Mathematical Documentation of a Market State Prediction Model

Kryštof Kouřil

March 26, 2025

1 Introduction

After experiencing working with machine learning models my mind yearned for a project more mathematically raw. I came across mentions of using Markov Chain Monte Carlo methods in finance. This combination seemed as a perfect are to explore. To make my work a bit more challenging, my friend with a background in investment analysis pointed out that for the model to be useful in real-world trading environments it has to be not only mathematically rigorous but also adaptable to changing market conditions.

This paper documents my attempt to build such a framework. I've developed a market prediction model that classifies next-day S&P 500 movements into five discrete states using adaptive thresholds and Bayesian inference with Markov Chain Monte Carlo (MCMC) sampling. What makes this interesting (at least to me) is how the model combines statistical methods with financial domain knowledge.

The key components I'll discuss include:

- Market regime detection through volatility clustering
- Skewed t-distribution fitting for non-normal returns
- Dynamic threshold calculation based on current market conditions
- Bayesian inference using replica exchange MCMC

My code implementation focuses on accuracy and numerical stability rather than optimization - it's probably not the fastest algorithm, but it handles the inherent uncertainty in financial prediction in a mathematically sound way.

2 Market Regime Modeling

2.1 Regime Detection Framework

The foundation of my approach is the recognition that financial markets operate in distinct regimes with different statistical properties. Following the work of

[1] and [2], I implemented a regime detection function that categorizes market conditions into four states: crisis, high volatility, normal, and low volatility.

Rather than using a Hidden Markov Model as in many papers, I opted for a more direct approach based on observable metrics. For each time t, I calculate a set of rolling statistics over a window of w days:

$$\sigma_t = \sqrt{252} \cdot \text{StdDev}(r_{t-w+1:t}) \tag{1}$$

$$\sigma_{\sigma,t} = \text{StdDev}(\sigma_{t-w+1:t}) \tag{2}$$

$$S_t = \text{Skewness}(r_{t-w+1:t}) \tag{3}$$

$$K_t = \text{Kurtosis}(r_{t-w+1:t}) \tag{4}$$

I also compute trend indicators to capture momentum effects:

$$\mu_{short} = \text{Mean}(r_{t-10:t}) \tag{5}$$

$$\mu_{medium} = \text{Mean}(r_{t-30:t}) \tag{6}$$

$$\text{trend}_{strength} = \frac{|\mu_{short}|}{\sigma_t} \tag{7}$$

The regime detection logic uses dynamic thresholds based on historical percentiles over a lookback period L:

$$\tau_t^{crisis} = \text{Percentile}_{95}(\sigma_{t-L:t-1}) \tag{8}$$

$$\tau_t^{high} = \text{Percentile}_{75}(\sigma_{t-L:t-1}) \tag{9}$$

$$\tau_t^{low} = \text{Percentile}_{25}(\sigma_{t-L:t-1}) \tag{10}$$

The core function `detect_market_regime()` assigns regimes using the following conditions:

Algorithm 1 Market Regime Detection

```
1: for each time $t$ do
2:     if $\sigma_t > \tau_t^{crisis}$ and $\sigma_{\sigma,t} > 1.8 \cdot \mu_{\sigma_\sigma,t}$ and $(S_t < S_{threshold}$ or $K_t > K_{threshold}$ or $\text{trend}_{strength} > 1.5)$ then
3:         $regime_t \leftarrow$ crisis
4:     else if $\sigma_t > \tau_t^{high}$ and $(\sigma_{\sigma,t} > 0.8 \cdot \mu_{\sigma_\sigma,t}$ or $\text{trend}_{strength} > 1.0)$ then
5:         $regime_t \leftarrow$ high_vol
6:     else if $\sigma_t < \tau_t^{low}$ and $\sigma_{\sigma,t} < 0.6 \cdot \mu_{\sigma_\sigma,t}$ and $|mean\_reversion| < 0.5$ then
7:         $regime_t \leftarrow$ low_vol
8:     else
9:         $regime_t \leftarrow$ normal
10:    end if
11: end for
```

This approach differs from standard Markov switching models in that it directly uses observable market features rather than latent states. The thresholds are dynamically adjusted based on recent history, making the regime detection adaptive to changing market conditions.

One challenge I faced was determining appropriate scaling factors for the conditions. Through experimentation, I found that the volatility of volatility (vol-of-vol) is particularly important during crisis periods, while trend strength matters more in high volatility regimes. The specific multipliers (1.8, 0.8, 0.6, etc.) were chosen based on backtesting performance, though I'd love to find a more principled way to set these in the future.

2.2 Skewed t-Distribution Modeling

Financial returns typically exhibit fat tails and skewness, making normal distributions a poor fit. After trying several alternatives, I settled on the skewed t-distribution introduced by [3] as it captures both these properties.

For each regime r, I fit a skewed t-distribution with parameters $\theta_r = (\mu_r, \sigma_r, \nu_r, \lambda_r)$ representing location, scale, degrees of freedom, and skewness. The probability density function is:

$$f(x;\mu,\sigma,\nu,\lambda) = \begin{cases} \frac{bc}{\sigma}(1+\frac{1}{\nu-2}(\frac{bz+a}{1-\lambda})^2)^{-\frac{\nu+1}{2}}, & \text{if } z < -\frac{a}{b} \\ \frac{bc}{\sigma}(1+\frac{1}{\nu-2}(\frac{bz+a}{1+\lambda})^2)^{-\frac{\nu+1}{2}}, & \text{if } z \geq -\frac{a}{b} \end{cases} \tag{11}$$

where:

$$z = \frac{x-\mu}{\sigma} \tag{12}$$

$$a = 4\lambda\frac{\nu-2}{\nu-1} \tag{13}$$

$$b = \sqrt{1+3\lambda^2-a^2} \tag{14}$$

$$c = \frac{\Gamma(\frac{\nu+1}{2})}{\sqrt{\pi(\nu-2)}\Gamma(\frac{\nu}{2})} \tag{15}$$

The function `fit_skewed_t()` implements maximum likelihood estimation to find the optimal parameters:

$$\hat{\theta}_r =_{\theta_r} - \sum_{t:regime_t=r} \log f(r_t;\theta_r) \tag{16}$$

This was one of the trickiest parts of the implementation. The likelihood function can be numerically unstable, especially when estimating the degrees of freedom parameter ν. To address this, I added regularization terms and implemented multiple optimization strategies:

$$\mathcal{L}_{reg}(\theta) = -\sum_t \log f(r_t;\theta) + \lambda_\nu(\nu-8)^2 + \lambda_\lambda|\lambda|^4 + \lambda_\sigma \mathbb{I}_{\sigma<0.2}\frac{0.2}{\sigma} \tag{17}$$

where $\mathbb{I}$ is an indicator function. The regularization discourages extreme parameter values: very low degrees of freedom, high skewness, or near-zero scale parameters.

The optimization uses a multi-start approach with different initial values and methods (Nelder-Mead, Powell, L-BFGS-B, and SLSQP). I track the best solution across all attempts and fall back to robust estimators if optimization fails.

I'm particularly proud of the safeguards I built in for numerical stability - working with the skewed t-distribution taught me a lot about robust numerical optimization. For example, adding noise to prevent exact zeros in discrete chains and scaling parameters before optimization improved convergence significantly.

3 State Classification

3.1 Adaptive Threshold Calculation

With the regime-specific distributions fitted, I derive state classification thresholds using quantile functions. For each regime r, I compute percentile-based thresholds:

$$Q_r^{big} = F^{-1}(p_r^{big}; \hat{\theta}_r) \tag{18}$$

$$Q_r^{small} = F^{-1}(p_r^{small}; \hat{\theta}_r) \tag{19}$$

where F^{-1} is the inverse CDF of the skewed t-distribution.

A key innovation in my approach is the asymmetric treatment of upward and downward moves. Based on the leverage effect documented by [4], negative returns tend to have larger impacts on volatility than positive returns of the same magnitude. I implement this by using different percentiles for up and down moves:

$$p_{crisis}^{big} = 0.87, \qquad p_{crisis}^{small} = 0.70 \tag{20}$$

$$p_{high}^{big} = 0.85, \qquad p_{high}^{small} = 0.65 \tag{21}$$

$$p_{normal}^{big} = 0.80, \qquad p_{normal}^{small} = 0.60 \tag{22}$$

$$p_{low}^{big} = 0.75, \qquad p_{low}^{small} = 0.55 \tag{23}$$

Notice how the percentiles vary by regime - I use higher percentiles during crisis periods to account for the increased uncertainty.

Computing the inverse CDF analytically is challenging for the skewed t-distribution, so I use numerical techniques:

I also apply dynamic bounds based on volatility regime to ensure the thresholds remain within reasonable ranges:

Algorithm 2 Finding Percentiles for Thresholds

```
1: function FindPercentile($target\_prob, \mu, \sigma, \nu, \lambda$)
2:     Define $F(x) = \int_{-\infty}^{x} f(t; \mu, \sigma, \nu, \lambda)dt - target\_prob$
3:     // Use Brent's method to find root of $F(x)$
4:     $result \leftarrow$ BrentQ($F$, $\mu - 5\sigma$, $\mu + 5\sigma$, tol=$10^{-4}$)
5:     return $result$
6: end function
```

$$\tau_r^{big} = \text{clip}(|Q_r^{big}|, \sigma_r \cdot min\_factor_r, \sigma_r \cdot max\_factor_r) \tag{24}$$

$$\tau_r^{small} = \text{clip}(|Q_r^{small}|, \sigma_r \cdot min\_factor_r, \sigma_r \cdot max\_factor_r) \tag{25}$$

where σ_r is the annualized volatility in regime r.

Finally, I apply asymmetric scaling to create separate thresholds for up and down moves:

$$\tau_r^{big\_up} = \tau_r^{big} \cdot (1.0 + up\_factor_r) \tag{26}$$

$$\tau_r^{small\_up} = \tau_r^{small} \cdot (1.0 + up\_factor_r) \tag{27}$$

$$\tau_r^{big\_down} = \tau_r^{big} \cdot (1.0 - down\_factor_r) \tag{28}$$

$$\tau_r^{small\_down} = \tau_r^{small} \cdot (1.0 - down\_factor_r) \tag{29}$$

The factors vary by regime, with larger asymmetry during crisis and high volatility periods.

3.2 Volatility Adjustment

The thresholds are further refined based on recent market dynamics through the function `calculate_adaptive_thresholds()`. This applies adjustments for:

1. Volatility clustering - periods of high volatility tend to cluster together 2. Leverage effect - negative returns lead to higher volatility than positive returns 3. Trend persistence - strong trends tend to continue

I calculate several adjustment factors:

$$\text{vol\_ratio} = \frac{\sigma_{short}}{\sigma_{long}} \tag{30}$$

$$\text{sign\_effect} = \frac{\sum_{i=1}^{10} \text{sign}(r_{t-i}) \cdot |r_{t-i}|^{1.5}}{\sum_{i=1}^{10} |r_{t-i}|^{0.5}} \tag{31}$$

$$\text{up\_trend} = \frac{\sum_{i=1}^{10} \max(r_{t-i}, 0)}{\sum_{i=1}^{10} |r_{t-i}| + \epsilon} \tag{32}$$

$$\text{down\_trend} = \frac{\sum_{i=1}^{10} \min(r_{t-i}, 0)}{\sum_{i=1}^{10} |r_{t-i}| + \epsilon} \tag{33}$$

I then combine these into a composite adjustment factor:

$$A_t = f_{trend} \cdot f_{vol\_cluster} \cdot f_{leverage} \tag{34}$$

where:

$$f_{trend} = \begin{cases} (1.0 + \text{clip}(up\_trend \cdot 0.25, 0, 0.12) \cdot vol\_ratio) \cdot regime\_mult, & \text{if } \bar{r}_{recent} > 0 \\ (1.0 + \text{clip}(|down\_trend| \cdot 0.4, 0, 0.20) \cdot vol\_ratio) \cdot regime\_mult, & \text{if } \bar{r}_{recent} \leq 0 \end{cases} \tag{35}$$

$$f_{vol\_cluster} = vol\_ratio^{0.45} \tag{36}$$

$$f_{leverage} = 1.0 + \text{clip}(weighted\_neg\_ret/\sigma_{base} \cdot 0.25, 0, 0.15) \tag{37}$$

The final adjustment factor is smoothed and clipped to prevent extreme values:

$$A_t^{final} = \text{clip}(EMA(A_t, span = 5), 0.65, 1.45) \tag{38}$$

This gives us time-varying thresholds for each state:

$$\tau_t^{big\_up} = \tau_{r_t}^{big\_up} \cdot \sigma_t \cdot A_t^{final} \tag{39}$$

$$\tau_t^{small\_up} = \tau_{r_t}^{small\_up} \cdot \sigma_t \cdot A_t^{final} \tag{40}$$

$$\tau_t^{big\_down} = \tau_{r_t}^{big\_down} \cdot \sigma_t \cdot A_t^{final} \tag{41}$$

$$\tau_t^{small\_down} = \tau_{r_t}^{small\_down} \cdot \sigma_t \cdot A_t^{final} \tag{42}$$

These dynamically adjusted thresholds allow the model to adapt to changing market conditions.

4 MCMC Framework

4.1 State Transition Modeling

For the Bayesian prediction framework, I first construct statistical models for each market state. The `fit_state_models()` function creates these models by analyzing historical data:

$$\text{states} = \{big\_up, small\_up, flat, small\_down, big\_down\} \tag{43}$$

For each state $s \in$ states, I calculate:

1. Mean vector of features (returns, volume change, volatility) 2. Covariance matrix of features 3. Degrees of freedom for a multivariate t-distribution 4. Historical frequency 5. Transition probabilities to other states 6. Regime-conditional probabilities

The transition probabilities are calculated with Laplace smoothing to avoid zero probabilities:

$$P(S_{t+1} = s_j | S_t = s_i) = \frac{N_{ij} + \alpha}{\sum_k (N_{ik} + \alpha)} \tag{44}$$

where N_{ij} is the count of transitions from state i to state j, and α is a smoothing parameter.

I also calculate regime-conditional probabilities:

$$P(S_t = s | R_t = r) = \frac{N_{s,r} + \alpha}{N_r + \alpha \cdot |\text{states}|} \tag{45}$$

where $N_{s,r}$ is the count of state s occurring in regime r, and N_r is the total count for regime r.

4.2 Likelihood Function

The likelihood function `log_likelihood()` evaluates how consistent a proposed state is with recent market conditions. This is one of the most important components of the model, as it encodes our financial domain knowledge and dictates the sampling behavior.

For a state s, the log-likelihood is defined as:

$$\log P(\mathcal{D}_t | S_{t+1} = s) = \log P(s|R_t) + \sum_{i=1}^{N-1} \log P(s|S_{t+1-i}) + f_{vol} + f_{mom} + f_{vol\_trend} \tag{46}$$

Let me break down each component:

1. Regime probability:

$$\log P(s|R_t) = \log(\max(P_r(s), 10^{-10})) \tag{47}$$

This captures how likely state s is in the current regime.

2. State persistence:

$$\sum_{i=1}^{N-1} \log P(s|S_{t+1-i}) \tag{48}$$

This uses transition probabilities to account for the influence of recent states.

3. Volatility consistency:

$$f_{vol} = \begin{cases} \log(1.2), & \text{if } s \text{ aligns with regime } R_t \\ \log(0.8), & \text{otherwise} \end{cases} \tag{49}$$

Certain states are more likely in certain volatility regimes.

4. Momentum effect:

$$f_{mom} = \begin{cases} \log(\min(1.5, 1.0 + |m_t| \cdot 10)), & \text{if } s \text{ aligns with momentum sign} \\ \log(\max(0.5, 1.0 - |m_t| \cdot 8)), & \text{otherwise} \end{cases} \tag{50}$$

where m_t is the recent return momentum. This captures the tendency for movements to persist.

5. Volume trend:

$$f_{vol\_trend} = \begin{cases} \log(1.15), & \text{if volume increasing and } s \in \{big\_up, big\_down\} \\ \log(0.9), & \text{if volume increasing and } s = flat \\ \log(1.1), & \text{if volume decreasing and } s = flat \\ \log(0.95), & \text{if volume decreasing and } s \in \{big\_up, big\_down\} \end{cases} \tag{51}$$

Higher volume often precedes bigger moves.

6. Rarity bonus:

$$f_{rarity} = 0.2 \cdot \log(0.3 + 0.7 \cdot (1 - f_s)) \tag{52}$$

where f_s is the historical frequency of state s. This slightly boosts the likelihood of rare states to ensure they're not completely ignored.

I chose the specific multipliers and formulas based on financial insights and experimentation. For example, the momentum effect is stronger for larger moves, consistent with findings in the momentum literature [5]. Similarly, the volume-state relationship follows from the observation that higher trading volumes often accompany significant price movements [6].

4.3 Proposal Function

The proposal function `proposal()` generates candidate states for the MCMC algorithm. I opted for a mixture proposal:

$$q(s'|s) = p_{global} \cdot q_{global}(s'|s) + (1 - p_{global}) \cdot q_{local}(s'|s) \tag{53}$$

Global moves propose any state except the current one with uniform probability:

$$q_{global}(s'|s) = \begin{cases} \frac{1}{|\text{states}|-1}, & \text{if } s' \neq s \\ 0, & \text{if } s' = s \end{cases} \tag{54}$$

Local moves use transition probabilities with adjustments:

$$q_{local}(s'|s) = \begin{cases} \frac{\pi_{ss'} \cdot \beta}{\sum_j \pi_{sj} \cdot \beta_j}, & \text{if } s' \neq s \\ \frac{\pi_{ss} \cdot \beta_s \cdot \gamma}{\sum_j \pi_{sj} \cdot \beta_j}, & \text{if } s' = s \end{cases} \tag{55}$$

where:

$$\gamma = \begin{cases} 0.5, & \text{if chain is stagnant} \\ 1.0, & \text{otherwise} \end{cases} \tag{56}$$

The global proportion p_{global} varies with temperature:

$$p_{global} = \min(0.8, p_{global}^{base} \cdot T^{0.5}) \tag{57}$$

This helps higher-temperature chains explore more broadly.

To ensure the Metropolis-Hastings algorithm remains valid, I calculate the proposal ratio:

$$r = \frac{q(s|s')}{q(s'|s)} \tag{58}$$

For global moves, this ratio is 1 (symmetric proposal). For local moves:

$$r = \frac{\pi_{s's} \cdot \beta_s \cdot \gamma_s}{\pi_{ss'} \cdot \beta_{s'} \cdot \gamma_{s'}} \tag{59}$$

where $\gamma_s, \gamma_{s'}$ account for any stagnation adjustments.

I experimented with various proposal schemes and found that this mixture approach gave the best balance between exploration (finding all possible modes) and exploitation (refining the probability estimates around each mode).

5 Replica Exchange MCMC Implementation

5.1 Core Algorithm

The replica exchange MCMC (also known as parallel tempering) is implemented in `replica_exchange_mcmc()`. This advanced MCMC variant runs multiple chains at different "temperatures" and occasionally swaps states between chains to improve mixing.

The temperature ladder I use is:

$$T = \{1.0, 5.0, 25.0\} \tag{60}$$

Higher temperatures flatten the posterior distribution, making it easier to traverse barriers between modes. At temperature T, the acceptance probability becomes:

$$\alpha(s \to s') = \min\left(1, \exp\left(\frac{\log P(\mathcal{D}_t|s') - \log P(\mathcal{D}_t|s)}{T} + \log r\right)\right) \tag{61}$$

where r is the proposal ratio. The algorithm alternates between:

1. Standard Metropolis-Hastings updates for each chain 2. Temperature swap attempts between adjacent chains

Temperature swaps occur with probability:

$$\alpha_{swap}(i,j) = \min\left(1, \exp\left(\left(\frac{1}{T_i} - \frac{1}{T_j}\right) \cdot (\log P(\mathcal{D}_t|s_j) - \log P(\mathcal{D}_t|s_i))\right)\right) \tag{62}$$

I chose replica exchange after initial experiments with standard MCMC showed poor mixing - the chain would often get stuck in a single state, especially in periods of high market uncertainty. The multiple temperatures approach dramatically improved convergence. That was however with an older state classification method with high flat state bias causing poor mixing. The current implementation does not require replica exchange, but I chose to leave it in partially to show off, and partially to not let hours of work go to waste.

My implementation balances between chain allocation across temperatures:

$$\text{chains\_per\_temp} = \{4, 4, 4\} \tag{63}$$

This gives 4 chains at each temperature for a total of 12 parallel chains. Only samples from the base temperature ($T = 1$) are used for the final prediction.

5.2 Convergence Diagnostics

I implement several diagnostics to assess MCMC convergence:

1. Gelman-Rubin statistic ($\hat{R}$):

$$\hat{R} = \sqrt{\frac{\hat{V}}{W}} \tag{64}$$

where $\hat{V} = \frac{n-1}{n}W + \frac{1}{n}B$, W is the within-chain variance, and B is the between-chain variance.

2. Effective sample size:

$$n_{eff} = \frac{mn}{1 + 2\sum_{k=1}^{\infty} \rho_k} \tag{65}$$

where m is the number of chains, n is the length of each chain, and ρ_k is the autocorrelation at lag k.

3. Adaptive burn-in detection using multiple window sizes and tests: - Mean stationarity - Variance stationarity - Distribution stationarity (Kolmogorov-Smirnov)

These diagnostics help determine when to stop sampling and how many initial samples to discard as burn-in.

5.3 Prediction Generation

The final prediction is generated by counting state frequencies in the post-burn-in samples:

$$P(S_{t+1} = s) = \frac{\text{Count}(s)}{\text{Total samples}} \tag{66}$$

I also calculate a prediction certainty metric using Jensen-Shannon divergence from a uniform distribution:

$$m_{dist} = \{s : (P(s) + \frac{1}{|\text{states}|})/2\} \tag{67}$$

$$JS_{div} = 0.5 \cdot KL(P||m_{dist}) + 0.5 \cdot KL(U||m_{dist}) \tag{68}$$

$$\text{certainty} = \frac{JS_{div}}{\log(|\text{states}|)} \tag{69}$$

where KL is the Kullback-Leibler divergence and U is the uniform distribution.

Finally, I extract the primary and secondary predictions:

$$s_{primary} =_s P(s) \tag{70}$$

$$s_{secondary} =_{s \neq s_{primary}} P(s) \tag{71}$$

This approach gives both a point prediction and a measure of confidence, which is crucial for decision-making in uncertain environments like financial markets.

6 Numerical Stability Considerations

One of the most challenging aspects of implementing this model was ensuring numerical stability. Financial time series can exhibit extreme behaviors that push statistical methods to their limits, and several components of my code required special attention to prevent numerical issues.

6.1 Skewed t-Distribution Fitting Safeguards

The skewed t-distribution fitting was particularly prone to numerical instability. The parameter estimation involves maximizing a likelihood function that can easily produce overflow, underflow, or invalid values. I implemented several safeguards:

$$\text{log-likelihood}(\theta) = \sum_i \log f(x_i; \theta) \tag{72}$$

When directly evaluating the PDF $f(x;\theta)$, I encountered issues with extreme parameter values producing invalid results. My solution was a robust objective function with built-in constraints:

$$\text{robust\_objective}(\theta, \text{data}) = \begin{cases} 10^6, & \text{if } \sigma \leq 0.1 \text{ or } \nu \leq 3.0 \text{ or } \nu > 30 \text{ or } |\lambda| \geq 0.85 \\ -\sum_i \log f(x_i;\theta) + \text{penalties}(\theta), & \text{otherwise} \end{cases} \tag{73}$$

The penalty terms discourage extreme parameter values:

$$\text{penalties}(\theta) = 0.05 \cdot \left(\frac{\nu - 8}{5}\right)^2 + 0.2 \cdot \left(\frac{|\lambda|}{0.7}\right)^4 + 0.8 \cdot \mathbb{I}_{\sigma<0.2} \cdot \left(\frac{0.2}{\sigma}\right)^2 \tag{74}$$

These penalties serve as a form of regularization, pushing the optimization toward stable parameter regions based on financial domain knowledge. For instance, very low degrees of freedom (ν) create extremely heavy tails that are rarely justified by the data.

Additionally, I implemented a data scaling approach that improved optimization stability:

$$\text{data\_median} = \text{median}(\text{data}) \tag{75}$$

$$\text{data\_scale} = \max\left(\frac{q_{75} - q_{25}}{1.35}, 10^{-4}\right) \tag{76}$$

$$\text{scaled\_data} = \frac{\text{data} - \text{data\_median}}{\text{data\_scale}} \tag{77}$$

Optimizing with scaled data and then transforming parameters back to the original scale made a huge difference in convergence reliability.

As a last resort, I implemented a fallback estimator that uses robust statistics when optimization fails:

$$\hat{\mu} = \text{mean}(\text{data}) \tag{78}$$

$$\hat{\sigma} = \max(\text{std}(\text{data}), 10^{-5}) \tag{79}$$

$$\hat{\nu} = \min\left(\max\left(3.0, \frac{6}{\max(0.1, K)} + 4\right), 20.0\right) \tag{80}$$

$$\hat{\lambda} = \text{clip}\left(\frac{S}{3}, -0.7, 0.7\right) \tag{81}$$

where K is the excess kurtosis and S is the skewness of the data.

6.2 Integration and Root-Finding Numerical Techniques

When calculating regime-specific thresholds using quantile functions, I needed to find where the CDF equals a target probability. The CDF for the skewed t-distribution requires numerical integration:

$$F(x;\mu,\sigma,\nu,\lambda) = \int_{-\infty}^{x} f(t;\mu,\sigma,\nu,\lambda)dt \tag{82}$$

I implemented a careful approach that combines multiple numerical techniques:

Algorithm 3 Stable Quantile Calculation

```
function QuantileEquation($x, target\_prob, \mu, \sigma, \nu, \lambda$)
    // Integrate PDF up to x using adaptive quadrature
    $result \leftarrow$ Integrate.quad($f$, $-\infty$, $x$, limit=100)
    return $result - target\_prob$
end function
function FindPercentile($target\_prob, \mu, \sigma, \nu, \lambda$)
    // Use Brent's method for root finding
    $result \leftarrow$ optimize.brentq(QuantileEquation, $\mu - 5\sigma$, $\mu + 5\sigma$, xtol=$10^{-4}$)
    return $result$
    // Fallback using Monte Carlo sampling
    $samples \leftarrow$ RandomT($\nu$, size=100000)
    Transform $samples$ according to skew t-distribution
    return Percentile($samples$, $target\_prob \cdot 100$)
end function
```

The fallback Monte Carlo approach was crucial for cases where the numerical integration failed. By generating a large number of random samples from the fitted distribution, I could directly estimate quantiles even for skewed distributions with complex shapes.

6.3 MCMC Numerical Safeguards

In the MCMC implementation, I addressed several numerical challenges:

1. **Log-space calculations**: All probability calculations are performed in log space to prevent underflow:

$$\log P(\mathcal{D}_t|S_{t+1} = s) = \log P(s|R_t) + \sum_{i=1}^{N-1} \log P(s|S_{t+1-i}) + \dots \tag{83}$$

$$\log \alpha = \frac{\log P(\mathcal{D}_t|s') - \log P(\mathcal{D}_t|s)}{T} + \log r \tag{84}$$

2. **Small constant addition**: To avoid log(0) errors, I add small constants to probabilities:

$$\log(\max(P, \epsilon)), \quad \text{where } \epsilon = 10^{-10} \tag{85}$$

3. **Discrete state handling**: When calculating the Gelman-Rubin statistic for discrete states, I add tiny random noise to break exact ties:

$$\text{noisy\_chain} = \text{chain} + \text{Normal}(0, 10^{-6}) \tag{86}$$

This prevents issues with zero variance in the calculation.

4. **Effective sample size calculation**: I implemented a robust approach that handles high autocorrelation:

$$\tau_{estimate} = \max\left(1.0, 1 + 2\sum_{k=1}^{cutoff}\left(1 - \frac{k}{cutoff}\right) \cdot \rho_k\right) \tag{87}$$

where ρ_k is the autocorrelation at lag k, and $cutoff$ is determined adaptively where autocorrelation becomes insignificant.

These numerical safeguards were essential for producing reliable predictions, especially during periods of market stress where extreme values become more common.

7 Advanced MCMC Diagnostics

7.1 Detailed Gelman-Rubin Implementation

The Gelman-Rubin statistic ($\hat{R}$) is essential for monitoring MCMC convergence. While the basic idea is straightforward (compare within-chain and between-chain variance), implementing it robustly for discrete financial states required several enhancements.

My implementation in `calculate_gelman_rubin()` includes:

1. Mapping categorical states to numerical values:

$$\text{state\_map} = \{big\_up : 2, small\_up : 1, flat : 0, small\_down : -1, big\_down : -2\} \tag{88}$$

2. Calculating both standard and split-chain $\hat{R}$:

$$\hat{R}_{standard} = \sqrt{\frac{\left(\frac{n-1}{n}\right)W + \frac{1}{n}B}{W}} \tag{89}$$

$$\hat{R}_{split} = \sqrt{\frac{\left(\frac{n_{split}-1}{n_{split}}\right)W_{split} + \frac{1}{n_{split}}B_{split}}{W_{split}}} \tag{90}$$

where B is the between-chain variance:

$$B = \frac{n}{m-1}\sum_{j=1}^{m}(\bar{\theta}_j - \bar{\theta})^2 \tag{91}$$

and W is the within-chain variance:

$$W = \frac{1}{m}\sum_{j=1}^{m} s_j^2 \tag{92}$$

with s_j^2 being the variance of chain j.

3. Quantile-based $\hat{R}$ for more robust assessment:

$$\text{quantiles} = [q_{10\%}, q_{90\%}] \tag{93}$$
$$\hat{R}_{low} = \text{calculate\_quantile\_rhat}(\text{chains}, 0.1) \tag{94}$$
$$\hat{R}_{high} = \text{calculate\_quantile\_rhat}(\text{chains}, 0.9) \tag{95}$$

This helps detect convergence issues that might be missed by looking only at means.

4. Rank-normalized $\hat{R}$ specifically designed for discrete distributions:

$$\text{ranks} = \text{rankdata}(\text{pooled\_chains}) \tag{96}$$
$$\hat{R}_{rank} = \sqrt{\frac{\left(\frac{n-1}{n}\right) W_{rank} + \frac{1}{n} B_{rank}}{W_{rank}}} \tag{97}$$

This transformation handles discrete distributions more effectively, as noted by [8].

The final $\hat{R}$ is the maximum of all these variants:

$$\hat{R}_{final} = \max(\hat{R}_{standard}, \hat{R}_{split}, \hat{R}_{low}, \hat{R}_{high}, \hat{R}_{rank}) \tag{98}$$

From experimentation, I found that $\hat{R} < 1.05$ typically indicates good convergence for this model, while values above 1.1 suggest more iterations are needed.

7.2 Adaptive Burn-in Detection

The traditional approach of discarding a fixed percentage of initial samples as "burn-in" isn't optimal for market prediction, where convergence rates can vary dramatically based on market conditions. I implemented a data-driven method in `adaptive_burn_in_detection()` that:

1. Calculates $\hat{R}$ for increasing prefixes of the chains 2. Tests for mean stationarity, variance stationarity, and distribution stationarity 3. Combines these indicators to estimate the optimal burn-in point

For stationarity testing, I implement:

$$\text{mean\_test}(s_1, s_2) = \text{t-test}(s_1, s_2) \tag{99}$$
$$\text{variance\_test}(s_1, s_2) = \text{Levene-test}(s_1, s_2) \tag{100}$$
$$\text{distribution\_test}(s_1, s_2) = \text{KS-test}(s_1, s_2) \tag{101}$$

where s_1 and s_2 are consecutive segments of the chain.
I assign weights to different test results based on window size and test type:

$$\text{weight}_{test,window} = \begin{cases} window \cdot 2.0, & \text{if test} = KS \\ window \cdot 1.5, & \text{if test} = mean \\ window \cdot 1.0, & \text{if test} = variance \end{cases} \tag{102}$$

The combined stationarity point is:

$$\text{stationarity\_point} = \frac{\sum_{test,window} \text{point}_{test,window} \cdot \text{weight}_{test,window}}{\sum_{test,window} \text{weight}_{test,window}} \tag{103}$$

Finally, I combine this with the $\hat{R}$ convergence point and apply conservative bounds:

$$\text{burn\_in} = \min(\text{r\_hat\_point}, \text{stationarity\_point}) \tag{104}$$
$$\text{burn\_in} = \max(\text{burn\_in}, \min(500, \text{min\_length}/10)) \tag{105}$$
$$\text{burn\_in} = \min(\text{burn\_in}, \text{min\_length} \cdot 2/5) \tag{106}$$

This ensures I never discard too little (missing non-stationary initial samples) or too much (wasting good samples).

7.3 Effective Sample Size Estimation

In MCMC analysis, the raw number of samples can be misleading due to autocorrelation. I implemented an enhanced effective sample size (ESS) estimator that:

1. Uses FFT for efficient autocorrelation calculation:

$$\text{fft} = \text{FFT}(series, n = 2 \cdot n_{points}) \tag{107}$$
$$\text{acf} = \text{IFFT}(fft \cdot \text{conjugate}(fft))[: lag_{max} + 1] \tag{108}$$

2. Dynamically determines the autocorrelation cutoff:

$$\text{cutoff\_idx} = \min\{i : |\rho_i| < 0.05 \text{ or } i = lag_{max}\} \tag{109}$$

3. Uses a Bartlett window to reduce variance in spectral density estimation:

$$\text{bartlett\_weights} = 1 - \frac{\text{arange}(\text{cutoff\_idx})}{\text{cutoff\_idx}} \tag{110}$$
$$\text{weighted\_acf} = 2 \cdot \sum_{i=1}^{\text{cutoff\_idx}-1} \text{bartlett\_weights}[i] \cdot \rho_i \tag{111}$$

4. Calculates τ and ESS:

$$\tau_{estimate} = \max(1.0, 1 + \text{weighted\_acf}) \tag{112}$$

$$n_{eff} = \frac{m \cdot n}{\tau_{estimate}} \tag{113}$$

This approach, inspired by [7], provides more accurate estimates of sampling efficiency, especially when chains exhibit strong autocorrelation.

8 Performance Metrics and Evaluation

8.1 State Metrics Calculation

To evaluate prediction performance, I implemented the `calculate_state_metrics()` function that computes:

1. State distribution:

$$P(state) = \frac{\text{Count}(state)}{\text{Total observations}} \tag{114}$$

2. State stability (how often states change):

$$\text{stability} = 1 - \frac{\text{transitions}}{\text{total\_steps} - 1} \tag{115}$$

3. Extreme capture ratio (accuracy during large market moves):

$$\text{extreme\_capture} = \frac{\text{correct\_signals}}{\text{total\_big\_moves}} \tag{116}$$

where a signal is "correct" if the predicted direction matches the actual direction during large market moves.

4. False signal rate:

$$\text{false\_signal\_rate} = \frac{\text{false\_signals}}{\text{total\_signals}} \tag{117}$$

where a "false signal" is a non-flat prediction where the direction was wrong.

These metrics go beyond simple accuracy, addressing the specific challenges of financial prediction where some errors are more costly than others. For example, missing a big market move (false negative) is often more damaging than predicting a move that doesn't materialize (false positive).

8.2 Confidence Calculation

The function `get_prediction_confidence()` converts raw MCMC samples into interpretable confidence metrics:

1. Raw state probabilities:

$$P(state) = \frac{\text{Count}(state)}{\text{Total samples}} \tag{118}$$

2. Simple certainty (gap between top predictions):

$$\text{simple\_certainty} = P(primary\_state) - P(secondary\_state) \tag{119}$$

3. Jensen-Shannon divergence from uniform distribution:

$$m_{dist} = \{s : (P(s) + \frac{1}{|\text{states}|})/2 \text{ for all } s\} \tag{120}$$

$$JS_{div} = 0.5 \cdot KL(P||m_{dist}) + 0.5 \cdot KL(U||m_{dist}) \tag{121}$$

4. Normalized prediction certainty:

$$\text{certainty} = \frac{JS_{div}}{\log(|\text{states}|)} \tag{122}$$

This approach gives a nuanced view of prediction confidence that isn't just based on the primary probability. A prediction with 40% probability for the primary state might not seem confident, but if all other states have 15% each, it's actually quite decisive relative to the uniform distribution.

9 Conclusion

This market prediction model represents my attempt to bridge the gap between purely statistical methods and domain-specific financial knowledge. By combining adaptive thresholds, regime detection, and Bayesian inference with MCMC, I've created a framework that provides probabilistic predictions for next-day market states.

The key contributions include:

- A dynamic threshold approach that adapts to changing market conditions
- A robust skewed t-distribution fitting procedure for fat-tailed returns
- A carefully designed likelihood function that incorporates financial domain knowledge
- A replica exchange MCMC implementation with convergence monitoring

Throughout the development process, I emphasized numerical robustness and proper uncertainty quantification over point-forecast accuracy. This approach better aligns with the inherent unpredictability of financial markets while still providing actionable insights.

The project has been an incredible learning journey, combining statistical theory, computational methods, and financial domain knowledge.

If I ever return to this project I will definitely switch to a Hamiltonian Markov Chain Monte Carlo method in order to make predictions in the continuous space.

References

[1] Ang, A., & Timmermann, A. (2012). Regime changes and financial markets. *Annual Review of Financial Economics.*

[2] Hamilton, J. D. (2010). Regime switching models. In *Macroeconometrics and Time Series Analysis.* Palgrave Macmillan.

[3] Azzalini, A., & Capitanio, A. (2003). Distributions generated by perturbation of symmetry with emphasis on a multivariate skew t-distribution. *Journal of the Royal Statistical Society: Series B.*

[4] Black, F. (1976). Studies of stock price volatility changes. In *Proceedings of the 1976 Meeting of the Business and Economic Statistics Section*, American Statistical Association.

[5] Jegadeesh, N., & Titman, S. (1993). Returns to buying winners and selling losers: Implications for stock market efficiency. *The Journal of Finance.*

[6] Karpoff, J. M. (1987). The relation between price changes and trading volume: A survey. *Journal of Financial and Quantitative Analysis.*

[7] Geyer, C. J. (1991). Markov chain Monte Carlo maximum likelihood. In *Computing Science and Statistics: Proceedings of the 23rd Symposium on the Interface.*

[8] Gelman, A., & Rubin, D. B. (1992). Inference from iterative simulation using multiple sequences. *Statistical Science.*