IEOR4724 Final Project Report

Ruoyu Lin

Due Dec. 13th

1 Introduction

In this literature review study, we examine *Looking Forward to Backward-Looking Rates: A Modeling Framework for Term Rates Replacing LIBOR* by Lyashenko and Mercurio (SSRN-id330240)[1]. Observing the transition from LIBOR Market Model (LMM) based on IBOR-like rates to the Forward Market Model (FMM) based on risk-free rates (RFRs) published by central banks (*e.g.* SOFR, TONAR), the original paper proposes the extension of the original LIBOR model to adapt to this new environment.

In particular, the authors propose a simple expression that consolidates backward-looking in-arrears forward rates and forward-looking forward rates, making the forward rate dynamics well-defined at all times $t \in \mathbb{R}$, while the classical LIBOR forward rates are defined only for $t \leq T$ with T denoting the start of the accrual period for the LIBOR rate.

In this study, we attempt to present these key impacts and to facilitate the readers' understanding via literature review and via our own implementation of simulations according to the authors' formulation.

2 Key Concepts and Impact

2.1 Recap: LIBOR Market Model

In the classical LMM, the IBOR-like rates are defined via the following equation under no arbitrage assumptions:

$$L(t, T, T+\tau) = \frac{P(t,T) - P(t, T+\tau)}{\tau P(t, T+\tau)} \qquad \text{([C2]Eq2.29)}$$

where t, T, τ denote time of quotation, time of maturity/start of accrual period, and length of accrual period respectively. $L(t, T, T+\tau)$ is to be understood as the fixed rate that replicates the value of a forward bond:

$$P(t, T, T+\tau) = \frac{1}{1 + \tau L(t, T, T+\tau)} \qquad \text{([C2]Eq2.28)}$$

[1] [L-M]

Notice that in the classical LMM, this rate is defined $\forall t \leq T$ and ends its lifecycle precisely at $t = T$ maturity, as the guaranteed fixed forward rate starts to accrue. This mandates the shortcoming that the forward rate dynamics of $L(t, T, T + \tau)$ terminates at T, and all proceeding computation for the forward term rates with $t > T$ must be conducted from scratch again, inducing in significantly more computational cost.

2.2 RFR and Forward Market Model

The aforementioned shortcoming is effectively resolved in the FMM model proposed by Lyahenko and Mercurio. In particular, the following key concepts are introduced to complete this model:

- As the basis for studying $R_j(t)$ dynamics, the authors introduce *extended bond price* definitions and the *extended* T*-forward measure* as its consequence. In particular, bond price

$$P(t,T) = \mathbb{E}\left[\exp\left(-\int_t^T r(u)du\right)\middle|\mathcal{F}_t\right] = \frac{B(t)}{B(T)} \qquad \text{([L-M]Eq3)}$$

  is defined $\forall t \in \mathbb{R}$ instead of $\forall t \leq T$. Since $P(t,T)$ is $\mathcal{F}_T$-measurable, for $t > T$, $P(t,T)$ is effectively an observed constant but nonetheless well-defined.

- First we introduce *Setting-in-arrears rate* for $[T_{j-1}, T_j)$, which is backward-looking and known at the **end** of the corresponding accrual period and thus $\mathcal{F}_{T_j}$**-measurable**.

$$R(T_{j-1}, T_j) = \frac{1}{\tau_j}[P(T_{j-1}, T_j) - 1] \qquad \text{([L-M]Eq5)}$$
$$\text{where} \quad \tau_j = T_j - T_{j-1}$$

  which implies that unlike forward-looking LMM term rates, this value is not known at the beginning of the accrual period.

- The authors also reference *forward-looking rate* for $[T_{j-1}, T_j)$, which is $\mathcal{F}_{T_{j-1}}$**-measurable**, characterized by

$$F(T_{j-1}, T_j) = \mathbb{E}^{T_j}[R(T_{j-1}, T_j)|\mathcal{F}_{T_{j-1}}] \qquad \text{([L-M]Eq6)}$$

  This is equivalent to the $L(T_{j-1}, T_{j-1}, T_j)$ forward rate in the LMM.

- *Backward-looking in-arrears forward rate*, based on the definition of the setting-in-arrears rate, is equivalent to the breakeven value of the fix rate K_R in the swaplet (FRA) paying $\tau_j[R(T_{j-1}, T_j) - K_R]$ at T_j. This rate is more favored in the FMM framework, due to its capacity to address the continuous RFR compounding during the accrual period. In [L-M], this rate is characterized by

$$R_j(t) = \mathbb{E}^{T_j}\left[R(T_{j-1}, T_j)|\mathcal{F}_t\right] \qquad \text{([L-M]Eq7)}$$

  It is worth noting that at $t = T_{j-1}$, the forward-looking forward rate coincides with $R_j(t)$ at the end of $F(T_{j-1}, T_j)$'s lifecycle:

$$F(T_{j-1}, T_j) = R_j(T_{j-1})$$

- Finally, the *forward-looking forward rate* $F_j(t)$ is defined as the T_j-expectation made at time t on the forward-looking rate. This rate is equivalent to the breakeven rate of an FRA paying $\tau_j[F(T_{j-1}, T_j) - K_F]$ at T_j. The authors showed that this expectation turns out to be equivalent to the in-arrears forward rate:

$$\begin{aligned} F_j(t) &= \mathbb{E}^{T_j}\left[F(T_{j-1}, T_j)|\mathcal{F}_t\right] \\ &= \mathbb{E}^{T_j}\left\{\mathbb{E}^{T_j}\left[R(T_{j-1}, T_j)|\mathcal{F}_{T_{j-1}}\right]|\mathcal{F}_t\right\} \\ &= \mathbb{E}^{T_j}\left[R(T_{j-1}, T_j)|\mathcal{F}_t\right] \\ &= R_j(t) \end{aligned} \tag{[L-M]Eq11}$$

  $\forall t \leq T_{j-1}$.

- *Consolidation.* With concepts stated above, the authors combine the two forward rates into one due to the equivalence stated in the last point. In particular, we see that $R_j(t)$ and $F_j(t)$ satisfy

$$\begin{aligned} t \in (-\infty, T_{j-1}]: &\qquad F_j(t) = R_j(t) \quad \text{the same process;} \\ t \in (T_{j-1}, T_j]: &\qquad F_j(t) = F(T_{j-1}, T_j) \text{ constant,} \quad R_j(t) \text{ evolves;} \\ t \in (T_j, \infty]: &\qquad R_j(t) = R(T_{j-1}, T_j) \text{ constant.} \end{aligned}$$

  Given the simple extension on the bond definition $P(t, T_j)$ to $\forall t \in \mathbb{R}$ and consequently T_j forward measure, we see that $R_j(t)$ is a T_j-forward measure martingale.

Using this model established by the authors, FMM is a more complete model than the LMM as it preserves the dynamics of the forward-looking (LIBOR-like) rates, while providing additional information about the rate dynamics between the term-rate fixing/payment times. From T_{j-1} to T_j the rate $R_j(t)$ evolves instead of being constant compared to the LIBOR Market Model (LMM) in section 2.1. The model also has some nice properties the traditional LMM does not have, the completeness of the model, better pricing of future contracts and easier extension to a cross-curency interest rate model; model-implied dynamics of forward rates under the classic money-market measure, and not only under the discrete spot measure as in the LMM case.

In the following section, we aim to arrive at a better understanding along with the readers on the consolidated forward rate $R_j(t)$ by studying its dynamics using Monte Carlo simulations, and to consequently price some fixed-income products using these simulated rates.

3 Simulations

In this section we present the Monte Carlo implementations of the FMM rates in [L-M], and some consequent pricing routines of essential fixed-income products. Some notes on pseudocode notations are listed below for the understanding of the readers:

- Bolded characters that are not generic operations denote vectors/arrays.
- Pseudocode is Pythonic: when matrix algebra is available, loops are not used.

3.1 RFR Forward Rate Dynamics

3.1.1 T_j-Forward Measure

To account for $R_j(t)$'s decaying evolution in $[T_{j-1}, T_j]$ and its martingale property under T_j forward measure, we characterize

$$dR_j(t) = \sigma_j(t) g_j(t) dW_j(t) \tag{[L-M]Eq14}$$

where the decay function that effectively *forces the convergence* of $R_j(T_j)$, is defined as

$$g_j(t) = \min\left\{\frac{(T_j - t)^+}{T_j - T_{j-1}}, 1\right\} \tag{[L-M]Eq13}$$

See the following pseudocode block for the simulation algorithm of the $R_j(t)$ dynamics under T_j-forward measure.

Algorithm 1 `sim_r_path_T1`: Simulate RFR Dynamics under $\mathbb{Q}^{T_j}$ measure

```
Set: $dt$, $\sigma_j(\cdot)$, $T_{j-1}$, $T_j$, $T$, $n$.
$m \leftarrow \frac{T}{dt}$
$\mathbf{t} \leftarrow (0, dt, \ldots, (m-1)dt)$                    ▷ $\mathbf{t} \in \mathbb{R}^m$
$\mathbf{G} \leftarrow g_j(T_{j-1}, T_j, \mathbf{t})$                    ▷ $\mathbf{G} \in \mathbb{R}^m$
$\mathbf{Z} \leftarrow$ RandStdNorm$(n, m)$                    ▷ $\mathbf{Z} \in \mathbf{M}(\mathbb{R}_{n\times m})$
$\mathbf{dR} = \mathbf{Z} \circ (\sigma_j(\mathbf{t}) \circ \mathbf{G} \cdot \sqrt{dt})$    ▷ $\circ$ element-wise multiplication; $\mathbf{dR} \in \mathbf{M}(\mathbb{R}_{n\times m})$
return dR
```

where `RandStdNorm`(n, m) denotes a standard normal sample of dimension $n \times m$. Note that this is a straightforward implementation of the Gaussian T_j-forward martingale dynamics specified in ([L-M]Eq14). To obtain RFR forward rate levels instead of increments, simply set initial rate $R_j(0)$ and apply a cumulative summation. For situations where log-normal dynamics are pertinent (*e.g.* caps and floors pricing using Black's formula), apply an additional exponential transformation to obtain desired result.

3.1.2 Risk-neutral $\mathbb{Q}$ Measure

More importantly, built upon the T_j-forward measure, in [L-M] section 4.1, the authors also derived the RFR dynamics under the risk-neutral measure:

$$dR_j(t) = \sigma_j(t) g_j(t) \sum_{i=1}^{j} \rho_{i,j} \frac{\tau_i \sigma_i(t) g_i(t)}{1 + \tau_i R_i(t)} dt + \sigma_j(t) g_j(t) dW_j^Q(t) \tag{[L-M]Eq17}$$

A close examination of ([L-M]Eq17) demonstrates an important realization for the implementation perspective: the drift of $R_j(t)$ has inherent sequential dependence on $R_i(t) \quad \forall i < j$. Inevitably, this means that the simulation routine cannot be vectorized on both the time increment dimension $m = T/dt$ and the term dimension $J = |\mathbf{Ts}| - 1$.

Lyashenko and Mercurio also pointed out in ([L-M]Eq18) that ([L-M]Eq17) is equivalent to the analogue of changing the $i = 1$ in the summation subscript to $i = \eta(t) \triangleq \min\{j : T_j \geq t\}$. From the implementation perspective, this reduction is *already factored in* by the definition of $g_i(t)\ \forall i < j$, and the standardization of $i = 1$ for each j helps reducing the summation to matrix multiplication.

Finally, from the implementation perspective, one way to *standardize the summation superscript j is to replace j by J*, the number of accrual terms in the simulation, and setting $\rho_{i,j} = 0 \quad \forall i > j$. This operation is simply completed by setting the input correlation matrix to be lower-triangular. Thus, through these minor modifications, the optimized dynamics used in composing the following algorithm is expressed as follows:

$$dR_j(t) = \sigma_j(t) \circ g_j(t) \circ \left[\left(\boldsymbol{\rho} \cdot \frac{\boldsymbol{\tau} \circ \boldsymbol{\sigma}(t) \circ \mathbf{g}(t)}{1 + \boldsymbol{\tau} \circ \mathbf{R}(t)} \right) dt + dW_j^Q(t) \right] \tag{1}$$

$$\text{where} \quad \boldsymbol{\rho} = [\rho_{i,j}]_{1 \leq i,j \leq J}$$
$$dW_j^Q(t) = \texttt{sim\_r\_path\_T1}(dt, \sigma_j(\cdot), T_{j-1}, T_j, T, n)$$

Algorithm 2 `sim_r_path_Q`: Simulate RFR Dynamics under $\mathbb{Q}$ measure,

```
**Set**: $dt$, $n$, $\boldsymbol{\sigma} = [\sigma_j(\cdot)]'_{1 \leq j \leq J}$, $\boldsymbol{\rho} = [\rho_{i,j}]_{1 \leq i \leq j \leq J}$, $\mathbf{r}_0 = [r_j(0)]_{1 \leq j \leq J}$, $\mathbf{Ts} = (T_0, \ldots, T_J)$, $T$, $\mathbf{R}^{(0)} = [\mathbf{R}_j(0)]_{1 \leq j \leq J}$.
$m \leftarrow \frac{T}{dt}$
$\mathbf{t} \leftarrow (0, dt, \ldots, (m-1)dt)$    ▷ $\mathbf{t} \in \mathbb{R}^m$
$\boldsymbol{\tau} \leftarrow \mathbf{Ts}.\texttt{diff()}$    ▷ $\boldsymbol{\tau} \in \mathbb{R}^J$
for $j = 1, \ldots, J$ do
    Initialize $\mathbf{dR}_j = [\mathbf{0}]_m$; $\mathbf{R}_j = [\mathbf{R}^{(0)}(j) \cdot \mathbf{1}', [\mathbf{0}]'_{n \times m}]$ ▷ $\mathbf{dR}_j \in \mathbf{M}(\mathbb{R}_{n \times m})$, $\mathbf{R}_j \in \mathbf{M}(\mathbb{R}_{n \times m+1})$
    $\sigma_j(\cdot) = \boldsymbol{\sigma}(j)$
    $\mathbf{dW}_j \leftarrow \texttt{sim\_r\_path\_T1}(dt, \sigma_j(\cdot), \mathbf{Ts}_{j-1}, \mathbf{Ts}_j, T, n)$    ▷ $\mathbf{dW}_j \in \mathbf{M}(\mathbb{R}_{n \times m})$
    Initialize $\boldsymbol{\alpha}_j = [\mathbf{0}]_{n \times m}$    ▷ $\boldsymbol{\alpha}_j \in \mathbf{M}(\mathbb{R}_{n \times m})$
    for $k = 0, \ldots, m-1$ do
        $t \leftarrow k \cdot dt$
        $\boldsymbol{\alpha}_j(:, t) \leftarrow \sigma_j(t) g_j(t) \left[ \sum_{i=1}^{j} \boldsymbol{\rho}_{i,j} \frac{\boldsymbol{\tau}_i \boldsymbol{\sigma}_i(\boldsymbol{t}) g_i(t)}{1 + \boldsymbol{\tau}_i \mathbf{R}_i(:, t)} \right]$    ▷ $\mathbb{Q}$-Drift term; $\boldsymbol{\alpha}_j(:, t) \in \mathbb{R}^n$
        $\mathbf{dR}_j(:, t) \leftarrow \boldsymbol{\alpha}_j(t) + \mathbf{dW}_j(t)$    ▷ $\mathbf{dR}_j(:, t) \in \mathbb{R}^n$
        $\mathbf{R}_j(:, t + dt) \leftarrow \mathbf{R}_j(t) + \mathbf{dR}_j(t)$    ▷ $\mathbf{R}_j(:, t + dt) \in \mathbb{R}^n$
    end for
end for
return dR, R
```

In this implementation, when simulating $R_j(t)$, all previous term rates observations $R_i(t) \quad \forall i < j \forall t \in [0, T]$, for all n scenarios are already in memory. It's because of this aforementioned sequential dependence that we cannot eliminate the nested loop in the algorithm above.

3.2 $P_j(t)$ Term Bond Pricing

When a simulated RFR sample is obtained, we seek to obtain bond prices associated with the terms specified in the RFR simulation, *i.e.* $P_j(t) = P(t, T_j) \quad \forall j$. Here we propose a recurrence relation that is a straightforward extension from the definition of $R_j(t)$. Recall that

$$R_j(t) = \frac{1}{\tau_j}\left[\frac{P_{j-1}(t)}{P_j(t)} - 1\right] \qquad \text{([L-M]Eq10)}$$

Rewrite ([L-M]Eq10) renders

$$P_j(t) = \frac{P_{j-1}(t)}{1 + \tau_j R_j(t)}$$

which shows that given any sample path of $R_j(t)$ and a parameter $P(0, T_0)$ either from assumption or short rate models, we have determined bond prices $P_j(t)$ for all relevant T_j in the simulation for said sample path. In the special case where the first time tick is T_0, we have that $P_0(t) = P(T_0, T_0) = 1$, and all term bond prices are determined by the simulated $R_j(t)$ sample paths.

For completeness, the trivial algorithm is given as follows:

Algorithm 3 `rfr_bond_rec`: Compute term bond prices given $\mathbf{R}(t)\ \forall j$

```
Set: $\mathbf{\Theta}$, $dt$, $\mathbf{Ts}$, $T$, $p_0$.
$\mathbf{R} \leftarrow$ sim_r_paths_Q($\mathbf{\Theta}$)                ▷ $\mathbf{\Theta}$ relevant simulation parameters
$J, n, M \leftarrow$ shape($\mathbf{R}$)                ▷ (#acc. periods, #sample, #time inc.)
Initialize $\mathbf{P}= [0]_{J+1,n,M}$
$\mathbf{P}(0,:,:) \leftarrow p_0$
for $j = 1, \ldots, J+1$ do
    $\tau_j \leftarrow \mathbf{Ts}(j) - \mathbf{Ts}(j-1)$
    $\mathbf{P}(j,:,:) \leftarrow \frac{\mathbf{P}(j-1,:,:)}{1+\tau_j \cdot \mathbf{R}(j-1,:,:)}$
end for
```

Although Lyashenko and Mercurio did not specify bond pricing using $R_j(t)$, we choose to take this additional step to facilitate proceeding pricing routines. The main advantage of this additional step in our study rids most product pricing routine of the need of picking a short rate model once the first accrual period $[T_0, T_1]$ starts at T_0, as all term bond values are directly implied once $R_j(t)$ samples are given. This algorithm becomes specifically helpful in the later implementation of RFR swaps pricing routine and that of the RFR swap and cap.

3.3 RFR Futures Pricing

The pricing of an RFR futures under FMM is nothing more than the $\mathbb{Q}$-expectation of simulated $R_j(t)$ $\mathbb{Q}$-measure using the algorithm specified in section 3.2. For this contract in particular, we abbreviate the algorithm, and should the reader be interested, our code can serve as a reference.

3.4 RFR Fixed-Floating Swap Pricing

Corresponding to [L-M] section 6.2, we implement the following vanilla fixed-floating swap pricing routine.

Algorithm 4 `rfr_swap_mc`: Compute cap prices given using simulation

```
Set: $\mathbf{\Theta}$, $dt$, $\boldsymbol{\sigma}$, $\mathbf{Ts}$, $\boldsymbol{\rho}$, $T$, $K$, $n$, $r_0$, $p_0$.
$\mathbf{R} \leftarrow$ sim_r_paths_Q($\mathbf{\Theta}$)                        ▷ $\mathbf{\Theta}$ relevant simulation parameters
$\boldsymbol{\tau} \leftarrow \mathbf{Ts}$.diff()                                  ▷ $\boldsymbol{\tau} \in \mathbb{R}^J$
$J, n, M \leftarrow$ shape($\mathbf{R}$)                    ▷ (#acc. periods, #sample, #time inc.)
bond ← rfr_bond_rec($\boldsymbol{\sigma}, \mathbf{Ts}, \boldsymbol{\rho}, T, n, r_0, p_0, dt$)       ▷ Using $\mathbb{Q}$-dynamics
idx ← $\mathbf{Ts}/dt$
Initialize Swaplets ← $[\mathbf{0}]_{2,J,n,M}$
for $j = 1, \ldots, J$ do
    fra ← $\mathbf{bond}(j, :, : \mathbf{idx}(j)) \circ (\mathbf{R}(j-1, :, : \mathbf{idx}(j)) - K)$
    $\mathbf{Swaplets}(j-1, :, \mathbf{idx}(j)) \leftarrow$ fra
        ▷ Below line is to fill caplet price for the whole time horizon instead of only $[0, T_j]$
    $\mathbf{Swaplets}(j-1, :, \mathbf{idx}(j) :) \leftarrow \mathbf{Swaplets}(j-1, :, \mathbf{idx}(j))$
end for
Swap ← sum(Swaplets, axis = 0)                              ▷ summation over terms
Price ← mean(Swap, axis = 0)                                   ▷ mean over samples
return Swap, Swaplets, Price
```

Note that the RFR vanilla swap is similar to that of the classical vanilla swap in the LMM swap as defined in Lecture 2. The difference is like that which differentiates the Libor rate from the RFR backward-looking forward rate: each FRA/swaplet finishes accruing value at the end of the accrual period instead of the beginning of the accrual period.

Additionally, to obtain the breakeven swap rate, the RFR in-arrears rate has a simpler expression than the vanilla swap rate in LMM due to the recurrence relation specified in section 3.2 as a proxy of [L-M] Equation 10. Consider

$$\begin{aligned} S(t) &= \frac{\sum_{j=a+1}^{b} \tau_j P_j(t) R_j(t)}{\sum_{j=a+1}^{b} \tau_j P_j(t)} \\ &= \frac{\sum_{j=a+1}^{b} (P_{j-1}(t) - P_j(t))}{\sum_{j=a+1}^{b} \tau_j P_j(t)} && \text{([L-M]Eq10)} \\ &= \frac{P_a(t) - P_b(t)}{\sum_{j=a+1}^{b} \tau_j P_j(t)} && \text{(Telescope)} \end{aligned}$$

The implementation of the trivial algorithm that make use of previously implemented `rfr_bond_rec` is shown below for completeness:

Algorithm 5 `rfr_swap_rate`: compute breakeven swap rate using term bond prices

```
**Set**: $dt$, $\boldsymbol{\sigma}(\cdot)$, $\mathbf{Ts}$, $T$, $p_0$.
$\boldsymbol{\tau} \leftarrow \texttt{diff}(\mathbf{Ts})$
$\mathbf{P} \leftarrow \texttt{rfr\_bond\_rec}(\boldsymbol{\sigma}(\cdot), dt, T, \mathbf{Ts})$
swaprate $\leftarrow \frac{\mathbf{P}(0,:,:)-\mathbf{P}(-1,:,:)}{\mathbf{P}(1:,:,:)\cdot\boldsymbol{\tau}}$
```

3.5 RFR Cap Pricing

Finally, we implemented a pricing routine to price a RFR cap using caplet formula derived in [L-M] section 6.3. This algorithm is similar to that of the vanilla swap in section 3.4, except that (i) Black's formula is employed and (ii) forward and backward caplet variance are different. This algorithm requires several function calls to algorithm specified previously, such as the pricing routine for term bonds specified in section 3.2, and the log-normal version of the martingale measure dynamics of $R_j(t)$.

Right below is vectorized helper function implementation of Black's formula.

Algorithm 6 `Black`: Helper function: Black's formula

```
**Set**: $\mathbf{R}_j$, $K$, $\mathbf{v}$
$d_1 \leftarrow \frac{\log(\mathbf{R}_j/K)+\mathbf{v}/2}{\sqrt{\mathbf{v}}}$
$d_2 \leftarrow d_1 + \sqrt{\mathbf{v}}$
**return** $\mathbf{R} \cdot \texttt{normalcdf}(d_1) - K \cdot \texttt{normalcdf}(d_2)$
```

where similar to above, bolded symbols are vectors. With the Black's formula vectorized, below is the main pricing routine of the RFR cap:

Algorithm 7 `rfr_cap_mc`: Compute cap prices given using simulation

```
Set: $\mathbf{\Theta}$, $dt$, $\boldsymbol{\sigma}$, $\mathbf{Ts}$, $\boldsymbol{\rho}$, $T$, $K$, $n$, $r_0$, $p_0$.
$\mathbf{R} \leftarrow$ sim_r_paths_T1($\mathbf{\Theta}$, lognormal $= 1$)    ▷ $\mathbf{\Theta}$ relevant simulation parameters
$\boldsymbol{\tau} \leftarrow \mathbf{Ts}$.diff()    ▷ $\boldsymbol{\tau} \in \mathbb{R}^J$
$J, n, M \leftarrow$ shape($\mathbf{R}$)    ▷ (#acc. periods, #sample, #time inc.)
$\mathbf{t} \leftarrow (0, dt, \ldots, (m-1)dt)$    ▷ $\mathbf{t} \in \mathbb{R}^m$
$\mathbf{v}_f \leftarrow \boldsymbol{\sigma}^2 \cdot (\mathbf{Ts}(: -1) - \mathbf{t})$    ▷ Forward variance
$\mathbf{v}_b \leftarrow \boldsymbol{\sigma}^2 \cdot (\max(\mathbf{Ts}(: -1) - \mathbf{t}, 0) + \frac{\mathbf{Ts}(1:) - \max(\mathbf{Ts}(:-1), \mathbf{t})^3}{3 \cdot \boldsymbol{\tau}^2})$    ▷ Backward variance
bond$\leftarrow$ rfr_bond_rec($\boldsymbol{\sigma}, \mathbf{Ts}, \boldsymbol{\rho}, T, n, r_0, p_0, dt$)    ▷ Using $\mathbb{Q}$-dynamics
idx$\leftarrow \mathbf{Ts}/dt$
Initialize Caplets$\leftarrow [\mathbf{0}]_{2,J,n,M}$
for $j = 1, \ldots, J$ do
    $C_f \leftarrow$ bond$(j, :, :$ idx$(j-1)) \circ$ Black($\mathbf{R}(j-1, :, :$ idx$(j-1)), K, \mathbf{v}_f(j-1, :$ idx$(j-1))$)
    $C_b \leftarrow$ bond$(j, :, :$ idx$(j)) \circ$ Black($\mathbf{R}(j-1, :, :$ idx$(j)), K, \mathbf{v}_f(j-1, :$ idx$(j))$)
    Caplets$(0, j-1, :, :$ idx$(j-1)) \leftarrow C_f$
    Caplets$(1, j-1, :,$ idx$(j) :) \leftarrow C_b$
    ▷ Below 2 lines are to fill caplet price for the whole time horizon instead of only $[0, T_j]$
    Caplets$(0, j-1, :,$ idx$(j-1) :) \leftarrow$ Caplets$(0, j-1, :,$ idx$(j-1))$
    Caplets$(1, j-1, :,$ idx$(j) :) \leftarrow$ Caplets$(1, j-1, :,$ idx$(j))$
end for
Cap$\leftarrow$ sum(Caplets, axis $= 1$)    ▷ summation over terms
Price$\leftarrow$ mean(Cap, axis $= 1$)    ▷ mean over samples
return Cap, Caplets, Price
```

where the volatility terms for forward and backward caps followed [L-M] section 6.3 exactly. A close examination of the **idx** specification in the above algorithm would show that the forward caplets evaluation becomes constant exactly at the *beginning* of the relevant accrual period, and the backward caplets evaluation becomes constant exactly at the *end* of the relevant accrual period. This is exactly the idiosyncratic feature that differentiates the two kinds of caps.

4 Results

All simulation Python code are implemented and stored in two modules, `rfr.py` and `contracts.py`, where the former holds the fundamental simulation routines for RFR rates and the latter those of different products specified in [L-M]. Selected simulation outputs will be included in Appendix A, and all implemented modules will be included in Appendix B. We as well invite the readers to experiment with our implementation.

Appendix A: Demo Outputs

In this appendix we provide some sample simulation outputs as a reference for the readers.

Figure 1: Demo: LMM forward term rate vs. FMM forward term rate

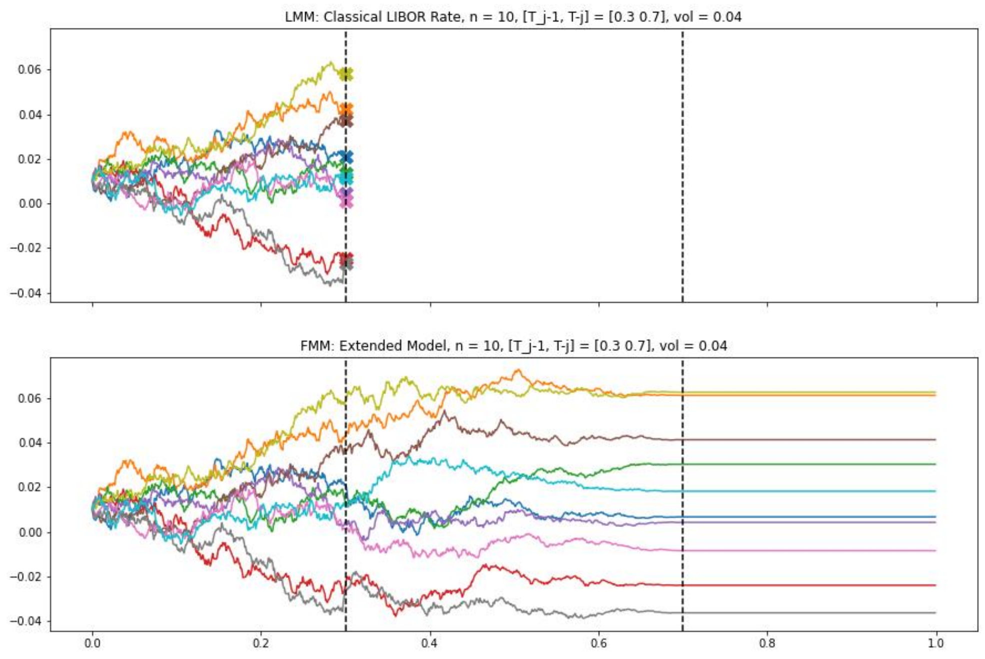


Figure 2: $R_j(t)$ dynamics and respective bond price under $\mathbb{Q}$

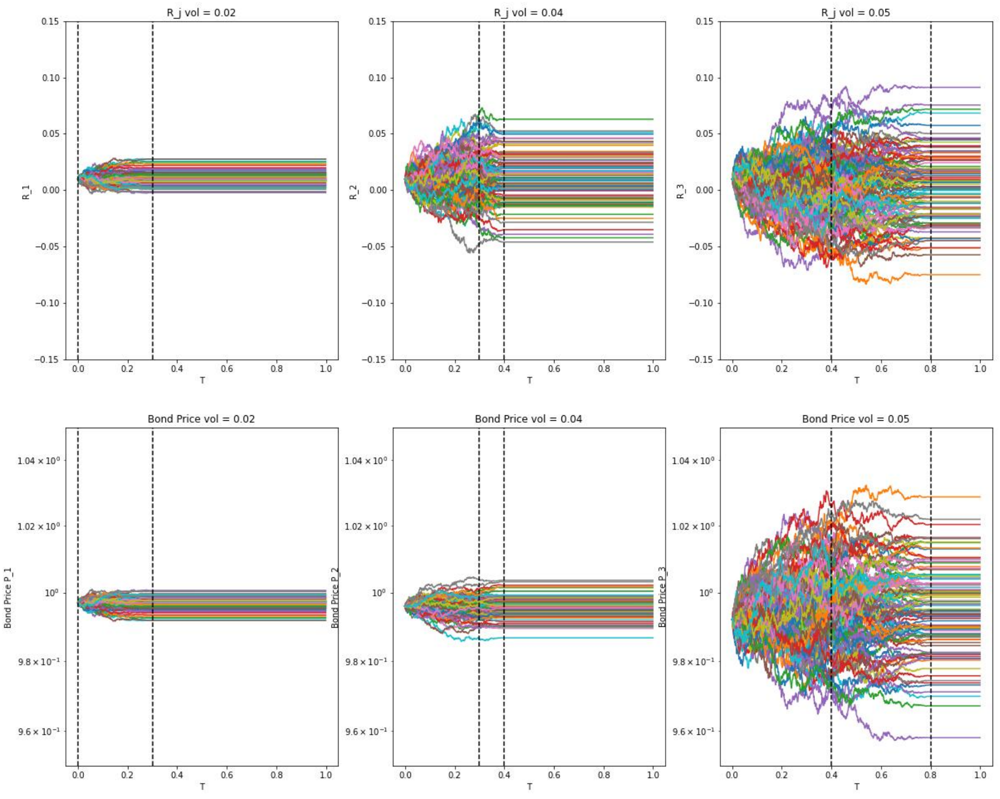


Figure 3: R_j Futures Pricing

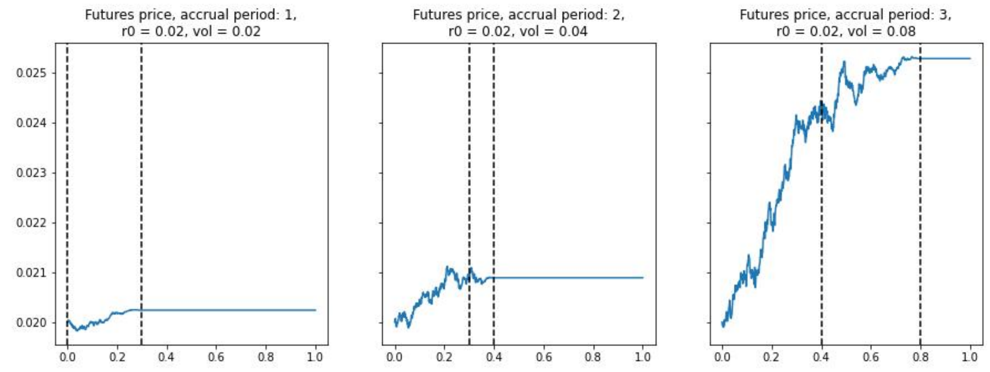


Figure 4: R_j Vanilla Swaps Pricing

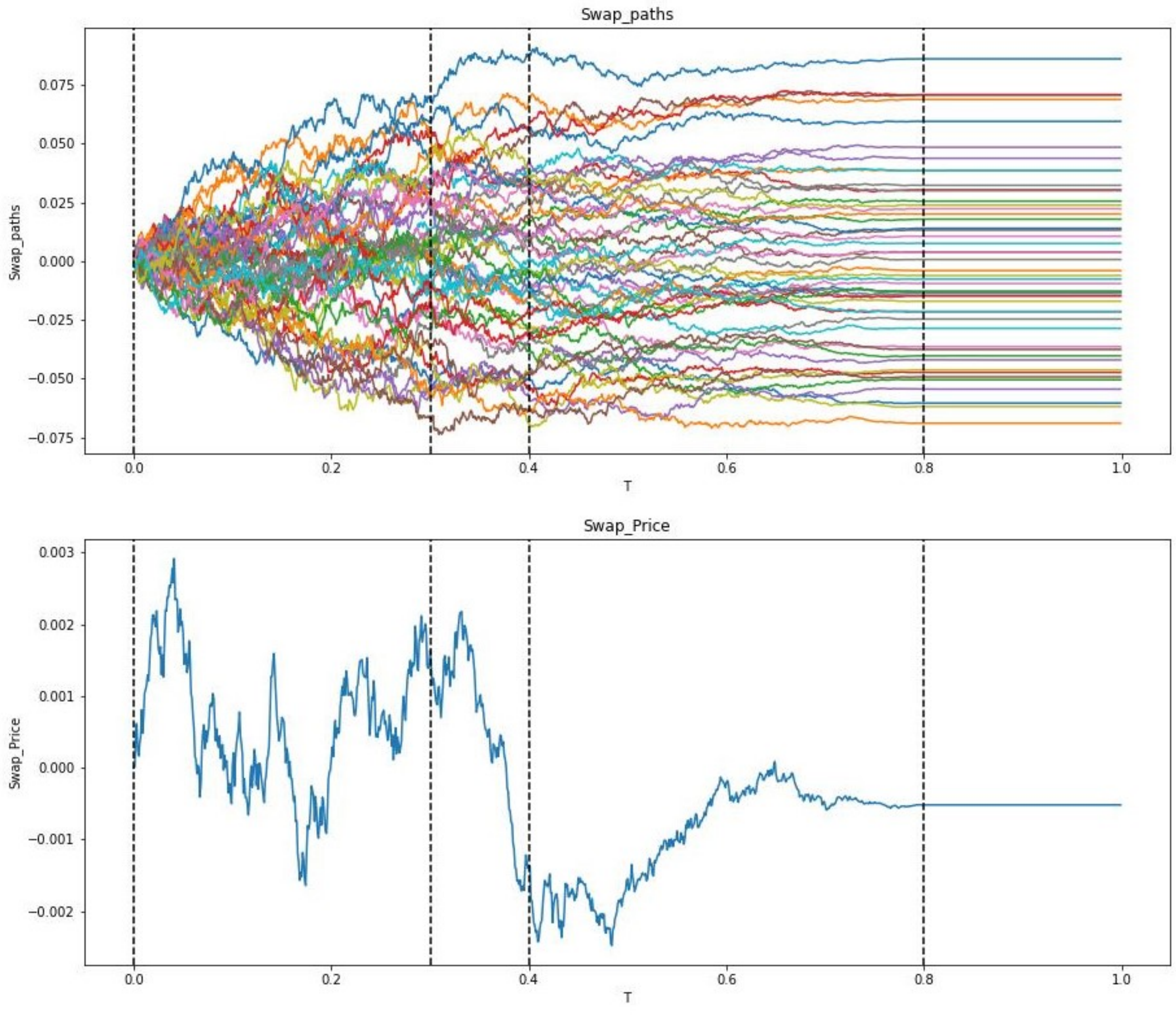
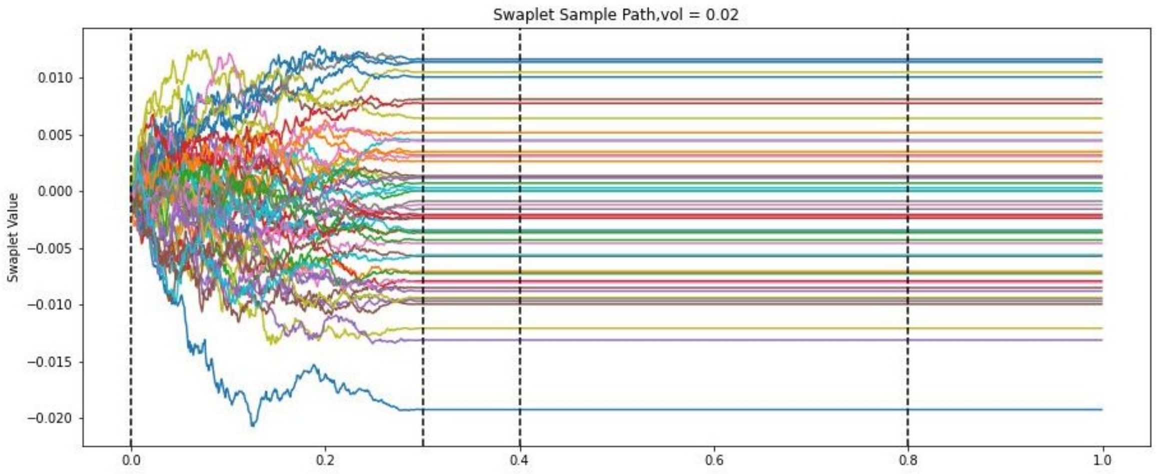
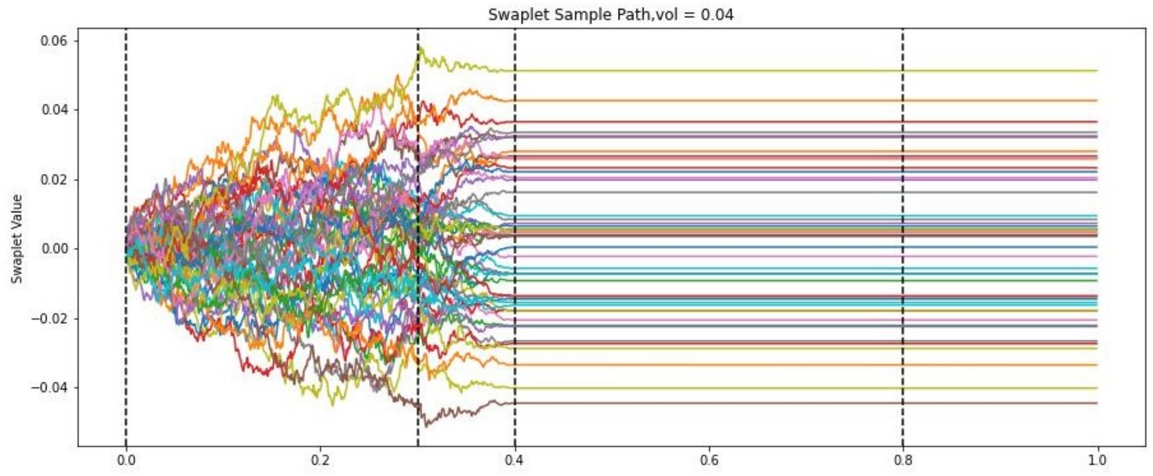


Figure 5: R_j Vanilla Swaplets Pricing





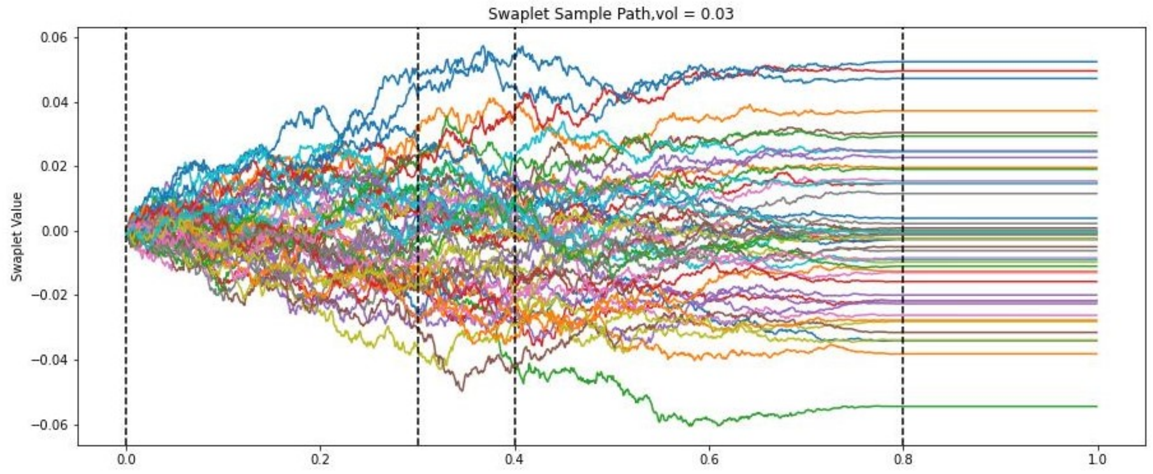


Figure 6: R_j Vanilla Swaps Rates Computation

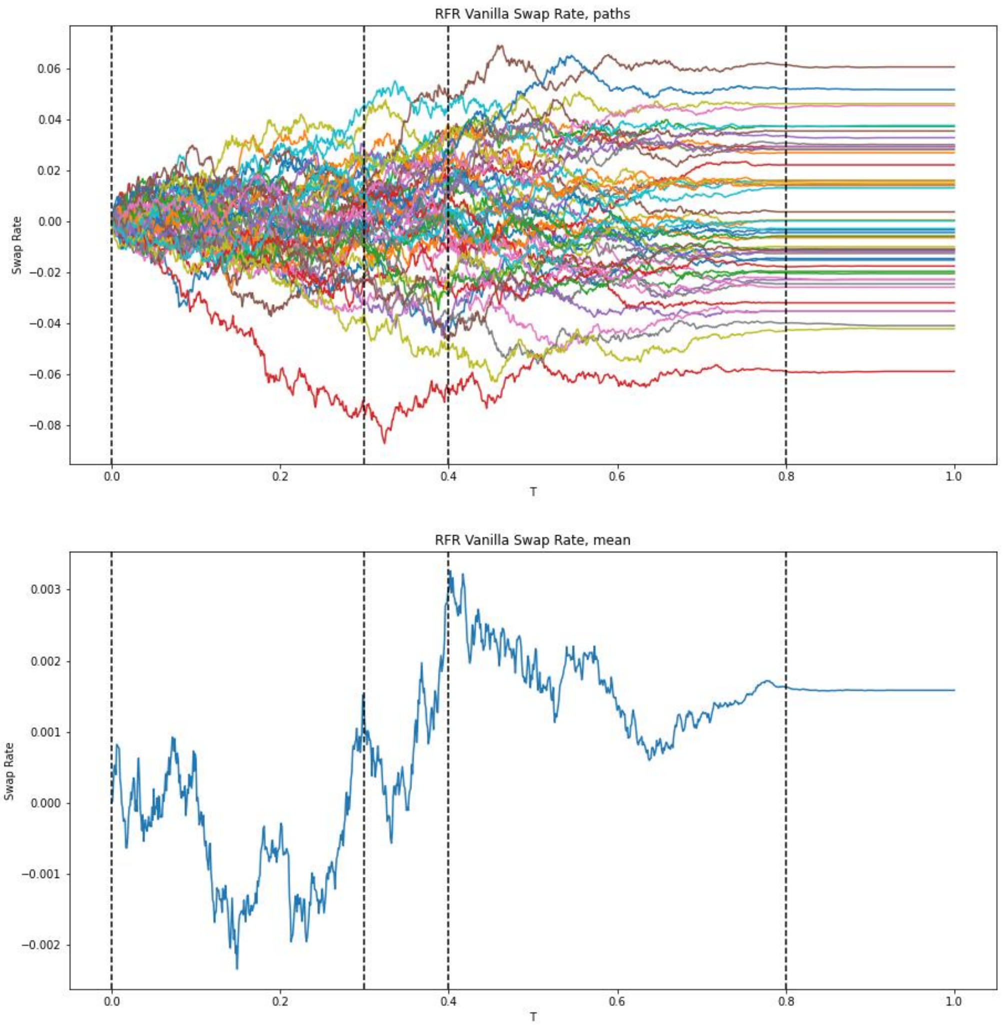
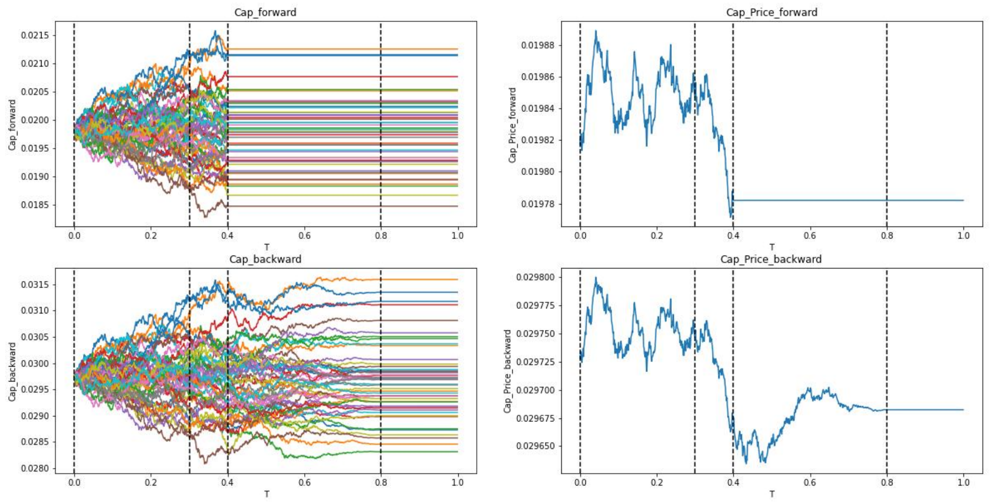


Figure 7: R_j Vanilla Caps Pricing



Appendix B: Python Implementation

Rates module: `rfr.py`

The following module is constructed to simulate RFR rates, in particular, $R_j(t)$, backward-looking in-arrears forward rate under different measures.

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats


def rfr_decay(T0, T1, t):
    return np.minimum(np.maximum(T1 - t, 0) / (T1 - T0), 1)


def sim_r_paths_T1(vol, T0, T1, T, n=1000, r0=0, dt=1e-4, lognormal: bool =
    False):
    m = int(T / dt)
    if not isinstance(vol, float):
        if len(vol) != m:
            raise ValueError(
                f"If␣input␣dynamic␣volatility,␣then␣vol␣must␣equal␣simulation
                    ␣length.␣len(vol)={len(vol)},␣m␣=␣{m}."
            )
    else:
        vol = np.ones(m) * vol
    t = np.arange(0, T, dt)
    G = rfr_decay(T0, T1, t)
    Z = np.random.randn(n, m)
    dR = np.multiply(Z, vol * G * np.sqrt(dt))
    R = dR.cumsum(axis=1) + r0
    R = np.hstack((np.ones((n, 1)) * r0, R))
    if lognormal:
        R = np.exp(R)
    return dR, R


def sim_r_paths_Q(
    vol: np.ndarray,
    Ts: np.ndarray,
    rhos: np.ndarray,
    T: float,
    n: int = 1000,
    r0: float = 0,
    dt: float = 1e-4,
):
    """
    RFR rate under Q risk neutral measure for P accrual periods.


    Parameters:
    --------------
    vol: np.ndarray: volatility matrix of shape (J, m). Each colums is
    the discrete volatility function for accrual period i.

    Ts: Iterable: list of accrual times. i.e. [T_0, T_1, ..., T_J]

    rhos: np.ndarray: correlation matrix between Wiener processes
       associated
    with different accrual periods.

    T: float: length of simulation period. Must be greater than Ts[-1].

    n: int: number of samples

    r0: np.ndarray: vector of starting rates for each accrual period

    dt: float: time increment


    Returns:
    --------------
    np.ndarray: a tensor of dimension (#periods, #samples, #time points)
       containing
    RFR increments dRj for each period (Tj-1, Tj).
    """
    # make correlation lower triangular for better vectorization
    rhos = np.tril(rhos)
    M = int(T / dt)
    # calculate accrual times
    taus = np.diff(Ts)
    J = vol.shape[0]
    dR = np.zeros((J, n, M))
    R = np.zeros((J, n, M + 1))
    for j in range(1, J + 1): # for each accrual period
        # initialize dRj, Rj and drift
        dRj = np.zeros((n, M))
        Rj = np.zeros((n, M + 1))
        Rj[:, 0] = r0
        alpha = np.zeros((n, M))
        # compute diffusion term (martingale)
        dWj, _ = sim_r_paths_T1(vol[j - 1, :], Ts[j - 1], Ts[j], T, n, r0,
            dt)
        # compute decay function used in drift
        gj = rfr_decay(Ts[j - 1], Ts[j], np.arange(0, T, dt))

        for m in range(M): # for each time increment
            # vectorized decay function
            decay = np.array(
                [rfr_decay(Ts[i - 1], Ts[i], m * dt) for i in range(1, J + 1)
                    ]
            )

            # vectorized previous accrual effects
            prev_period_term = (taus * vol[:, m] * decay).reshape(-1, 1) / (
                1 + taus.reshape(-1, 1) * Rj[:, m]
            )
            # since correlation matrix is lower triangular, can use inner
                product
            # thus all periods after j has 0 effect
            np.copyto(
                alpha[:, m],
                vol[j - 1, m] * gj[m] * (rhos[j - 1] @ prev_period_term) * dt
                    ,
            )
            # Rj increment given by drift + diffusion
            np.copyto(dRj[:, m], alpha[:, m] + dWj[:, m])
            # Rj level given by previous level + increment
            np.copyto(Rj[:, m + 1], Rj[:, m] + dRj[:, m])

        np.copyto(dR[j - 1], dRj)
        np.copyto(R[j - 1], Rj)
    return dR, R


def debug(name="Var", *arg):
    for a in arg:
        if hasattr(a, "shape"):
            printout = a.shape
        else:
            printout = a
        if hasattr(a, "__name__"):
            name = a.__name__
        else:
            name = name
        print()
        print(f"{name}:␣{printout},␣type:␣{type(a)}")
        print()


def random_corr_mat(
    full_random=False, n: int = 2, eigs: np.ndarray = None, seed=None, decay
        =False
):
    if n == 1:
        return np.array([[1.0]])
    if not full_random:
        assert eigs is not None
    else:
        np.random.seed(seed)
        eigs = np.random.random(n)
        if decay:
            eigs = np.cumprod(eigs)
        eigs = eigs / eigs.sum() * n
    return stats.random_correlation.rvs(eigs, seed)
```

Products module: `contracts.py`

The following module is constructed to make use of `rfr.py` to price some common derivative contracts with RFR rate $R_j(t)$.

```
import rfr
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats

def rfr_bond_rec(
    vol: np.ndarray=None,
    Ts: np.ndarray=None,
    rhos: np.ndarray=None,
    T: float=None,
    n: int = 1000,
    r0: np.ndarray = 0,
    p0: np.ndarray=1,
    dt: float = 1e-4,
    R: np.ndarray = None
):
    if R is None:
        try:
            _, R = rfr.sim_r_paths_Q(vol, Ts, rhos, T, n, r0, dt)
        except:
            raise RuntimeError(
                ("No␣RFR␣tensor␣passed,␣nor␣are␣specified␣arguments␣"
                 "valid␣to␣construct␣RFR␣samples."
                )
            )
    taus = np.diff(Ts)
    J, n, M = R.shape
    P = np.zeros((J+1,n,M))
    P[0,:,:] = p0
    for j in range(1, J+1):
        P[j,:,:] = P[j-1,:,:] / (1 + taus[j-1] * R[j-1,:,:])
    return P

def rfr_futures_mc(
    vol: np.ndarray,
    Ts: np.ndarray,
    rhos: np.ndarray,
    T: float,
    n: int = 1000,
    r0: float = 0,
    dt: float = 1e-4,
    R: np.ndarray = None
):
    if R is None:
        try:
            _, R = rfr.sim_r_paths_Q(vol, Ts, rhos, T, n, r0, dt)
        except:
            raise RuntimeError(
                ("No␣RFR␣tensor␣passed,␣nor␣are␣specified␣arguments␣"
                 "valid␣to␣construct␣RFR␣samples."
                )
            )
    J, n, M = R.shape
    M -= 1
    futures = R.mean(axis = 1)
    convexity = (R - futures.reshape(J, 1, M+1)).mean(axis=1)
    return convexity, futures

def rfr_vanilla_swap_mc(
    vol: np.ndarray,
    Ts: np.ndarray,
    rhos: np.ndarray,
    T: float,
    K: float,
    n: int = 1000,
    r0: np.ndarray = 0,
    p0: float=1,
    dt: float = 1e-4,
    R: np.ndarray = None
):
    if R is None:
        try:
            _, R = rfr.sim_r_paths_Q(vol, Ts, rhos, T, n, r0, dt)
        except:
            raise RuntimeError(
                ("No␣RFR␣tensor␣passed,␣nor␣are␣specified␣arguments␣"
                 "valid␣to␣construct␣RFR␣samples."
                )
            )

    J, n, M = R.shape
    Ts = Ts.reshape(-1, 1)
    M -= 1
    if vol.shape[1] == M:
        vol = np.append(vol, np.zeros((vol.shape[0], 1)), axis = 1)
    elif vol.shape[1] == M+1:
        pass
    else:
        raise RuntimeError("vol␣invalid␣shape.")

    disc_bond = rfr_bond_rec(
        vol=vol[:,:-1], Ts=Ts.flatten(),
        rhos=rhos, T=T, n=n, r0=0,
        p0=p0, dt=dt
    )
    Swaplet = np.zeros((J, n, M))
    idx = (Ts / dt).astype(int).flatten()
    for j in range(J):
        fra = disc_bond[j+1,:,:idx[j+1]] * (R[j,:,:idx[j+1]] - K)
        np.copyto(Swaplet[j,:,:idx[j+1]], fra)
        # fill with last realized value
        Swaplet[j,:,idx[j+1]:] = Swaplet[j,:,idx[j+1]-1].reshape(-1,1)

    Swap = Swaplet.sum(axis=0)
    Price = Swap.mean(axis=0)
    return Swap, Swaplet, Price

def rfr_swap_rate(
    vol: np.ndarray,
    Ts: np.ndarray,
    rhos: np.ndarray,
    T: float,
    n: int = 1000,
    p0: float=1,
    dt: float = 1e-4,
):
    M = int(T / dt)
    taus = np.diff(Ts)
    Ts = Ts.reshape(-1, 1)
    if vol.shape[1] == M:
        vol = np.append(vol, np.zeros((vol.shape[0], 1)), axis = 1)
    elif vol.shape[1] == M+1:
        pass
    else:
        raise RuntimeError("vol␣invalid␣shape.")

    disc_bond = rfr_bond_rec(
        vol=vol[:,:-1], Ts=Ts.flatten(),
        rhos=rhos, T=T, n=n, r0=0,
        p0=p0, dt=dt
    )

    s = disc_bond.shape
    swap_rate = (disc_bond[0,:,:] - disc_bond[-1,:,:]) / (disc_bond[1:,:,:].
        reshape(s[1], s[2], s[0]-1).dot(taus))
    return swap_rate

def Black(
    Rj:np.ndarray,
    K:np.ndarray,
    v:np.ndarray
):
    # R (J, n, M+1)
    # v (J, M+1)
    Phi = stats.norm.cdf
    d1 = (np.log(Rj / K) + v / 2) / (np.sqrt(v))
    # print("d1", d1)
    # print(Phi(d1))
    d2 = d1 + np.sqrt(v)
    # print("d2", d2)
    # print(Phi(d2))
    return Rj * Phi(d1) - K * Phi(d2)

def rfr_cap_mc(
    vol: np.ndarray,
    Ts: np.ndarray,
    rhos: np.ndarray,
    T: float,
    # kind: str= "f",
    K: float = 1,
    n: int = 1000,
    p0: float=1,
    r0: np.ndarray = np.log(1e-2),
    dt: float = 1e-3,
    R: np.ndarray = None
):
    if R is None:
        try:
            _ , R = zip(*[rfr.sim_r_paths_T1(vol[j], Ts[j], Ts[j+1], T, n, r0
                , dt, lognormal=True)
                 for j in range(len(Ts)-1)])
            # dR = np.array(dR)
            R = np.array(R)
        except:
            raise RuntimeError(
                ("No␣RFR␣tensor␣passed,␣nor␣are␣specified␣arguments␣"
                 "valid␣to␣construct␣RFR␣samples."
                )
            )

    taus = np.diff(Ts)
    J, n, M = R.shape
    Ts = Ts.reshape(-1, 1)
    M -= 1
    if vol.shape[1] == M:
        vol = np.append(vol, np.zeros((vol.shape[0], 1)), axis = 1)
    elif vol.shape[1] == M+1:
        pass
    else:
        raise RuntimeError("vol␣invalid␣shape.")

    t = np.arange(0, T+dt, dt)
    vf = vol**2 * ((Ts[:-1]-t)) # -t?
    vb = vol**2 * (
        np.maximum(
            Ts[:-1]-t, 0
        ) + (
            (Ts[1:] - np.maximum(Ts[:-1], t))**3 / (taus**2 * 3).reshape(-1,
                1)
        )
    )
    # vf, vb shapes (3, 1001) = (J, M+1)
    # disc_bond = rfr_bond_rec(Ts=Ts.flatten(), R=R)
    disc_bond = rfr_bond_rec(
        vol=vol[:,:-1], Ts=Ts.flatten(),
        rhos=rhos, T=T, n=n, r0=0,
        p0=p0, dt=dt
    )
    # rfr.debug("disc_bond", disc_bond)
    idx = (Ts / dt).astype(int).flatten()

    Caplet = np.zeros((2, J, n, M))
    for j in range(J):

        Cf = disc_bond[j+1,:,:idx[j]] * Black(R[j,:,:idx[j]], K, vf[j][:idx[
            j]]) # (n, Tj-1 / dt + 1)
        Cb = disc_bond[j+1,:,:idx[j+1]] * Black(R[j,:,:idx[j+1]], K, vb[j][:
            idx[j+1]]) # (n, Tj / dt + 1)
        np.copyto(Caplet[0,j,:,:idx[j]], Cf)
        np.copyto(Caplet[1,j,:,:idx[j+1]], Cb)
        # fill with last realized value
        Caplet[0,j,:,idx[j]:] = Caplet[0,j,:,idx[j]-1].reshape(-1,1)
        Caplet[1,j,:,idx[j+1]:] = Caplet[1,j,:,idx[j+1]-1].reshape(-1,1)

    Cap = Caplet.sum(axis=1)
    Price = Cap.mean(axis=1)
    return Cap, Caplet, Price
```

References

[L-M] Lyashenko, Andrei and Mercurio, Fabio, *Looking Forward to Backward-Looking Rates: A Modeling Framework for Term Rates Replacing LIBOR* (February 6, 2019). Available at SSRN: https://ssrn.com/abstract=3330240 or http://dx.doi.org/10.2139/ssrn.3330240

[C2] Capriotti, Luca, *Lecture 2: Fixed income instruments and Markets.* Columbia University, IEOR4724, Term Structure and Credit Models.