentile(vol_forecasts, 5, axis=0)
vol_upper = np.percentile(vol_forecasts, 95, axis=0)

# Plot
fig, axes = plt.subplots(2, 1, figsize=(14, 10))

# Forecast fan chart
ax = axes[0]
ax.plot(range(n_ahead), vol_mean, linewidth=2.5, color='blue', label='Mea
ax.plot(range(n_ahead), vol_median, linewidth=2, color='green',
        linestyle='--', label='Median forecast')
ax.fill_between(range(n_ahead), vol_lower, vol_upper,
                alpha=0.3, color='blue', label='90% prediction interval')
ax.axhline(vol_mean[-1], color='red', linestyle=':', linewidth=2,
           label=f'Long-run mean = {vol_mean[-1]:.4f}')
ax.set_xlabel('Days Ahead', fontsize=11)
ax.set_ylabel('Volatility', fontsize=11)
ax.set_title('Volatility Forecast from Stochastic Volatility Model',
             fontsize=12, fontweight='bold')
ax.legend()
ax.grid(True, alpha=0.3)

# Uncertainty growth
ax = axes[1]
forecast_std = vol_forecasts.std(axis=0)
ax.plot(range(n_ahead), forecast_std, linewidth=2.5, color='purple')
ax.fill_between(range(n_ahead), 0, forecast_std, alpha=0.3, color='purple
ax.set_xlabel('Days Ahead', fontsize=11)
ax.set_ylabel('Forecast Uncertainty (Std Dev)', fontsize=11)
ax.set_title('Volatility Forecast Uncertainty Growth', fontsize=12, fontw
ax.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

print("\n" + "="*70)
print("VOLATILITY FORECAST SUMMARY")
print("="*70)
print(f"\nCurrent volatility (last obs): {vol_mean[0]:.4f}")
print(f"1-week ahead forecast:         {vol_mean[4]:.4f}")
print(f"1-month ahead forecast:        {vol_mean[19]:.4f}")
print(f"Long-run mean:                 {vol_mean[-1]:.4f}")

print("\n" + "="*70)
print("TRADING IMPLICATIONS")
print("="*70)
print(f"""
1. **Mean Reversion**:
   - Volatility forecasts converge to long-run mean (~{vol_mean[-1]:.4f})
   - If current vol > mean → expect decrease (sell volatility)
   - If current vol < mean → expect increase (buy volatility)

2. **Uncertainty Growth**:
   - Forecast uncertainty increases with horizon
   - Day 1 uncertainty: {forecast_std[0]:.4f}
   - Day 30 uncertainty: {forecast_std[29]:.4f}
   - Factor: {forecast_std[29]/forecast_std[0]:.1f}x higher

3. **Position Sizing**:
   - Scale position by 1/forecast_vol
   - High vol forecast → reduce position size
   - Low vol forecast → can increase position size

4. **Option Strategies**:
   - Compare forecast vol to implied vol
   - If forecast < implied → sell options (vol is overpriced)
   - If forecast > implied → buy options (vol is underpriced)

5. **Stop-Loss Adjustment**:
   - Stop distance ∝ forecast_vol
   - High vol → wider stops (avoid noise-triggered exits)
   - Low vol → tighter stops (protect gains)
""")
```

6 . Summary: Uncertainty Quantification in Practice

6 . 1 Key Takeaways

Concept	Why It Matters
Epistemic vs Aleatoric	Know what uncertainty you can reduce (get more data) vs what's irreducible (randomness)
GARCH	Volatility clusters—use GARCH to capture time-varying variance
Stochastic Volatility	Volatility has its own random shocks—better for option pricing and tail risk
Bayesian approach	Parameter uncertainty propagates to forecasts → more honest risk estimates
VaR vs CVaR	CVaR captures tail risk better—use it for position limits
Volatility forecasting	Scale positions inversely with forecast vol

6 . 2 Model Selection Guide

Use GARCH when:

- ✅ Need computationally fast forecasts
- ✅ Primarily interested in point volatility forecasts
- ✅ High-frequency trading (GARCH updates quickly)

Use Stochastic Volatility when:

- ✅ Pricing options (needs realistic vol dynamics)
- ✅ Risk management (want full distribution of future vol)
- ✅ Modeling volatility of volatility matters

6 . 3 Practical Workflow

1. **Model volatility** (GARCH or SV)
2. **Forecast multi-step ahead**
3. **Calculate VaR/CVaR** from posterior predictive
4. **Size positions**: $w_t = \frac{ 1 }{\text{CVaR}_t}$ (Kelly-like)
5. **Set stop-losses**: $\text{Stop distance} = 2 \times \text{forecast vol}$
6. **Rebalance**: Update forecasts daily/weekly

6 . 4 Common Pitfalls

❌ **Assuming constant volatility**

- Use time-varying vol models (GARCH/SV)

❌ **Ignoring parameter uncertainty**

- Bayesian approach accounts for this automatically

❌ **Using only VaR**

- VaR doesn't capture tail risk—use CVaR

❌ **Over-relying on historical volatility**

- Markets have regime changes—use priors that allow for this

❌ **Not updating forecasts**

- Volatility changes—refit models regularly (weekly for commodities)

Knowledge Check Quiz

Q 1 : Epistemic uncertainty can be reduced by:

- A) Collecting more data
- B) Using better risk management
- C) Diversification
- D) It cannot be reduced

Q 2 : In GARCH(1 , 1), high $\alpha + \beta$ (close to 1) means:

- A) Volatility changes very quickly
- B) Volatility shocks are persistent
- C) The model is misspecified
- D) Returns are normally distributed

Q 3 : CVaR is better than VaR for risk management because:

- A) It's easier to calculate
- B) It captures the average loss in the tail, not just the threshold
- C) It's always smaller than VaR
- D) Regulators don't require it

Q 4 : Stochastic Volatility models differ from GARCH by:

- A) Being faster to estimate
- B) Having separate random shocks for volatility
- C) Not requiring MCMC
- D) Always fitting better

Q 5 : If a commodity has high forecasted volatility, you should:

- A) Increase position size (more opportunity)
- B) Decrease position size (more risk)
- C) Keep position size constant
- D) Exit all positions immediately

In [ ]:
```
# Quiz Answers
print("="*70)
print("QUIZ ANSWERS")
print("="*70)
print("""
Q1: A) Collecting more data
    Epistemic uncertainty is parameter/model uncertainty. More data
    → tighter posterior → less epistemic uncertainty. Aleatoric
    uncertainty (market randomness) cannot be reduced.

Q2: B) Volatility shocks are persistent
    α + β is the persistence parameter. Close to 1 means volatility
    shocks decay very slowly (integrated GARCH). Typical for financial
    time series where high-vol periods last for weeks/months.

Q3: B) It captures the average loss in the tail, not just the threshold
    VaR only tells you the threshold (e.g., 5th percentile). CVaR
    tells you the average of all losses worse than VaR. This is much
    more useful for risk management and is a coherent risk measure.

Q4: B) Having separate random shocks for volatility
    GARCH: σ²_t is a deterministic function of past returns
    SV: h_t has its own random shock η_t
    This makes SV more flexible and better for option pricing.

Q5: B) Decrease position size (more risk)
    Higher volatility = higher risk. Position sizing should be
    inversely proportional to volatility: size ∝ 1/σ_forecast.
    This is the foundation of risk parity and Kelly criterion.
""")
```

Exercises

Exercise 1 : GARCH Parameter Sensitivity (Easy)

Simulate GARCH(1 , 1) with different parameter values:

- High α, low β (volatile but mean-reverting)
- Low α, high β (smooth but persistent)
- α + β = 0.99 (nearly integrated)

Compare volatility dynamics and forecast accuracy.

Exercise 2 : VaR Backtesting (Medium)

1. Generate synthetic returns with time-varying vol
2. Calculate daily VaR forecasts (95 % confidence)
3. Count how many days actual losses exceed VaR
4. Should be ~ 5 % for well-calibrated forecasts
5. Test with constant vol vs GARCH vol

Exercise 3: Volatility Trading Strategy (Hard)

Build a strategy that trades based on vol forecasts:

1. Forecast volatility using SV model
2. Compare to realized volatility
3. Trade signals:
    - If forecast_vol > realized_vol → reduce position
    - If forecast_vol < realized_vol → increase position
4. Backtest and calculate Sharpe ratio
5. Compare to constant position size

Exercise 4: Option Pricing with SV (Advanced)

Use the SV model to price European call options:

1. Simulate future price paths from SV model
2. Calculate option payoffs: max(S_T - K, 0)
3. Discount to present value
4. Compare to Black-Scholes (constant vol)
5. Analyze implied volatility smile

Next Module Preview

In **Module 10: Backtesting, Evaluation, and Trading Strategies**, we'll bring everything tc

- Walk-forward validation for Bayesian models
- Proper backtesting (avoiding look-ahead bias)
- CRPS and calibration assessment
- Trading strategies: mean reversion, trend following, pairs trading
- Portfolio optimization with Bayesian returns
- Complete energy portfolio trading system

Module 9 Complete