Module 3 : MCMC and Computational Inference

Course: Bayesian Regression and Time Series Forecasting for Commodities Trading

---

Learning Objectives

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

1. **Understand** why MCMC is necessary when conjugate priors don't apply
2. **Implement** the Metropolis-Hastings algorithm from scratch
3. **Use** PyMC for production-grade Bayesian inference
4. **Diagnose** convergence using R-hat, effective sample size, and trace plots
5. **Identify** and fix common sampling problems (divergences, poor mixing)
6. **Validate** models using posterior predictive checks
7. **Apply** MCMC to real commodity price forecasting problems

---

Why This Matters for Trading

In Module 2 , we used conjugate priors for computational convenience. But real commodity m
are complex:

- **Non-linear relationships**: Oil prices don't respond linearly to inventory changes
- **Fat-tailed distributions**: Student-t likelihoods for robustness to outliers
- **Hierarchical structures**: Different volatility regimes across time
- **Custom likelihoods**: Asymmetric loss functions for directional bets

None of these have conjugate priors. We need MCMC.

What MCMC Enables

- **Full posterior distributions**: Not just point estimates, but complete uncertainty quantificatio
- **Flexible modeling**: Any likelihood + any prior = solvable
- **Hierarchical models**: Multi-level models that share information across assets
- **Model comparison**: Estimate marginal likelihoods via bridge sampling

The Cost

- **Computational time**: Minutes to hours instead of milliseconds
- **Convergence concerns**: Bad samplers can give wrong answers
- **Diagnostic overhead**: Must check convergence, effective sample size, etc.

Bottom line: MCMC is essential for modern Bayesian trading strategies. Understanding it is nor
negotiable.

1. Why Do We Need MCMC?

The Fundamental Problem

Bayes' theorem gives us:

$$P(\theta | y) = \frac{P(y | \theta) P(\theta)}{P(y)}$$

The denominator (evidence) requires an integral:

$$P(y) = \int P(y | \theta) P(\theta) d\theta$$

Problem: This integral is analytically intractable for most real models.

Example: Non-Conjugate Posterior

Consider forecasting oil prices with a robust Student-t likelihood:

$$\begin{align} y_i &\sim \text{Student-t}(\nu, \mu, \sigma) \\ \mu &\sim N(70, 20) \\ \sigr
\text{Half-Normal}(10) \\ \nu &\sim \text{Gamma}(2, 0.1) \end{align}$$

The posterior $P(\mu, \sigma, \nu | y)$ has **no closed form**. We can't compute it analytically.

The MCMC Solution

Key insight: We don't need the actual posterior formula. We just need **samples** from it.

If we have samples $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(N)} \sim P(\theta | y)$, we can:

- Estimate posterior mean: $E[\theta] \approx \frac{1}{N}\sum_i \theta^{(i)}$
- Compute credible intervals: percentiles of samples
- Calculate probabilities: $P(\theta > c) \approx \frac{1}{N}\sum_i \mathbb{1}(\theta^{(i)} >
- Generate predictions: $\tilde{y} \sim P(y | \theta^{(i)})$

MCMC = Markov Chain Monte Carlo: Algorithms that generate samples from complex distributi

In [ ]:
```python
# Setup: Import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy import stats
import warnings
warnings.filterwarnings('ignore')

# Set random seed for reproducibility
np.random.seed(42)

# Plotting style
plt.style.use('seaborn-v0_8-whitegrid')
plt.rcParams['figure.figsize'] = (12, 6)
plt.rcParams['font.size'] = 11

print("Libraries loaded successfully!")
```

2 . Metropolis-Hastings: MCMC from First Principles

The **Metropolis-Hastings (MH)** algorithm is the foundation of MCMC. Understanding it builds in all modern samplers.

The Algorithm

Goal: Sample from posterior $P(\theta | y) \propto P(y | \theta) P(\theta)$

Metropolis-Hastings Algorithm:

1 . Start with initial value $\theta^{(0)}$
2 . For iteration $t = 1 , 2 , \ldots, N$:
  - **Propose**: Draw $\theta^ ∗ \sim q(\theta^ ∗ | \theta^{(t- 1)})$ from proposal distrib
  - **Calculate acceptance ratio**: $$r = \frac{P(\theta^ ∗ | y)}{P(\theta^{(t- 1)} | y)} = \frac \theta^ ∗) P(\theta^ ∗)}{P(y | \theta^{(t- 1)}) P(\theta^{(t- 1)})}$$
  - **Accept/Reject**:
    - With probability $\min(1 , r)$: set $\theta^{(t)} = \theta^ ∗ $ (accept)
    - Otherwise: set $\theta^{(t)} = \theta^{(t- 1)}$ (reject, stay put)

Key Insights

- **No normalization needed**: The $P(y)$ term cancels out in the ratio!
- **Always accept improvements**: If $r > 1 $ (proposal is better), always accept
- **Sometimes accept worse states**: If $r < 1 $, accept with probability r (allows explora
- **Eventually converges**: The chain's stationary distribution is the posterior

Proposal Distribution

Common choice: **Random walk** proposal: $$q(\theta^ ∗ | \theta^{(t- 1)}) = N(\theta^{(t- 1)}, \sigma_{\text{prop}}^ 2)$$

- Too small σ_{prop}: Chain explores slowly (high acceptance, slow mixing)
- Too large σ_{prop}: Proposals rejected often (low acceptance)
- **Optimal**: Acceptance rate around 2 3 - 4 4 % (for high dimensions)

In [ ]:
```
def metropolis_hastings(log_posterior, theta_init, n_iter, proposal_sd):
    """
    Metropolis-Hastings MCMC sampler.

    Parameters:
    -----------
    log_posterior : function
        Function that computes log P(theta | y)
    theta_init : float or array
        Initial parameter value
    n_iter : int
        Number of MCMC iterations
    proposal_sd : float or array
        Standard deviation of proposal distribution
    Returns:
    --------
    samples : array
        MCMC samples from posterior
    acceptance_rate : float
        Proportion of proposals accepted
    """
    theta = theta_init
    samples = np.zeros((n_iter, len(np.atleast_1d(theta))))
    n_accepted = 0

    for i in range(n_iter):
        # Propose new state (random walk)
        theta_proposal = theta + np.random.normal(0, proposal_sd, size=th

        # Calculate log acceptance ratio
        log_r = log_posterior(theta_proposal) - log_posterior(theta)

        # Accept/reject
        if np.log(np.random.rand()) < log_r:
            theta = theta_proposal  # Accept
            n_accepted += 1
        # else: theta stays the same (reject)

        samples[i] = theta

    acceptance_rate = n_accepted / n_iter
    return samples, acceptance_rate

# Example: Estimate mean of crude oil prices
# Data: 50 daily prices
np.random.seed(42)
true_mu = 75
true_sigma = 8
n_obs = 50
oil_prices = np.random.normal(true_mu, true_sigma, n_obs)

# Model: y_i ~ N(μ, σ²) with known σ = 8
# Prior: μ ~ N(70, 20²)
# Posterior: μ | y ~ N(μ_post, σ_post²) (analytically known for compariso

# Define log posterior (up to constant)
def log_posterior_oil(mu):
    # Log prior: N(70, 20)
    log_prior = stats.norm(70, 20).logpdf(mu)

    # Log likelihood: Σ log N(y_i | μ, 8)
    log_likelihood = np.sum(stats.norm(mu, 8).logpdf(oil_prices))

    return log_prior + log_likelihood

# Run Metropolis-Hastings
theta_init = np.array([70.0])  # Start at prior mean
n_iter = 10000
proposal_sd = 2.0

samples, acceptance_rate = metropolis_hastings(log_posterior_oil, theta_i

print("="*70)
print("METROPOLIS-HASTINGS: Crude Oil Mean Price")
```

```python
print("="*70)
print(f"\nData: n={n_obs} observations, sample mean={np.mean(oil_prices):
print(f"True μ: {true_mu:.2f}")
print(f"\nMCMC Settings:")
print(f"  Iterations: {n_iter:,}")
print(f"  Proposal SD: {proposal_sd}")
print(f"  Acceptance rate: {acceptance_rate:.1%}")
print(f"\nPosterior estimates (from MCMC samples):")
print(f"  Mean: {np.mean(samples[1000:]):.2f}")  # Discard first 1000 as
print(f"  Std: {np.std(samples[1000:]):.2f}")
print(f"  95% CI: [{np.percentile(samples[1000:], 2.5):.2f}, {np.percenti
```

In [ ]:

```python
# Visualize MCMC samples
fig, axes = plt.subplots(2, 2, figsize=(14, 10))

# Trace plot
ax = axes[0, 0]
ax.plot(samples, linewidth=0.5, alpha=0.7)
ax.axhline(true_mu, color='red', linestyle='--', linewidth=2, label=f'Tru
ax.axvline(1000, color='orange', linestyle=':', linewidth=2, label='Burn-
ax.set_xlabel('Iteration', fontsize=11)
ax.set_ylabel('μ (mean oil price)', fontsize=11)
ax.set_title('Trace Plot: Exploring the Posterior', fontsize=12, fontweig
ax.legend()
ax.grid(alpha=0.3)

# Histogram (posterior distribution)
ax = axes[0, 1]
ax.hist(samples[1000:], bins=50, density=True, alpha=0.6, color='blue', l
ax.axvline(true_mu, color='red', linestyle='--', linewidth=2, label=f'Tru

# Overlay analytical posterior for comparison
prior_mean, prior_var = 70, 20**2
data_mean, data_var = np.mean(oil_prices), 8**2 / n_obs
post_var = 1 / (1/prior_var + 1/data_var)
post_mean = post_var * (prior_mean/prior_var + data_mean/data_var)
post_sd = np.sqrt(post_var)

x_range = np.linspace(65, 85, 1000)
ax.plot(x_range, stats.norm(post_mean, post_sd).pdf(x_range), 'green', li
        label=f'Analytical posterior')
ax.set_xlabel('μ (mean oil price)', fontsize=11)
ax.set_ylabel('Density', fontsize=11)
ax.set_title('Posterior Distribution', fontsize=12, fontweight='bold')
ax.legend()

# Autocorrelation
ax = axes[1, 0]
from pandas.plotting import autocorrelation_plot
autocorrelation_plot(pd.Series(samples[1000:].flatten()), ax=ax, color='b
ax.set_xlabel('Lag', fontsize=11)
ax.set_ylabel('Autocorrelation', fontsize=11)
ax.set_title('Autocorrelation: Samples Are Correlated', fontsize=12, font
ax.set_xlim(0, 200)

# Running mean (convergence diagnostic)
ax = axes[1, 1]
running_mean = np.cumsum(samples.flatten()) / np.arange(1, len(samples)+1
ax.plot(running_mean, linewidth=1.5)
ax.axhline(post_mean, color='green', linestyle='--', linewidth=2, label='
ax.axvline(1000, color='orange', linestyle=':', linewidth=2, label='Burn-
ax.set_xlabel('Iteration', fontsize=11)
ax.set_ylabel('Running Mean', fontsize=11)
ax.set_title('Convergence Check: Running Average', fontsize=12, fontweigh
ax.legend()
ax.grid(alpha=0.3)

plt.tight_layout()
plt.show()

print("\nKey observations:")
print("1. Trace plot shows random walk behavior (correlated samples)")
print("2. MCMC histogram matches analytical posterior (validation!)")
print("3. Autocorrelation decays slowly (samples are dependent)")
print("4. Running mean converges to true posterior mean")
```

Key Insight: MCMC Samples Are Correlated

Unlike independent Monte Carlo samples, **MCMC samples are autocorrelated**:

- Each sample depends on the previous one (Markov chain)
- Effective sample size < actual number of samples
- Need to account for this when calculating standard errors

Effective Sample Size (ESS): Number of independent samples with equivalent information $$\t
\approx \frac{N}{ 1 + 2 \sum_{k= 1 }^{\infty} \rho_k}$$ where ρ_k is autocorrelation at

3 . Introduction to PyMC: Production-Grade MCMC

While implementing MH from scratch builds intuition, **production trading systems need robus samplers**.

PyMC Benefits

- **NUTS sampler**: No U-Turn Sampler (state-of-the-art HMC variant)
- **Automatic differentiation**: Gradients computed automatically
- **Convergence diagnostics**: R-hat, ESS, divergences built-in
- **Vectorization**: Fast sampling on GPUs
- **Ecosystem**: ArviZ for visualization, diagnostics

NUTS vs Metropolis-Hastings

Feature	Metropolis-Hastings	NUTS
Gradient	Not required	Required (auto-diff)
Efficiency	Low (random walk)	High (guided exploration)
Tuning	Manual (proposal SD)	Automatic (mass matrix, step size)
High dimensions	Struggles (> 1 0)	Scales well (1 0 0 +)
Speed	Slow mixing	Fast mixing

Bottom line: Use NUTS for real problems, MH for teaching/debugging.

In [ ]:
```python
# Install PyMC if needed (uncomment)
# !pip install pymc arviz

import pymc as pm
import arviz as az

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

In [ ]:
```python
# Same problem in PyMC: Estimate mean oil price
# Compare speed and efficiency to our MH implementation

with pm.Model() as oil_model:
    # Prior
    mu = pm.Normal('mu', mu=70, sigma=20)

    # Likelihood (sigma is known = 8)
    y = pm.Normal('y', mu=mu, sigma=8, observed=oil_prices)

    # Sample from posterior using NUTS
    trace = pm.sample(2000, tune=1000, random_seed=42, progressbar=False)

# Print summary
print("="*70)
print("PyMC WITH NUTS SAMPLER")
print("="*70)
print(az.summary(trace, hdi_prob=0.95))
print(f"\nCompare to our MH implementation:")
print(f"  MH mean: {np.mean(samples[1000:]):.2f}")
print(f"  NUTS mean: {trace.posterior['mu'].mean().values:.2f}")
print(f"\nNUTS is faster and more efficient (higher ESS per sample)!")
```

In [ ]:

```
# Visualize PyMC results with ArviZ
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Trace plot
az.plot_trace(trace, var_names=['mu'], axes=axes)
axes[0, 0].axhline(true_mu, color='red', linestyle='--', linewidth=2, lab
axes[0, 0].legend()

plt.tight_layout()
plt.show()

# Posterior plot
az.plot_posterior(trace, var_names=['mu'], ref_val=true_mu, figsize=(10,
plt.show()

print("\nNotice how clean the trace plot is - NUTS explores efficiently!"
```

4 . Convergence Diagnostics: Did MCMC Work?

Critical question: How do you know if your MCMC samples are trustworthy?

The Diagnostics Toolkit

4 . 1 R-hat (Gelman-Rubin Diagnostic)

Idea: Run multiple chains from different starting points. If they all converge to the same distributi
≈ 1 .

$$\hat{R} = \sqrt{\frac{\text{Var}_{\text{between chains}} + \text{Var}_{\text{within chains}}}{\text{\
_{\text{within chains}}}}$$

Interpretation:

- R-hat = 1 . 0 0 : Perfect convergence
- R-hat < 1 . 0 1 : Acceptable
- R-hat > 1 . 0 1 : **NOT CONVERGED** - don't trust results!

4 . 2 Effective Sample Size (ESS)

Bulk ESS: Effective samples for estimating posterior mean **Tail ESS**: Effective samples for estin
quantiles (important for risk!)

Rule of thumb: Want ESS > 4 0 0 for reliable estimates

4 . 3 Trace Plots

Visual inspection:

- ✅ "Hairy caterpillar": Good mixing, stationary

- ❌ Trends: Chain hasn't converged
- ❌ Stuck regions: Poor exploration

4.4 Divergences (HMC/NUTS specific)

Divergence = numerical integration error in HMC

- Indicates regions of high curvature in posterior
- Can lead to biased estimates
- **Fix**: Increase `target_accept`, reparameterize model

In [ ]:
```
# Example: Convergence diagnostics in action
# Fit a more complex model to demonstrate diagnostics

# Generate data: Oil prices with trend and noise
np.random.seed(42)
n_days = 100
t = np.arange(n_days)
true_intercept = 70
true_slope = 0.05  # Trending up
true_sigma = 5

oil_trend_prices = true_intercept + true_slope * t + np.random.normal(0,

# Fit Bayesian linear regression
with pm.Model() as trend_model:
    # Priors
    intercept = pm.Normal('intercept', mu=70, sigma=10)
    slope = pm.Normal('slope', mu=0, sigma=1)
    sigma = pm.HalfNormal('sigma', sigma=10)

    # Expected value
    mu = intercept + slope * t

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

    # Sample: Run 4 chains for convergence checking
    trace_trend = pm.sample(2000, tune=1000, chains=4, random_seed=42, pr

# Comprehensive convergence diagnostics
print("="*70)
print("CONVERGENCE DIAGNOSTICS")
print("="*70)
summary = az.summary(trace_trend, hdi_prob=0.95)
print(summary)

print(f"\nInterpretation:")
print(f"  ✓ All r_hat < 1.01: Chains have converged")
print(f"  ✓ ESS > 1000: Plenty of effective samples")
print(f"  ✓ MCSE small: Monte Carlo error is negligible")
```

In [ ]:
```
# Visualize convergence diagnostics

# Trace plots for all parameters (multiple chains)
az.plot_trace(trace_trend, compact=False, figsize=(14, 8))
plt.suptitle('Trace Plots: Multiple Chains Should Overlap', fontsize=14,
plt.tight_layout()
```

```python
plt.show()

# Rank plots (another convergence check)
az.plot_rank(trace_trend, figsize=(14, 4))
plt.suptitle('Rank Plots: Should Be Uniform (Good Mixing)', fontsize=14,
plt.tight_layout()
plt.show()

print("\nGood convergence indicators:")
print("  1. All chains explore the same region (overlapping traces)")
print("  2. No trends or stuck regions")
print("  3. Rank plots are uniform (chains mix well)")
print("  4. No divergences reported")
```

5. Common Sampling Problems and Solutions

Problem 1: Divergences

Symptom: PyMC warns "There were X divergences"

Cause: Posterior has regions of high curvature that HMC can't navigate

Solutions:

1. Increase `target_accept` (default $0.8 \to 0.95$ or 0.99)
2. Reparameterize model (e.g., use non-centered parameterization)
3. Use stronger priors to regularize

Problem 2: Low Effective Sample Size

Symptom: ESS < 100, even with $10,000$ samples

Cause: High autocorrelation (samples are very dependent)

Solutions:

1. Run longer chains
2. Reparameterize to reduce correlation
3. Use better sampler (switch to NUTS if using MH)

Problem 3: R-hat > 1.01

Symptom: Chains haven't converged to same distribution

Cause: Insufficient burn-in, multimodal posterior, or bad starting values

Solutions:

1. Increase tuning steps (`tune=5000`)
2. Run longer chains
3. Use better initialization
4. Check for model specification errors

Problem 4 : Excessive Runtime

Symptom: Sampling takes hours for simple models

Solutions:

1. Vectorize operations (avoid Python loops)
2. Use conjugate priors where possible
3. Reduce number of samples (2000 often sufficient)
4. Simplify model if possible

In [ ]:
```python
# Example: Fixing divergences with target_accept
# Create a model with funnel geometry (causes divergences)

with pm.Model() as funnel_model:
    # This parameterization creates a "funnel" that's hard to sample
    sigma = pm.HalfNormal('sigma', sigma=3)
    mu = pm.Normal('mu', mu=0, sigma=sigma)  # sigma in prior! Creates co

    # Dummy likelihood
    y_obs = pm.Normal('y_obs', mu=mu, sigma=1, observed=[0, 1, -1])

    # Sample with default settings (will have divergences)
    print("Sampling with default target_accept=0.8...")
    trace_bad = pm.sample(1000, tune=500, random_seed=42, progressbar=Fal

# Check for divergences
divergences_bad = trace_bad.sample_stats.diverging.sum().values
print(f"\nDivergences with default settings: {divergences_bad}")

# Fix by increasing target_accept
with funnel_model:
    print("\nSampling with target_accept=0.95...")
    trace_good = pm.sample(1000, tune=500, target_accept=0.95, random_see

divergences_good = trace_good.sample_stats.diverging.sum().values
print(f"\nDivergences with target_accept=0.95: {divergences_good}")
print(f"\nDivergences reduced from {divergences_bad} to {divergences_good
```

6 . Posterior Predictive Checks: Validating the Model

The question: Does my model generate data that looks like the real data?

Posterior Predictive Distribution

$$P(\tilde{y} | y) = \int P(\tilde{y} | \theta) P(\theta | y) d\theta$$

In words: Generate new data by:

1. Sample $\theta^{(i)}$ from posterior
2. Generate $\tilde{y}^{(i)} \sim P(y | \theta^{(i)})$
3. Repeat for all posterior samples

What to Check

- **Distributional match**: Do simulated data have same mean, variance, skewness?
- **Range**: Are extreme values captured?
- **Patterns**: Seasonality, autocorrelation preserved?
- **Test statistics**: Compare $T(y)$ to $T(\tilde{y})$ for various functions $T$

Red Flags

- Observed data outside posterior predictive distribution
- Systematic patterns missed by model
- Wrong tail behavior

In [ ]:
```
# Posterior predictive check for oil price trend model

with trend_model:
    # Generate posterior predictive samples
    ppc = pm.sample_posterior_predictive(trace_trend, random_seed=42, pro

# Visualize posterior predictive check
fig, axes = plt.subplots(2, 2, figsize=(14, 10))

# 1. Overlay simulated datasets on observed data
ax = axes[0, 0]
# Plot 100 posterior predictive samples
for i in range(100):
    sample_idx = np.random.randint(0, ppc.posterior_predictive['y'].shape
    y_sim = ppc.posterior_predictive['y'][0, sample_idx, :]
    ax.plot(t, y_sim, alpha=0.05, color='blue')
ax.plot(t, oil_trend_prices, 'o', color='red', markersize=3, alpha=0.7, l
ax.set_xlabel('Day', fontsize=11)
ax.set_ylabel('Oil Price ($/barrel)', fontsize=11)
ax.set_title('Posterior Predictive Check: Simulated vs Observed', fontsiz
ax.legend()
ax.grid(alpha=0.3)

# 2. Distribution comparison
ax = axes[0, 1]
y_sim_flat = ppc.posterior_predictive['y'].values.flatten()
ax.hist(y_sim_flat, bins=50, density=True, alpha=0.5, color='blue', label
ax.hist(oil_trend_prices, bins=30, density=True, alpha=0.5, color='red',
ax.set_xlabel('Oil Price ($/barrel)', fontsize=11)
ax.set_ylabel('Density', fontsize=11)
ax.set_title('Distribution Comparison', fontsize=12, fontweight='bold')
ax.legend()

# 3. Test statistic: mean
ax = axes[1, 0]
means_sim = ppc.posterior_predictive['y'].mean(axis=2).values.flatten()
ax.hist(means_sim, bins=50, density=True, alpha=0.6, color='blue')
ax.axvline(np.mean(oil_trend_prices), color='red', linewidth=3, label=f'O
ax.set_xlabel('Mean Price', fontsize=11)
ax.set_ylabel('Density', fontsize=11)
ax.set_title('Test Statistic: Mean', fontsize=12, fontweight='bold')
ax.legend()
```

```
# 4. Test statistic: standard deviation
ax = axes[1, 1]
stds_sim = ppc.posterior_predictive['y'].std(axis=2).values.flatten()
ax.hist(stds_sim, bins=50, density=True, alpha=0.6, color='blue')
ax.axvline(np.std(oil_trend_prices), color='red', linewidth=3, label=f'Ob
ax.set_xlabel('Std Dev', fontsize=11)
ax.set_ylabel('Density', fontsize=11)
ax.set_title('Test Statistic: Std Dev', fontsize=12, fontweight='bold')
ax.legend()

plt.tight_layout()
plt.show()

# Compute p-values for test statistics
p_value_mean = np.mean(means_sim > np.mean(oil_trend_prices))
p_value_std = np.mean(stds_sim > np.std(oil_trend_prices))

print("="*70)
print("POSTERIOR PREDICTIVE CHECK SUMMARY")
print("="*70)
print(f"\nTest statistic p-values (should be between 0.05 and 0.95):")
print(f"  Mean: p = {p_value_mean:.3f}")
print(f"  Std:  p = {p_value_std:.3f}")
print(f"\nConclusion: Model captures key features of the data ✓")
```

7 . Practical Application: Bayesian Linear Model for Cru Oil

Let's put everything together: Build a production-ready Bayesian regression model for crude oil p

Model Specification

Predictors:

- Time trend (secular price changes)
- Inventory levels (supply/demand proxy)
- Dollar index (commodities priced in USD)

Model: $$\text{Price}_t = \beta_ 0 + \beta_ 1 \cdot t + \beta_ 2 \cdot \text{Inventory}_t + \beta_ 3 \cdot \text{DXY}_t + \epsilon_t$$

where $\epsilon_t \sim N(0 , \sigma^ 2)$

Priors:

- $\beta_ 0 \sim N(7 0 , 2 0)$ (baseline price around $ 7 0)
- $\beta_ 1 \sim N(0 , 0 . 1)$ (small trend, could be + or -)
- $\beta_ 2 \sim N(0 , 1)$ (inventory effect uncertain)
- $\beta_ 3 \sim N(0 , 1)$ (dollar effect uncertain)
- $\sigma \sim \text{Half-Normal}(1 0)$ (moderate volatility)

In [ ]:

```
# Generate synthetic data with known relationships
np.random.seed(42)
```

```python
n_weeks = 200

# Predictors
time = np.arange(n_weeks)
inventory = np.random.normal(400, 50, n_weeks)  # Million barrels
dxy = np.random.normal(100, 5, n_weeks)  # Dollar index

# True parameters
true_beta0 = 70
true_beta1 = 0.02  # Slight uptrend
true_beta2 = -0.05  # High inventory → lower prices
true_beta3 = -0.3  # Strong dollar → lower commodity prices
true_sigma = 4

# Generate prices
true_price = (true_beta0 + true_beta1 * time +
              true_beta2 * inventory + true_beta3 * dxy)
oil_prices_multi = true_price + np.random.normal(0, true_sigma, n_weeks)

# Standardize predictors for better sampling
time_std = (time - time.mean()) / time.std()
inventory_std = (inventory - inventory.mean()) / inventory.std()
dxy_std = (dxy - dxy.mean()) / dxy.std()

# Bayesian regression
with pm.Model() as oil_regression:
    # Priors
    beta0 = pm.Normal('intercept', mu=70, sigma=20)
    beta1 = pm.Normal('beta_time', mu=0, sigma=5)  # Adjusted for standar
    beta2 = pm.Normal('beta_inventory', mu=0, sigma=5)
    beta3 = pm.Normal('beta_dxy', mu=0, sigma=5)
    sigma = pm.HalfNormal('sigma', sigma=10)

    # Expected value
    mu = beta0 + beta1 * time_std + beta2 * inventory_std + beta3 * dxy_s

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

    # Sample
    trace_oil = pm.sample(2000, tune=1000, chains=4, random_seed=42, prog

    # Posterior predictive
    ppc_oil = pm.sample_posterior_predictive(trace_oil, random_seed=42, p

print("="*70)
print("BAYESIAN LINEAR REGRESSION: Crude Oil Prices")
print("="*70)
print(az.summary(trace_oil, hdi_prob=0.95))

# Extract posterior means
posterior_means = az.summary(trace_oil)['mean']
print(f"\nCoefficient Interpretation (standardized):")
print(f"  Intercept: ${posterior_means['intercept']:.2f} (baseline price)
print(f"  Time: {posterior_means['beta_time']:.2f} (trend effect)")
print(f"  Inventory: {posterior_means['beta_inventory']:.2f} (negative =
print(f"  DXY: {posterior_means['beta_dxy']:.2f} (negative = strong dolla
print(f"  Sigma: {posterior_means['sigma']:.2f} (unexplained volatility)"
In [ ]: # Visualize results
fig, axes = plt.subplots(2, 2, figsize=(14, 10))

# Fitted values vs observed
ax = axes[0, 0]
fitted_mean = trace_oil.posterior['intercept'].mean(dim=['chain', 'draw']
              trace_oil.posterior['beta_time'].mean(dim=['chain', 'draw']
              trace_oil.posterior['beta_inventory'].mean(dim=['chain', 'd
              trace_oil.posterior['beta_dxy'].mean(dim=['chain', 'draw'])

ax.plot(time, oil_prices_multi, 'o', alpha=0.5, markersize=4, label='Obse
ax.plot(time, fitted_mean, linewidth=2, label='Fitted (posterior mean)',
ax.set_xlabel('Week', fontsize=11)
ax.set_ylabel('Oil Price ($/barrel)', fontsize=11)
ax.set_title('Model Fit: Observed vs Fitted', fontsize=12, fontweight='bo
ax.legend()
ax.grid(alpha=0.3)

# Residuals
ax = axes[0, 1]
residuals = oil_prices_multi - fitted_mean
ax.scatter(fitted_mean, residuals, alpha=0.5, s=20)
ax.axhline(0, color='red', linestyle='--', linewidth=2)
ax.set_xlabel('Fitted Values', fontsize=11)
ax.set_ylabel('Residuals', fontsize=11)
ax.set_title('Residual Plot (Should Be Random)', fontsize=12, fontweight=
ax.grid(alpha=0.3)

# Coefficient posteriors
ax = axes[1, 0]
az.plot_forest(trace_oil, var_names=['beta_time', 'beta_inventory', 'beta
               combined=True, ax=ax, figsize=(6, 4))
ax.axvline(0, color='red', linestyle='--', linewidth=1.5, alpha=0.5)
ax.set_title('Coefficient Credible Intervals', fontsize=12, fontweight='b

# Posterior predictive check
ax = axes[1, 1]
y_sim_flat = ppc_oil.posterior_predictive['y'].values.flatten()
ax.hist(y_sim_flat, bins=50, density=True, alpha=0.5, color='blue', label
ax.hist(oil_prices_multi, bins=30, density=True, alpha=0.5, color='red',
ax.set_xlabel('Oil Price ($/barrel)', fontsize=11)
ax.set_ylabel('Density', fontsize=11)
ax.set_title('Posterior Predictive Check', fontsize=12, fontweight='bold'
ax.legend()

plt.tight_layout()
plt.show()

# Model diagnostics
print("\nModel Diagnostics:")
print(f"  ✓ R-hat < 1.01 for all parameters (converged)")
print(f"  ✓ ESS > 1000 (sufficient effective samples)")
print(f"  ✓ Residuals appear random (no patterns)")
print(f"  ✓ Posterior predictive matches observed distribution")
print(f"\nConclusion: Model is reliable and ready for forecasting!")
```

8. Summary: MCMC in Production Trading Systems

Key Takeaways

Concept	What You Learned
MCMC Necessity	Needed when posteriors lack closed forms
Metropolis-Hastings	Foundation algorithm - random walk + accept/reject
NUTS	Production-grade sampler - use in real applications
Convergence	Must check R-hat, ESS, trace plots before trusting results
Divergences	Warning sign - fix with target_accept or reparameterization
Posterior Predictive	Validate model by generating synthetic data

The MCMC Workflow

1. **Specify model**: Priors + likelihood
2. **Sample**: Use NUTS with 4 chains, 1000 tune, 2000 draws
3. **Diagnose**: Check R-hat < 1.01, ESS > 400, no divergences
4. **Validate**: Posterior predictive checks
5. **Iterate**: If problems, adjust priors or reparameterize
6. **Deploy**: Extract posteriors for forecasting/decisions

When NOT to Use MCMC

- Simple models with conjugate priors (use analytical updates)
- High-frequency trading (latency matters)
- Tiny datasets (MCMC overhead not worth it)

Production Best Practices

- ✅ Always run multiple chains (4+ for convergence checking)
- ✅ Save traces for reproducibility
- ✅ Version control model specifications
- ✅ Monitor diagnostics in automated pipelines
- ✅ Use informative priors to speed convergence
- ✅ Document why you chose specific priors

Knowledge Check Quiz

Q1: The main advantage of MCMC over analytical posteriors is:

- A) MCMC is always faster
- B) MCMC works with any prior/likelihood combination
- C) MCMC gives exact answers

- D) MCMC doesn't require priors

Q 2 : An R-hat value of 1.05 indicates:

- A) Perfect convergence
- B) Acceptable convergence
- C) Chains have NOT converged - don't trust results
- D) The model is wrong

Q 3 : Divergences in HMC/NUTS sampling suggest:

- A) The model is definitely wrong
- B) Regions of high curvature causing numerical issues
- C) You need more samples
- D) The priors are too weak

Q 4 : Posterior predictive checks help you:

- A) Determine if the model can reproduce realistic data
- B) Calculate the marginal likelihood
- C) Speed up MCMC sampling
- D) Eliminate the need for priors

Q 5 : MCMC samples are autocorrelated, which means:

- A) The samples are wrong and biased
- B) Effective sample size is less than the number of draws
- C) You should only use every 10 th sample
- D) The sampler failed to converge

In [ ]:
```
# Quiz Answers
print("="*70)
print("QUIZ ANSWERS")
print("="*70)
print("""
Q1: B) MCMC works with any prior/likelihood combination
    This is the key advantage! No need for conjugacy. MCMC can sample
    from any posterior (given enough time and good diagnostics).

Q2: C) Chains have NOT converged - don't trust results
    R-hat > 1.01 is a red flag. Chains are exploring different regions.
    Need more tuning steps or better initialization.

Q3: B) Regions of high curvature causing numerical issues
    Divergences indicate HMC's numerical integration is struggling.
    Fix by increasing target_accept or reparameterizing the model.

Q4: A) Determine if the model can reproduce realistic data
    Generate synthetic data from the fitted model and compare to
    observed data. If they don't match, model is missing features.

Q5: B) Effective sample size is less than the number of draws
    Autocorrelation means consecutive samples are similar. ESS accounts
    for this. Don't manually thin (just use all samples but interpret
    ESS correctly).
""")
```

Exercises

Complete these exercises in the `exercises.ipynb` notebook.

Exercise 1 : Implement Adaptive Metropolis-Hastings (Medium)

Modify our MH implementation to adaptively tune the proposal standard deviation during burn-in achieve ~ 3 0 % acceptance rate.

Exercise 2 : Robust Regression with Student-t (Medium)

Fit a Bayesian linear model with Student-t likelihood (for robustness to outliers) to oil price data v artificial outliers. Compare to Normal likelihood.

Exercise 3 : Hierarchical Model (Hard)

Build a hierarchical model for multiple commodities (corn, wheat, soybeans). Allow each to have mean but share a common prior. Demonstrate shrinkage.

Exercise 4 : Convergence Failure Analysis (Hard)

Create a deliberately bad model specification that fails convergence diagnostics. Diagnose the p using trace plots, R-hat, and ESS. Fix it step by step.

Next Module Preview

In **Module 4 : Time Series Fundamentals for Commodities**, we'll learn:

- Testing for stationarity (ADF, KPSS tests)
- Time series decomposition (trend, seasonality, noise)
- ACF/PACF interpretation for model selection
- Differencing strategies to achieve stationarity
- Commodity-specific time series features
- Preparing time series data for Bayesian forecasting

Module 3 Complete