G2++ Interest Rate Model

Methodology and Implementation Documentation

IR-models Repository

December 12, 2025

Abstract

This document provides a complete mathematical and computational description of the G2++ (two-factor Gaussian) interest rate model as implemented in the IR-models repository. We detail the stochastic dynamics, zero-coupon bond pricing formulas, EURIBOR rate calculations, parameter estimation via Extended Kalman Filter, and Monte Carlo simulation methodology. All formulas and algorithms correspond exactly to the code implementation.

1 Introduction

The G2++ model is a two-factor Gaussian short rate model introduced by Brigo and Mercurio [1]. It extends the Hull-White model by incorporating two correlated mean-reverting factors, providing greater flexibility in fitting the term structure of interest rates and capturing complex volatility patterns.

1.1 Model Overview

The short rate $r(t)$ is decomposed into three components:

$$r(t) = x_1(t) + x_2(t) + \phi(t) \tag{1}$$

where:

- $x_1(t), x_2(t)$ are stochastic factors following Ornstein-Uhlenbeck processes
- $\phi(t)$ is a deterministic shift function used to fit the initial term structure

2 Model Dynamics

2.1 Factor Dynamics

The two factors evolve according to correlated Ornstein-Uhlenbeck (mean-reverting) processes:

$$dx_1(t) = -a\, x_1(t)\, dt + \sigma\, dW_1(t) \tag{2}$$
$$dx_2(t) = -b\, x_2(t)\, dt + \eta\, dW_2(t) \tag{3}$$

where:

- $a, b > 0$ are mean-reversion speeds
- $\sigma, \eta > 0$ are volatility parameters
- $W_1(t), W_2(t)$ are standard Brownian motions with correlation:

$$d\langle W_1, W_2 \rangle_t = \rho\, dt, \quad \rho \in (-1, 1) \tag{4}$$

2.2 Parameter Interpretation

Parameter	Range	Interpretation
a	$(0, \infty)$	Mean-reversion speed of first factor. Higher values indicate faster return to zero.
b	$(0, \infty)$	Mean-reversion speed of second factor.
σ	$(0, \infty)$	Volatility of first factor.
η	$(0, \infty)$	Volatility of second factor.
ρ	$(-1, 1)$	Correlation between the two Brownian drivers.
$\phi(t)$	$\mathbb{R}$	Deterministic shift ensuring fit to initial yield curve.

Table 1: G2++ model parameters

2.3 Stationary Distribution

Each factor has a stationary (long-run) Gaussian distribution:

$$x_i(\infty) \sim \mathcal{N}\left(0, \frac{\sigma_i^2}{2a_i}\right) \tag{5}$$

where $(a_1, \sigma_1) = (a, \sigma)$ and $(a_2, \sigma_2) = (b, \eta)$.

Implementation: See function `_stationary_variance()` in `ir_models/estimation/g2pp.py`:

```
def _stationary_variance(kappa: float, vol: float) -> float:
    return vol**2 / (2.0 * kappa)
```

3 Zero-Coupon Bond Pricing

3.1 Analytical Formula

The price at time t of a zero-coupon bond maturing at time T is given by:

$$P(t,T) = A(t,T) \cdot \exp\left(-B_1(t,T)\, x_1(t) - B_2(t,T)\, x_2(t)\right) \tag{6}$$

3.2 Factor Loading Functions

The factor loading functions are:

$$B_i(t,T) = \frac{1 - e^{-a_i(T-t)}}{a_i}, \quad i = 1, 2 \tag{7}$$

where $(a_1, a_2) = (a, b)$.

Limiting case: As $a_i \to 0$, we have $B_i(t,T) \to T - t$ by L'Hôpital's rule.

Implementation: See methods `B1()` and `B2()` in class `G2ppZeroCouponPricing`:

```
def B1(self, t: float, T: float) -> float:
    tau = T - t
    if tau <= 0:
        return 0.0
    if self.a < 1e-8:
        return tau
    return (1.0 - np.exp(-self.a * tau)) / self.a
```

3.3 Deterministic Component

The function $A(t,T)$ incorporates the deterministic shift and variance corrections:

$$A(t,T) = \exp\left(\int_t^T \phi(s)\, ds - V(t,T)\right) \tag{8}$$

where $V(t,T)$ is the variance contribution from the stochastic factors.

3.4 Variance Function

The variance function $V(t,T)$ is given by:

$$V(t,T) = V_1(t,T) + V_2(t,T) + V_{12}(t,T) \tag{9}$$

where:

$$V_1(t,T) = \frac{\sigma^2}{2a^2}\left(T - t + \frac{2}{a}e^{-a(T-t)} - \frac{1}{2a}e^{-2a(T-t)} - \frac{3}{2a}\right) \tag{10}$$

$$V_2(t,T) = \frac{\eta^2}{2b^2}\left(T - t + \frac{2}{b}e^{-b(T-t)} - \frac{1}{2b}e^{-2b(T-t)} - \frac{3}{2b}\right) \tag{11}$$

$$V_{12}(t,T) = \frac{\rho\sigma\eta}{ab}\left(T - t + \frac{e^{-a(T-t)} - 1}{a} + \frac{e^{-b(T-t)} - 1}{b} - \frac{e^{-(a+b)(T-t)} - 1}{a+b}\right) \tag{12}$$

Implementation: See method `V()` in class `G2ppZeroCouponPricing`.

3.5 Properties

Theorem 1 (Bond Price Properties)**.** *The zero-coupon bond price satisfies:*

1. $P(t,t) = 1$ *for all* t *(normalization)*
2. $P(t,T) \in (0,1)$ *for* $T > t$ *(assuming positive rates)*
3. $P(t,T)$ *is decreasing in* T *for fixed* t *and positive short rate*
4. $\lim_{T\to t} P(t,T) = 1$

4 EURIBOR Rate Calculation

4.1 Definition

EURIBOR (Euro Interbank Offered Rate) is a **forward rate**, not a short rate. It represents the rate at which banks lend to each other for a fixed tenor.

Definition 1 (EURIBOR Rate)**.** *The EURIBOR rate with tenor* δ *starting at time* T *and observed at time* $t \le T$ *is:*

$$L(t;T,T+\delta) = \frac{1}{\delta}\left(\frac{P(t,T)}{P(t,T+\delta)} - 1\right) \tag{13}$$

For **spot-starting** EURIBOR (most common), we have $T = t$:

$$L(t;t,t+\delta) = \frac{1}{\delta}\left(\frac{P(t,t)}{P(t,t+\delta)} - 1\right) = \frac{1}{\delta}\left(\frac{1}{P(t,t+\delta)} - 1\right) \tag{14}$$

4.2 Standard Tenors

Implementation: See method `euribor_rate()` in class `G2ppZeroCouponPricing`:

```
def euribor_rate(self, t: float, T: float, delta: float,
                 x1: float, x2: float) -> float:
    P_T = self.zero_coupon_price(t, T, x1, x2)
    P_T_delta = self.zero_coupon_price(t, T + delta, x1, x2)
    if P_T_delta <= 0:
        return 0.0
    return (1.0 / delta) * (P_T / P_T_delta - 1.0)
```

Name	Tenor	δ (years)
EURIBOR 1M	1 month	$\approx 1/12$
EURIBOR 3M	3 months	0.25
EURIBOR 6M	6 months	0.5
EURIBOR 12M	12 months	1.0

Table 2: Common EURIBOR tenors

4.3 Importance for Calibration

Remark 1. *When calibrating the G2++ model to market EURIBOR data, it is* ***critical*** *to use equation* (13) *rather than treating EURIBOR as the short rate $r(t)$. The former is mathematically correct and accounts for the forward-looking nature of EURIBOR.*

This distinction necessitates the use of an Extended Kalman Filter (see Section 6.3) since the observation equation becomes nonlinear.

5 Discrete-Time Simulation

5.1 Euler-Maruyama Scheme

The continuous-time SDEs (2)-(3) are discretized using the Euler-Maruyama method with time step Δt:

$$x_1(t+\Delta t) = x_1(t) - a\,x_1(t)\,\Delta t + \sigma\sqrt{\Delta t}\,Z_1 \tag{15}$$

$$x_2(t+\Delta t) = x_2(t) - b\,x_2(t)\,\Delta t + \eta\sqrt{\Delta t}\,Z_2 \tag{16}$$

where $(Z_1, Z_2) \sim \mathcal{N}(0, \Sigma)$ with correlation matrix:

$$\Sigma = \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix} \tag{17}$$

5.2 Correlation Generation

Correlated random variables are generated via Cholesky decomposition:

$$\begin{pmatrix} Z_1 \\ Z_2 \end{pmatrix} = L \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \end{pmatrix} \tag{18}$$

where $\varepsilon_1, \varepsilon_2 \overset{iid}{\sim} \mathcal{N}(0,1)$ and L is the Cholesky factor:

$$L = \begin{pmatrix} 1 & 0 \\ \rho & \sqrt{1-\rho^2} \end{pmatrix} \tag{19}$$

Implementation: See function `simulate_paths()` in class `G2ppModel`:

```
corr_matrix = np.array([[1.0, self.rho], [self.rho, 1.0]])
chol = np.linalg.cholesky(corr_matrix)
z = np.random.normal(size=(n_simulations, n_steps, 2))
dW = np.einsum("ijk,kl->ijl", z, chol.T)
```

5.3 Short Rate Reconstruction

At each time step, the short rate is:

$$r(t_i) = x_1(t_i) + x_2(t_i) + \phi(t_i) \tag{20}$$

5.4 Algorithm Summary

Algorithm 1 Monte Carlo Simulation of G2++ Short Rate Paths

Require: Parameters $(a, b, \sigma, \eta, \rho)$, initial values (x_1^0, x_2^0), function $\phi(t)$
Require: Time horizon T, number of steps N, number of paths M
Set $\Delta t = T/N$
Initialize arrays $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^{M\times(N+1)}$
Set $\mathbf{x}_1[:,0] = x_1^0$ and $\mathbf{x}_2[:,0] = x_2^0$
Compute Cholesky factor L of correlation matrix
for $i = 0$ to $N-1$ **do**
  Generate $\boldsymbol{\varepsilon}^{(i)} \sim \mathcal{N}(0, I_2)$ for all M paths
  Compute $\mathbf{Z}^{(i)} = \boldsymbol{\varepsilon}^{(i)} L^T$
  for each path $m = 1$ to M **do**
    $x_1^m(t_{i+1}) = x_1^m(t_i) - a\, x_1^m(t_i)\, \Delta t + \sigma\sqrt{\Delta t}\, Z_1^{m,(i)}$
    $x_2^m(t_{i+1}) = x_2^m(t_i) - b\, x_2^m(t_i)\, \Delta t + \eta\sqrt{\Delta t}\, Z_2^{m,(i)}$
  end for
end for
for $i = 0$ to N **do**
  $r^m(t_i) = x_1^m(t_i) + x_2^m(t_i) + \phi(t_i)$ for all paths m
end for
return Short rate paths $\{r^m(t_i)\}_{m=1,\dots,M}^{i=0,\dots,N}$

6 Parameter Estimation

6.1 State-Space Formulation

We formulate the parameter estimation problem in state-space form suitable for Kalman filtering.

6.1.1 State Equation (Transition Model)

The state vector is:

$$\mathbf{x}_t = \begin{pmatrix} x_1(t) \\ x_2(t) \end{pmatrix} \tag{21}$$

The exact discrete-time transition for OU processes is:

$$\mathbf{x}_{t+\Delta t} = F\, \mathbf{x}_t + \mathbf{u}_t \tag{22}$$

where:

$$F = \begin{pmatrix} e^{-a\Delta t} & 0 \\ 0 & e^{-b\Delta t} \end{pmatrix} \tag{23}$$

and $\mathbf{u}_t \sim \mathcal{N}(0, Q)$ with process noise covariance:

$$Q = \begin{pmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{pmatrix} \tag{24}$$

The standard deviations for the discrete increments are:

$$\sigma_i = \nu_i \sqrt{\frac{1 - e^{-2a_i \Delta t}}{2a_i}} \tag{25}$$

where $(a_1, \nu_1) = (a, \sigma)$ and $(a_2, \nu_2) = (b, \eta)$.

Implementation: See function `_transition_matrices()`:

```
def _transition_matrices(a, b, sigma, eta, rho, dt):
    f1 = math.exp(-a * dt)
    f2 = math.exp(-b * dt)
    F = np.array([[f1, 0.0], [0.0, f2]])

    std_x = _discrete_ou_coefficients(a, sigma, dt)
    std_y = _discrete_ou_coefficients(b, eta, dt)

    Q = np.array([
        [std_x**2, rho * std_x * std_y],
        [rho * std_x * std_y, std_y**2],
    ])
    return F, Q
```

6.1.2 Observation Equation

The observation equation depends on the type of data observed.

Mode 1: Short Rate Observations (Linear) If we observe the short rate directly:

$$z_t = H\,\mathbf{x}_t + \phi(t) + \varepsilon_t \tag{26}$$

where $H = \begin{pmatrix} 1 & 1 \end{pmatrix}$ and $\varepsilon_t \sim \mathcal{N}(0, R)$ with R being the measurement variance.

Mode 2: EURIBOR Observations (Nonlinear) If we observe EURIBOR rates:

$$z_t = h(\mathbf{x}_t, t) + \varepsilon_t \tag{27}$$

where:

$$h(\mathbf{x}_t, t) = L(t; T, T+\delta) = \frac{1}{\delta}\left(\frac{P(t, T, \mathbf{x}_t)}{P(t, T+\delta, \mathbf{x}_t)} - 1\right) \tag{28}$$

This observation function is **nonlinear** in the state $\mathbf{x}_t$, necessitating an Extended Kalman Filter.

6.2 Standard Kalman Filter (Short Rate Mode)

For linear observations (26), we use the standard Kalman filter.

6.2.1 Initialization

The initial state distribution uses the stationary distribution:

$$\mathbf{x}_0 \sim \mathcal{N}(\mathbf{0}, P_0) \tag{29}$$

where:

$$P_0 = \begin{pmatrix} \frac{\sigma^2}{2a} & 0 \\ 0 & \frac{\eta^2}{2b} \end{pmatrix} \tag{30}$$

6.2.2 Kalman Filter Recursion

For each observation z_t at time t:

1. **Predict:**

$$\hat{\mathbf{x}}_{t|t-1} = F\,\hat{\mathbf{x}}_{t-1|t-1} \tag{31}$$
$$P_{t|t-1} = F\,P_{t-1|t-1}\,F^T + Q \tag{32}$$

2. **Innovation:**

$$\hat{z}_t = H\,\hat{\mathbf{x}}_{t|t-1} + \phi(t) \tag{33}$$
$$\nu_t = z_t - \hat{z}_t \quad \text{(innovation)} \tag{34}$$
$$S_t = H\,P_{t|t-1}\,H^T + R \quad \text{(innovation covariance)} \tag{35}$$

3. **Update:**

$$K_t = P_{t|t-1}\,H^T\,S_t^{-1} \quad \text{(Kalman gain)} \tag{36}$$
$$\hat{\mathbf{x}}_{t|t} = \hat{\mathbf{x}}_{t|t-1} + K_t\,\nu_t \tag{37}$$
$$P_{t|t} = P_{t|t-1} - K_t\,H\,P_{t|t-1} \tag{38}$$

4. **Log-likelihood contribution:**

$$\ell_t = -\frac{1}{2}\left(\log(2\pi) + \log(S_t) + \frac{\nu_t^2}{S_t}\right) \tag{39}$$

6.3 Extended Kalman Filter (EURIBOR Mode)

For nonlinear observations (27), we use the Extended Kalman Filter (EKF).

6.3.1 Linearization

The observation function $h(\mathbf{x}_t, t)$ is linearized around the predicted state:

$$H_t = \nabla_{\mathbf{x}} h|_{\hat{\mathbf{x}}_{t|t-1}} = \left(\frac{\partial h}{\partial x_1} \quad \frac{\partial h}{\partial x_2}\right)\Big|_{\hat{\mathbf{x}}_{t|t-1}} \tag{40}$$

Numerical Jacobian: The partial derivatives are computed using finite differences:

$$\frac{\partial h}{\partial x_1} \approx \frac{h(x_1+\epsilon, x_2, t) - h(x_1, x_2, t)}{\epsilon} \tag{41}$$
$$\frac{\partial h}{\partial x_2} \approx \frac{h(x_1, x_2+\epsilon, t) - h(x_1, x_2, t)}{\epsilon} \tag{42}$$

with $\epsilon = 10^{-6}$ (default).

Implementation: See method `_observation_jacobian()`:

```
def _observation_jacobian(self, state, t, pricing,
                          epsilon=1e-6):
    x1, x2 = state
    h0 = self._observation_function(state, t, pricing)

    # Perturb x1
    state_dx1 = np.array([x1 + epsilon, x2])
    h_dx1 = self._observation_function(state_dx1, t, pricing)
```

```
    dh_dx1 = (h_dx1 - h0) / epsilon

    # Perturb x2
    state_dx2 = np.array([x1, x2 + epsilon])
    h_dx2 = self._observation_function(state_dx2, t, pricing)
    dh_dx2 = (h_dx2 - h0) / epsilon

    return np.array([[dh_dx1, dh_dx2]])
```

6.3.2 EKF Recursion

The EKF proceeds similarly to the standard Kalman filter, but with H_t computed at each time step:

1. **Predict:** Same as standard KF
2. **Compute Jacobian:** $H_t = \nabla_{\mathbf{x}} h|_{\hat{\mathbf{x}}_{t|t-1}}$
3. **Innovation:**

$$\hat{z}_t = h(\hat{\mathbf{x}}_{t|t-1}, t) \quad \text{(nonlinear)} \tag{43}$$

$$\nu_t = z_t - \hat{z}_t \tag{44}$$

$$S_t = H_t \, P_{t|t-1} \, H_t^T + R \tag{45}$$

4. **Update:** Use H_t instead of H
5. **Log-likelihood:** Same formula

6.4 Maximum Likelihood Estimation

The total log-likelihood is:

$$\mathcal{L}(\theta) = \sum_{t=1}^{N} \ell_t(\theta) \tag{46}$$

where $\theta = (a, b, \sigma, \eta, \rho, R)$ is the parameter vector.

6.4.1 Optimization

We maximize $\mathcal{L}(\theta)$ using the L-BFGS-B algorithm (limited-memory Broyden-Fletcher-Goldfarb-Shanno with box constraints).

Implementation: Uses `scipy.optimize.minimize`:

```
result = minimize(
    fun=self._negative_log_likelihood,
    x0=guess_vec,
    bounds=bound_list,
    method='L-BFGS-B'
)
```

6.4.2 Parameter Bounds

Default bounds to ensure physically meaningful parameters:

Parameter	Lower Bound	Upper Bound
a, b	10^{-4}	5.0
σ, η	10^{-4}	0.2
ρ	-0.999	0.999
R	10^{-8}	0.01

Table 3: Default parameter bounds for optimization

6.4.3 Fixed Parameters

Some parameters can be fixed during estimation to reduce the optimization dimension or incorporate prior knowledge:

```
result = estimator.fit(
    initial_guess={"sigma": 0.09, "eta": 0.09, "rho": -0.1},
    fixed_params={"a": 0.01, "b": 0.50}
)
```

6.5 Algorithm Summary

Algorithm 2 G2++ Parameter Estimation via Kalman/EKF and MLE

Require: Observed rates $\{z_1, \ldots, z_N\}$, time step Δt, mode (short_rate or euribor)
Require: Initial parameter guess θ_0, bounds $[\theta_{\min}, \theta_{\max}]$
Define negative log-likelihood function:
  $f(\theta) = -\mathcal{L}(\theta) = -\sum_{t=1}^{N} \ell_t(\theta)$
  where $\ell_t(\theta)$ is computed via Kalman filter or EKF
Run constrained optimization:
  $\hat{\theta} = \arg\min_{\theta \in [\theta_{\min}, \theta_{\max}]} f(\theta)$
  using L-BFGS-B algorithm
Run final Kalman/EKF pass with $\hat{\theta}$ to get filtered states
return Estimated parameters $\hat{\theta}$, filtered states $\{\hat{\mathbf{x}}_t\}$, log-likelihood $\mathcal{L}(\hat{\theta})$

7 Deterministic Shift Function $\phi(t)$

7.1 Purpose

The function $\phi(t)$ ensures that the model fits the initial observed term structure of interest rates. Without $\phi(t)$, the model would only capture the stochastic dynamics but not match market prices at $t = 0$.

7.2 Calibration Methods

7.2.1 Zero Shift (Default)

For simplicity or when only relative movements matter:

$$\phi(t) = 0 \quad \forall t \tag{47}$$

7.2.2 Constant Shift

Match the initial short rate:

$$\phi(t) = r_0 - x_1^0 - x_2^0 \tag{48}$$

7.2.3 Term Structure Fitting

For full calibration, $\phi(t)$ is chosen such that:

$$P^{\text{model}}(0, T; \phi) = P^{\text{market}}(0, T) \quad \forall T \tag{49}$$

This typically requires iterative or analytical inversion of the bond pricing formula.

Note: The current implementation uses $\phi(t) = 0$ by default. Full term structure fitting is marked as TODO in the code.

8 Model Validation and Diagnostics

8.1 Goodness of Fit Metrics

After calibration, assess model fit using:

1. **Mean Absolute Error (MAE):**

$$\text{MAE} = \frac{1}{N} \sum_{t=1}^{N} |z_t - \hat{z}_t| \tag{50}$$

2. **Root Mean Squared Error (RMSE):**

$$\text{RMSE} = \sqrt{\frac{1}{N} \sum_{t=1}^{N} (z_t - \hat{z}_t)^2} \tag{51}$$

3. **Correlation:**

$$\text{Corr}(z, \hat{z}) = \frac{\text{Cov}(z, \hat{z})}{\sigma_z \sigma_{\hat{z}}} \tag{52}$$

where $\hat{z}_t$ is the fitted value from the filtered states.

8.2 Residual Analysis

Examine residuals $\varepsilon_t = z_t - \hat{z}_t$ for:

- Zero mean: $\mathbb{E}[\varepsilon_t] \approx 0$
- No autocorrelation: $\text{Cov}(\varepsilon_t, \varepsilon_{t+k}) \approx 0$ for $k \neq 0$
- Normality: Q-Q plot, Shapiro-Wilk test

9 Computational Complexity

9.1 Kalman Filter

- **Per time step:** $O(n^3)$ where $n = 2$ (state dimension)
- **Total for N observations:** $O(N)$ (since n is constant)

9.2 Extended Kalman Filter

- **Additional cost:** Jacobian computation via finite differences
- **Per time step:** $O(n^2 \times \text{cost}(h))$ where h is the observation function
- For EURIBOR: h involves ZC bond pricing, which is $O(1)$ with analytical formulas

9.3 Monte Carlo Simulation

- **Time complexity:** $O(M \times N)$ where M = number of paths, N = number of time steps
- **Memory:** $O(M \times N)$ for storing all paths

10 Implementation Notes

10.1 Numerical Stability

1. **Exponential underflow:** For large $a(T-t)$ or $b(T-t)$, use approximations:
   - $e^{-x} \approx 0$ for $x > 35$
   - $1 - e^{-x} \approx 1$ for $x > 10$
2. **Near-zero mean reversion:** When $a_i < 10^{-8}$, use limiting formulas:
   - $B_i(t,T) \to T - t$
   - Simplified variance expressions
3. **Covariance matrix positivity:** Ensure Q and P remain positive definite by:
   - Enforcing $|\rho| < 0.999$ (not exactly ± 1)
   - Using Joseph form of covariance update (optional)

10.2 Vectorization

The implementation uses NumPy vectorization for efficiency:

```
# Vectorized Euler-Maruyama update
x_paths[:, i+1] = (x_paths[:, i]
                   - a * x_paths[:, i] * dt
                   + sigma * sqrt_dt * dW[:, i, 0])
```

10.3 Random Number Generation

For reproducibility:

```
np.random.seed(random_seed)
```

11 Usage Examples

11.1 Monte Carlo Simulation

```python
from ir_models.models.g2pp import G2ppModel

model = G2ppModel(
    a=0.1, b=0.2,
    sigma=0.01, eta=0.015,
    rho=-0.3,
    x0=0.0, y0=0.0,
    r0=0.02
)

result = model.simulate_paths(
    T=10.0,            # 10-year horizon
    n_steps=1000,      # 1000 time steps
    n_simulations=10000,  # 10,000 paths
    random_seed=42
)

# Access results
short_rates = result.short_rate_paths  # shape: (10000, 1001)
time_grid = result.time_grid
```

11.2 Parameter Estimation from EURIBOR Data

```python
from ir_models.estimation.g2pp import G2ppKalmanMLE

# Load EURIBOR 3M data (as decimals)
euribor_rates = ...  # shape: (N,)

estimator = G2ppKalmanMLE(
    observations=euribor_rates,
    dt=1.0/12.0,                # Monthly data
    phi=lambda t: 0.0,          # Zero shift
    observation_mode="euribor",
    euribor_tenor=0.25,         # 3 months
    euribor_start_offset=0.0    # Spot-starting
)

result = estimator.fit(
    initial_guess={
        "sigma": 0.09,
        "eta": 0.09,
        "rho": -0.1,
        "measurement_var": 1e-5
    },
    fixed_params={
        "a": 0.01,
        "b": 0.50
    }
)

print(f"Log-likelihood: {result.log_likelihood}")
print(f"Estimated parameters: {result.params}")
```

```
# Extract filtered factors
x1_filtered = result.filtered_states[:, 0]
x2_filtered = result.filtered_states[:, 1]
```

11.3 Zero-Coupon Bond Pricing

```
from ir_models.models.g2pp import G2ppZeroCouponPricing

pricing = G2ppZeroCouponPricing(
    a=0.1, b=0.2,
    sigma=0.01, eta=0.015,
    rho=-0.3,
    phi=lambda t: 0.0
)

# Price a 5-year zero-coupon bond
P = pricing.zero_coupon_price(t=0, T=5.0, x1=0.01, x2=-0.005)
print(f"ZC Bond Price: {P:.6f}")

# Compute 3M EURIBOR rate
euribor = pricing.euribor_rate(t=0, T=0, delta=0.25,
                               x1=0.01, x2=-0.005)
print(f"EURIBOR 3M: {euribor*100:.4f}%")
```

12 Limitations and Extensions

12.1 Current Limitations

1. **Single-curve framework:** The implementation assumes a single curve for both discounting and forward rate projection.
2. **Simplified $\phi(t)$:** Full term structure fitting is not yet implemented.
3. **No smile modeling:** The model produces flat volatility structures.
4. **Gaussian rates:** Can produce negative interest rates (realistic in some markets, but may be undesirable).

12.2 Possible Extensions

1. **Multi-curve framework:** Separate OIS (discount) and EURIBOR (projection) curves with basis spreads:
$$L^{\text{EURIBOR}}(t;T,T+\delta) = L^{\text{OIS}}(t;T,T+\delta) + \text{Spread}(t,T,\delta) \tag{53}$$
2. **Term structure fitting:** Implement analytical or numerical methods to calibrate $\phi(t)$ to market bond prices.
3. **Stochastic volatility:** Extend to models like G2++ with stochastic σ, η.
4. **Non-negativity constraints:** Use transformations or shifted lognormal models to ensure $r(t) \geq 0$.
5. **Parallel computation:** GPU acceleration for Monte Carlo simulations.

13 Conclusion

This document provides a complete description of the G2++ model implementation in the IR-models repository. All mathematical formulas, algorithms, and computational details correspond exactly to the code.

Key takeaways:

- The G2++ model captures interest rate dynamics via two correlated mean-reverting factors
- Zero-coupon bond prices have closed-form solutions
- EURIBOR rates are forward rates computed from ZC bond ratios
- Parameter estimation uses Kalman filtering (standard for short rates, extended for EURIBOR)
- Monte Carlo simulation uses Euler-Maruyama discretization with Cholesky-based correlation

For practical applications, the EURIBOR observation mode with Extended Kalman Filter is recommended for calibrating to market data.

References

[1] Brigo, D., & Mercurio, F. (2006). *Interest Rate Models—Theory and Practice* (2nd ed.). Springer Finance. Springer-Verlag Berlin Heidelberg.

[2] Hull, J. C. (2018). *Options, Futures, and Other Derivatives* (10th ed.). Pearson.

[3] Durbin, J., & Koopman, S. J. (2012). *Time Series Analysis by State Space Methods* (2nd ed.). Oxford University Press.

[4] Särkkä, S. (2013). *Bayesian Filtering and Smoothing*. Cambridge University Press.

A Mathematical Derivations

A.1 Stationary Variance of OU Process

For the SDE $dx = -ax\,dt + \sigma\,dW$, the variance satisfies:

$$\frac{d}{dt}\mathrm{Var}(x_t) = -2a\,\mathrm{Var}(x_t) + \sigma^2 \tag{54}$$

Setting the time derivative to zero:

$$\mathrm{Var}(x_\infty) = \frac{\sigma^2}{2a} \tag{55}$$

A.2 Discrete OU Increment Variance

For the exact discretization $x_{t+\Delta t} = e^{-a\Delta t}x_t + \int_t^{t+\Delta t} \sigma e^{-a(t+\Delta t-s)}dW_s$, the increment variance is:

$$\mathrm{Var}\left(\int_t^{t+\Delta t} \sigma e^{-a(t+\Delta t-s)}dW_s\right) = \sigma^2 \int_t^{t+\Delta t} e^{-2a(t+\Delta t-s)}ds \tag{56}$$

$$= \sigma^2 \frac{1 - e^{-2a\Delta t}}{2a} \tag{57}$$

A.3 Bond Price Formula Derivation

The bond price $P(t,T)$ satisfies the PDE:

$$\frac{\partial P}{\partial t} + \frac{\partial P}{\partial x_1}(-ax_1) + \frac{\partial P}{\partial x_2}(-bx_2) + \frac{1}{2}\frac{\partial^2 P}{\partial x_1^2}\sigma^2 + \frac{1}{2}\frac{\partial^2 P}{\partial x_2^2}\eta^2 + \frac{\partial^2 P}{\partial x_1 \partial x_2}\rho\sigma\eta - rP = 0 \tag{58}$$

with terminal condition $P(T,T) = 1$.

The exponential-affine ansatz:

$$P(t,T) = A(t,T)e^{-B_1(t,T)x_1 - B_2(t,T)x_2} \tag{59}$$

leads to ODEs for A, B_1, B_2, whose solutions yield equations (7) and the variance function $V(t,T)$.

B Code Structure

B.1 Module Organization

```
ir_models/
  __init__.py
  models/
    __init__.py
    vasicek.py
    g2pp.py            # Core model, ZC pricing
  estimation/
    __init__.py
    vasicek.py
    g2pp.py            # Kalman/EKF, MLE
  simulation/
    __init__.py
    vasicek.py
    g2pp.py            # Monte Carlo, plotting
```

B.2 Key Classes and Functions

Module	Class/Function	Purpose
`models/g2pp`	`G2ppModel`	Short rate simulation
	`G2ppZeroCouponPricing`	ZC bond and EURIBOR pricing
	`SimulationResult`	Container for simulation output
`estimation/g2pp`	`G2ppKalmanMLE`	Parameter estimation
	`EstimationResult`	Container for estimation output
	`_transition_matrices()`	State transition matrices
	`_discrete_ou_coefficients()`	Discrete increment std
`simulation/g2pp`	`plot_short_rate_paths()`	Visualization

Table 4: Key components of the G2++ implementation