Computing American options by Exercise Rate Optimization

March 3, 2020

Premia 22

1 Description of the algorithm

Let $(Y_t)_{t\in[0,T]}$ denote the discounted cash-flow process corresponding to an American option. The price of the option is, hence,

$$v = \sup_{\tau} \mathbb{E}\left[Y_{\tau\wedge T}\right], \tag{1}$$

where τ runs over all stopping times and $\mathbb{E}$ denotes the expectation with respect to some risk-neutral measure. We assume that we are given a Markov process S describing the dynamics of underlying asset.

Remark 1.1. Disregarding convention, S will often contain components other than the price of the underlying asset. For instance, in the case of a Markovian stochastic volatility model such as the Heston model, we will choose

$$S_t = (t, X_t, v_t),$$

where t denotes time, X_t denotes the log-price of the underlying asset and v_t denotes the instantaneous variance.

Under mild conditions, the optimal stopping time τ^* of (1) will be the hitting time of a set, the *exercise region* of the option. Hence, if the Markov process S lives in $[0,T]\times\mathbb{R}^d$, then we may also write (1) as

$$v = \sup_{E\in\mathcal{B}([0,T]\times\mathbb{R}^d)} \mathbb{E}\left[Y_{\tau_E\wedge T}\right], \tag{2}$$

where

$$\tau_E \coloneqq \inf\{\, t\geq 0 \mid S_t \in E \,\}.$$

An obvious – and classical – approach of approximating v is to parameterize exercise regions E. If the expectation in (2) is, in turn, approximated by Monte Carlo simulation, this leads to highly irregular optimization problems.

We smooth the optimization problem by randomization. That is, we exercise the option at the first jump time of a pure-jump process with intensity $\lambda_t = f(S_t)$ depending

on the state of the underlying Markov process. For a given such intensity, we hence look at the randomized stopping time

$$\tau_f := \inf\left\{ t \geq 0 \;\middle|\; \int_0^t \lambda_s \mathrm{d}s \geq Z \right\}, \tag{3}$$

for some independent exponentially distributed random variable Z. After integrating Z out again, this leaves us with

$$v = \sup_f \mathbb{E}\left[\int_0^T Y_t U_t \lambda_t \mathrm{d}t + Y_T U_T \right], \quad U_t := \exp\left(- \int_0^t \lambda_s \mathrm{d}s \right), \tag{4}$$

where f runs over all non-negative measurable functions and $\lambda_t = f(S_t)$. Note that (4) is a smooth optimization problem (in f), and all derivatives of the right hand side w.r.t. f can be explicitly computed.

In our implementation, f is parameterized by exponentials of polynomials in S. More precisely, we use

$$f \in F_{\mathcal{P}} := \left\{ f : s \mapsto \mathbb{1}_{y>0}(s) \exp\left(p(s)\right) \;\middle|\; p \in \mathcal{P} \right\}, \tag{5}$$

where $y = y(s)$ denotes the (discounted) payoff and $\mathcal{P}$ denotes a finite-dimensional space of polynomials.

2 Examples

Computing prices in the ERO-code (see [B+])provided involves two steps:

1. Generate a `model`, which encodes both the actual model of the underlying asset in the sense of quantitative finance and the payoff of the option.
2. Call the `pryce` function, which solves the optimal stopping problem. It internally calls the function `pryce_ero` when using the ERO method.

Regarding the second step, the function `pryce` is called as follows,

```
pryce(model, k, M, method='ero', seed=None, **kwargs),
```

where

- `model` encodes the model and the option, see below for examples.
- `k` denotes the polynomial degree used for parameterizing the rate function. It takes values in $\{0, 1, \ldots\}$.
- `M` denotes the number of samples used for the training step. For the re-sampling stage we use $20M$ samples. `M` takes values in $\{2, 3, \ldots\}$.
- `method` is a string indicating the method to be used, where `'ero'` denotes the ERO method. Other methods implemented are `'ls'` for Longstaff–Schwartz and `'parallel_ero'` for a parallelized ERO method. We suggest to only use `'ero'` at this point.
- `seed` denotes the seed for the RNG used, which is `numpy`'s default RNG. It takes unsigned integer values, or no value at all.

- The remaining parameters do not need to be set.

`pryce` returns a tuple containing the following numbers:

1. The estimated, low-biased value of the option computed by re-sampling.
2. The standard deviation corresponding to the re-sampling procedure.
3. The estimated value of the option directly obtained from the optimization procedure, without re-sampling.
4. The European option price.

2.1 Possible options

Options are described by a payoff-class. While the code allows for considerable amount of flexibility, we suggest to restrict to the following cases:

- `payoff=PutPayoff(K, 1 / d * np.ones(d))`. This denotes a basket-put option, where all components of the basket have equal weights $1/d$. `K` > 0 denotes the strike price. Replacing "Put" by "Call" will give the corresponding basket-call option. If the dimension $d = 1$, then the weighting function does not need to be provided.
- `payoff=CallPayoff(K, weight_function=lambda x: np.max(x,axis=-1))` denotes a max-call option, with strike price `K` > 0.

2.2 Black Scholes model

Compute the American option price in a multi-variate Black-Scholes model. Basket and max-call-options are implemented. The Black-Scholes model is represented by a class `BlackScholes`, which can be plugged in `pryce` as its `model` argument. It is initialized as follows:

`BlackScholes(d, T, sigma, r, S0, N, payoff=None, dividend=0).`

We have the following parameters:

- `d` is the dimension of the model, i.e., the number of underlying assets. It takes values in $\{1, 2, \ldots\}$.
- `T` > 0 is the maturity.
- `sigma` denotes the covariance matrix of the underlying Brownian motion. In general, it is assumed to be a `numpy` array of dimension $d \times d$. If all assets are assumed independent, then `sigma` can also be specified as the vector of variances. Finally, if all assets are independent and have the same variance, `sigma` can be specified as a positive scalar.
- `r` ≥ 0 denotes the interest rate.
- `S0` denotes the vector of initial values of the underlying asset price process encoded by a `numpy` vector. It takes values in $\mathbb{R}^d_{>0}$.
- `N` denotes the time-discretization used for sampling the process and for computing the optimal stopping problem. It takes values in $\{1, 2, \ldots\}$.

- `payoff` is a class describing the option. See above for possible choices.
- `dividend` ≥ 0 denotes the continuous dividend rate

2.3 Heston model

Compute the American option price in a multi-variate Heston model. This means in our implementation that there are multiple assets which are driven by the same variance process. Basket and max-call-options are implemented. The Heston model is represented by a class `Heston`, which can be plugged in `pryce` as its `model` argument. It is initialized as follows:

```
Heston(d, T, nu0, theta, r, kappa, xi, rho, S0, N, payoff).
```

We have the following parameters:

- `d` is the dimension of the model, i.e., the number of underlying assets. It takes values in $\{1, 2, \ldots\}$.
- `T` > 0 is the maturity.
- `nu0` > 0 denotes the initial value of the variance process.
- `theta` > 0 denotes the long-term mean of the variance process.
- `r` ≥ 0 denotes the interest rate.
- `kappa` > 0 denotes the speed of mean reversion of the variance process.
- `xi` > 0 denotes the vol-of-vol parameter.
- `rho` denotes the correlation matrix between the d asset price processes and the 1 variance process. It is assumed to be a `numpy` array of dimension $(d+1)\times(d+1)$. If $d = 1$, then `rho` can also be specified as a scalar taking values in $[-1, 1]$. It is the user's responsibility to provide a proper correlation matrix.
- `S0` denotes the vector of initial values of the underlying asset price process encoded by a `numpy` vector. It takes values in $\mathbb{R}^d_{>0}$.
- `N` denotes the time-discretization used for sampling the process and for computing the optimal stopping problem. It takes values in $\{1, 2, \ldots\}$.
- `payoff` is a class describing the option. See above for possible choices.

Remark 2.1. At this point, the code does enforce the Feller condition, and will raise an error if it is violated.

2.4 Rough Bergomi model

Compute the American option price in the rough Bergomi model. Here the asset price process is one-dimensional (plus one further variance process), but the model lacks the Markov property. We use a Markovian approximation by enhancing the state of the process by parts of the history of the path. The rough Bergomi model is represented by a class `RoughBergomi`, which can be plugged in `pryce` as its `model` argument. It is initialized as follows:

`RoughBergomi(H, T, eta, xi, rho, S0, N, r, payoff, J, memory).`

It is initialized as follows:

- `H` denotes the Hurst index, and takes values in $]0, 1/2[$.
- `T` > 0 denotes the maturity.
- `eta` > 0 denotes the vol-of-vol parameter of the rough Bergomi model.
- `xi` > 0 denotes the initial value of the variance process.
- `rho` denotes the correlation between asset price process and variance process. It takes values in $[-1, 1]$.
- `S0` > 0 denotes the initial value of the underlying asset price process.
- `N` denotes the time-discretization used for sampling the process and for computing the optimal stopping problem. It takes values in $\{1, 2, \ldots\}$.
- `r` ≥ 0 denotes the interest rate.
- `payoff` is a class describing the option. See above for possible choices.
- `J` denotes the number of additional past values of the asset price and the variance processes to be included in the state of the process for the optimization procedure. It takes values $\{0, 1, \ldots\}$.
- `memory` denotes the time period, over which `J` additional past values are included in the state. E.g., if `J` $= 1$, and we are at time t, then the state for the optimization consists of
$$\left(\log S_t, \log S_{t-\texttt{memory}}, v_t, v_{t-\texttt{memory}}\right).$$
`memory` takes values in $[0, T]$.

References

[B+] Ch. Bayer, R. Tempone, S. Wolfers: *Pricing American Options by Exercise Rate Optimization*, arXiv preprint 1809.07300, 2018.

2