Log-Gaussian-Mixture Stochastic Liquidity Kyle-Back Model Code Documentation

Brad Mostowski

February 2023

1 Executive Summary

This document is a partial summary of the Kyle-Back code found here which is based on the paper of [**?**] found here. Specifically I explain how the optimal transport map $h(x)$ (and hence stock prices) is computed by the code in the case when the logarithm of the final stock price is assumed to be distributed according a Gaussian mixture.

2 Statement of the Problem

We offer a brief summary of relevant results from [**?**] concerning the Kyle-Back model with stochastic liquidity. Suppose a stock is to be traded publicly from time 0 to time T, and that at time T the true value $\tilde{v}$ of the stock is to be revealed — thus the stock price will jump to $\tilde{v}$ if it is not already valued at that price at time T. If the market maker perceives beforehand that $\tilde{v} \sim \nu$, and if at time the t the processes ξ_t and Σ_t are such that $\xi_t = \xi$ and $\Sigma_t = \Sigma$, then using Feynmann-Kac representation, the price at time t of the stock is given by

$$R_\xi(\Sigma, \xi) = \int_{-\infty}^{\infty} \phi(x) h(\xi + \sqrt{\Sigma} x)\, dx \tag{1}$$

$$= \int_0^1 h(\xi + \sqrt{\Sigma}\Phi^{-1}(x))\, dx \tag{2}$$

where $\Phi(x)$ and $\phi(x)$ are the cumulative distribution function and probability distribution function of the standard normal distribution respectively. In fact, given a European option on the stock which expires at time T and gives payout $H : \mathbb{R} \to \mathbb{R}$, we have that the price of that option is given by

$$\Pi_H(\Sigma, \xi) = \int_0^1 H\big(h(\xi + \sqrt{\Sigma}\Phi^{-1}(x))\big)\, dx \tag{3}$$

For example, the price of a call option would be

$$C(\Sigma, \xi) = \int_0^1 \big(h(\xi + \sqrt{\Sigma}\Phi^{-1}(x)) - K\big)^+ \, dx \tag{4}$$

When the vector $x = \{x_i\}_{i=1}^N$ is chosen to be a suitable mesh on $[0, 1]$, then equation 3 can be handled numerically by computing

$$\frac{1}{N}\sum_{i=1}^{N} h\big(\xi + \sqrt{\Sigma}\Phi^{-1}(x_i)\big) \tag{5}$$

Our challenge is the following: *How do we vectorize the computation of 5?* When h is vectorizable, then this is easy with the python libraries `numpy` & `scipy`, since given vector x, I can do `xi + np.sqrt(Σ)*norm.ppf(x)` and then pass the result through h. However, equation 9 in general does not yield an explicit expression for $h(x)$, meaning it is not obvious how to vectorize $h(x)$.

3 Description of the Mathematics

In the Log-Gaussian-mixture (LGM) scenario, the market maker assumes the distribution ν of $\tilde{v}$ is the mixture of log-normal distributions given by $\nu \sim \{X_{m_i,\sigma_i}; w_i\}_{i=1}^N$, where

$$X_{m_i,\sigma_i} = m_i \exp\Big(-\frac{1}{2}\sigma_i^2 + \sigma_i Z\Big)$$

with Z's being a standard normal random variable and $\sum_{i=1}^N w_i = 1$ with $w_i \geq 0$ for all i. In this scenario, the log-price is a Gaussian mixture, $\ln(\nu) \sim \{Y_i; w_i\}_{i=1}^N$ where

$$Y_i = \ln(m_i) - \frac{1}{2}\sigma_i^2 + \sigma_i Z \tag{6}$$

Recall that $F_\nu(h(x)) = \Phi(x)$. Also,

$$\begin{aligned} F_\nu(x) &= P(\tilde{v} \leq x) &(7)\\ &= P(\ln(\tilde{v}) \leq \ln(x)) = F_{\ln(\nu)}(\ln(x)) &(8)\end{aligned}$$

In view of the above, we have that the optimal transport map $h(x)$ satisfies the equation

$$\Phi(x) = \sum_{i=1}^{N} w_i \Phi\Big(\frac{\ln(h(x)) - ln(m_i) + \frac{1}{2}\sigma_i^2}{\sigma_i}\Big) \tag{9}$$

To find $h(x)$, we set $\Phi_x = \Phi(x)$, $\mu_i = \ln(m_i) - \frac{1}{2}\sigma_i^2$ and use nonlinear root-finding methods on the function

$$f(\gamma; x) = \sum_{i=1}^{N} w_i \Phi\Big(\frac{\gamma - \mu_i}{\sigma_i}\Big) - \Phi_x \tag{10}$$

When γ is found, then $h(x) = e^\gamma$. Implementations of both the secant method and bisection method are made in python and vectorized using numpy. The function `hGM`, which handles the computation of $h(x)$ in the Gaussian mixture case, accepts a numpy 1d array x as its (first) argument and outputs the result of applying h component-wise to the elements of x.

4 Description of the Algorithm

4.1 Bisection Method

Unless otherwise stated, in this section, if $x \in \mathbb{R}^n$ and $f : \mathbb{R} \to \mathbb{R}$, then $f(x)$ is the vector whose element in the ith index is $f(x_i)$.

The primary challenging aspect to designing the bisection method is determining the boundary vectors that are used before beginning the bisection iteration steps. Since we are working with a vectorized environment, we must determine vectors a and b so that $a_i < \gamma_i < b_i$ for all i is guaranteed, where γ is such that $f(\gamma_i; x_i) = 0$ for all i. We must also keep in mind while performing the bisection method iterative steps that, in general, the kind of update to the bounding interval may vary from index to index — that is, for some indices that the update $b_i \leftarrow c_i$ must be performed, while for others the update $a_i \leftarrow c_i$ must be performed. We use an iterative method that starts with a naive guess at the root. Based on whether the root undershoots or overshoots, we guess a suitable interval. If the interval either entirely undershoots or entirely overshoots the root, we will shift until it contains the root. By utilizing numpy's array features, this entire process is vectorized.

To motivate the initial naive guess at the root, first, assume $N = 1$ so $w_1 = 1$. Then,

$$f(\gamma; x) = \Phi\Big(\frac{\gamma - \mu_1}{\sigma_1}\Big) - \Phi_x \tag{11}$$

has the root $\gamma = \sigma_1 x - \mu_1$. Our approach to the choice of initial boundary vectors a and b relies on the naive intuition that when $N > 1$ (as it will be in practice), the initial guess of $\gamma_0 = \sigma_i x - \mu_i$, where $i = \max_j w_j$ ought to be close to the actual root of f.

The initialization process is as follows:

1. Given the input vector x, compute $i = \max_j w_j$ and then define the vector $\gamma_0 = \sigma_i x - \mu_i$
2. Evaluate $f(\gamma_0)$. For all indices $i\ldots$
   (a) If $f(\gamma_0^{(i)}) < 0$, set $a_i \leftarrow \gamma_0^{(i)}$ and $b_i \leftarrow ln(2) + \gamma_0^{(i)}$.
   (b) If $f(\gamma_0^{(i)}) > 0$, set $a_i \leftarrow \ln(1/2) + \gamma_0^{(i)}$ and $b_i \leftarrow \gamma_0^{(i)}$.
3. Compute $f(a)$ and $f(b)$. While any of $\{a_i : f(a_i) > 0\}$ and $\{\beta_i : f(b_i) < 0\}$ are non-empty
   (a) If $f(a_i) > 0$, then put $b_i \leftarrow a_i$ and $a_i \leftarrow \ln(1/2) + a_i$
   (b) If $f(b_i) < 0$, then put $a_i \leftarrow b_i$ and $b_i \leftarrow \ln(2) + b_i$

We add by $\ln(2)$ and $\ln(1/2)$ rather than multiply by 2 and 1/2 because roots γ of f represents values of $\ln(h(x))$. Thus, adding by $\ln(2)$ (resp. $\ln(1/2)$) amounts to setting the upper bound of our guess at $h(x)$ as double the value implied by the naive initial guess (resp. setting the lower bound of our guess at $h(x)$ as half the value implied by the initial guess). Since $f(\gamma)$

is strictly increasing, the while loop of step 3 is guaranteed to terminate thus the vectors a and b by this process are guaranteed to be such that $a_i \leq \gamma_i \leq b_i$.

After the vector is initialized, we then proceed with the vectorized bisection method iteration steps:

1. For $k = 1$ until $iter$ do ...
   (a) Set $c \leftarrow \dfrac{a+b}{2}$. For each index i...
      i. If $f(c_i)f(b_i) > 0$, then set $b_i \leftarrow c$.
      ii. If $f(c_i)f(b_i) < 0$, then set $a_i \leftarrow c$.
2. Return $\exp\left(\dfrac{a+b}{2}\right)$

4.2 Secant Method

The secant method is much easier to explain than the bisection method. We feed the secant method two initial guesses before performing the iteration steps. The process for choosing the initial guesses is similar to how the naive initial guess at the solution in bisection method was made. The process works as follows:

1. Put $i \leftarrow \max_k w_k$ and then $j \leftarrow \max_{k \neq i} w_k$. Then, set $\gamma_0 \leftarrow \sigma_j x + \mu_j$ and $\gamma_1 \leftarrow \sigma_i x + \mu_i$
2. For k from 1 until $iter$, do ...
   (a) $\gamma_{k+1} = \gamma_k - f(\gamma_k)\dfrac{\gamma_k - \gamma_{k-1}}{f(\gamma_k) - f(\gamma_{k-1})}$
3. return γ_{iter+1}