MTH 9821 Numerical Methods for Finance
Fall 2023
Homework 5

1 Backtest overfitting

1. [Writing] Show that the maximum function, $\max : \mathbb{R}^n \to \mathbb{R}$, is convex.

2. [Writing] An equity analyst at a hedgefund is developing a trading strategy on a stock S with price S_t at time t.

   He has a signal α_t as well, which we uses to determine how much to long/short in the next period. In other words, his holdings at time t are given by $f(\alpha_t, S_t)$, for some "trading strategy" f.

   The strategy's PnL is then given by

   $$\mathrm{PnL}(f) = \sum_{t=0}^{N} f(\alpha_t, S_t)(S_{t+1} - S_t)$$

   where $\{\alpha_t\}_{t=0\ldots N}$ and $\{S_t\}_{t=0\ldots N}$ are the historical signal and stock price series and we assume interest rates are zero.

   The analyst backtests $n > 1$ trading strategies $\{f_i : i = 1 \ldots n\}$, and picks best performing one f_*. He tells his boss that his strategy can make $\mathrm{PnL}(f_*)$.

   Show using part (1) that *on average*, this is an overestimate.

2 Polynomial regression

[Coding] In this exercise, you will implement polynomial regression. Fill in all the missing pieces of code in the provided `regression.cpp`.

1. [Reading] Read *Lecture 4. The Singular Value Decomposition* and *Lecture 11. Least Squares Problems* in Trefethen's Numerical Linear Algebra [TB97] for a concise overview of SVD and polynomial regression.

2. Calculate the Vandermonde matrix given by

$$\mathcal{V}(x) = \begin{pmatrix} 1 & x_1 & x_1^2 & \dots & x_1^{\text{deg}} \\ 1 & x_2 & x_2^2 & \dots & x_2^{\text{deg}} \\ 1 & x_3 & x_3^2 & \dots & x_3^{\text{deg}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^{\text{deg}} \end{pmatrix}$$

3. For the least squares fit, we will solve the so-called *normal equation*

$$X'X\beta = X'y \tag{1}$$

As discussed in the lecture, the matrices $X = \mathcal{V}(x)$ have a tendency to be ill-conditioned[1].

To improve numerical stability, we *precondition* the matrix X by dividing each column by its ℓ^2 norm:

$$\hat{X} = XD^{-1}$$

where

$$D_{ij} = \delta_{ij}\sqrt{\sum_{k=1}^{n} x_{ki}^2}$$

Note: you should not be calculating D as a dense matrix in Eigen.

4. Use Eigen's BDCSVD solver to calculate the singular value decomposition of $\hat{X}$. To save time, be sure to pass the flags `ComputeThinU | ComputeThinV` to the solver.

$$\hat{X} = U\Sigma V'$$

The coefficients and fitted values are found by

$$\beta = D^{-1}V\Sigma^{-1}U'y \tag{2}$$
$$\hat{y} = UU'y \tag{3}$$

Note: the calculation of $U'y$ can be shared between β and $\hat{y}$. Don't calculate $\hat{y}$ as $X\beta$ as this may lead to unnecessary loss of precision.

[Writing] Prove that β as given in equation (2) satisfies equation (1) and that $\hat{y}$ as given in equation (3) equals $X\beta$. Do not assume in your proof that either U or V are square matrices or that X has full rank.

Explain (with math and formulas) the meaning of the `ComputeThinU`, `ComputeThinV` flags.

[1] if the values x aren't uniformly distributed on the unit circle in $\mathbb{C}$

5. Calculate the pseudo Vandermonde matrix for Hermite polynomials given by

$$\mathcal{H}(x) = \begin{pmatrix} 1 & x_1 & x_1^2 - 1 & \dots & \mathrm{He}_{\mathrm{deg}}(x_1) \\ 1 & x_2 & x_2^2 - 1 & \dots & \mathrm{He}_{\mathrm{deg}}(x_2) \\ 1 & x_3 & x_3^2 - 1 & \dots & \mathrm{He}_{\mathrm{deg}}(x_3) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 - 1 & \dots & \mathrm{He}_{\mathrm{deg}}(x_n) \end{pmatrix}$$

   If x roughly follows a distribution $\mathcal{N}(\mu, \sigma^2)$, the columns of $\mathcal{H}(x)$ can be made roughly orthogonal by standardizing:

$$\hat{x}_i = \frac{x_i - \mathrm{mean}(x)}{\mathrm{std}(x)}$$

   To ensure consistent results, don't use Bessel's correction when implementing this.

6. [Reading] Singular value decomposition is a slow, but highly accurate way of solving least squares problems, particularly ones where $\mathrm{rank}(X)$ is less than the number of columns. In our application, the matrices $\mathcal{V}(x)$ and $\mathcal{H}(x)$ won't be close to rank-deficient, so instead of SVD, QR decomposition could be used for increased performance. The efficient usage of Eigen's `HouseholderQR` class is, however, beyond the scope of this assignment.

3 Regression Monte Carlo

[Coding] In this exercise, you will implement the Tsitsiklis–Van Roy [TV01] and the Longstaff–Schwartz [LS01] algorithms to price American options. Further details and discussion of these algorithms can be found in [CLP02] and [Gla13].

3.1 Continuation value

Suppose we're pricing a Bermudan with a discrete set of exercise times $\{0 = t_0, t_1, \dots t_M = T\}$. In order to decide at any given point t_i whether or not to exercise, the immediate exercise value must be compared to the estimated continuation value $C_i(S)$.

We're going to be using polynomial regression methods implemented in Problem 2, so $C_i(S)$ will be of the form:

$$C_i(S) = \beta_{i0} + \beta_{i1} S + \dots + \beta_{id} S^d \tag{4}$$

To store the coefficients β in an orderly manner, you have been provided with the class `PolynomialMCRegression` in `mc_regression.cpp`.

Implement the missing methods in the similar class `HermiteMCRegression`.

3.2 Backward pricers

We describe how to implement `regression_pricer_backward` as declared in `american_pricers.h`.

```
double regression_pricer_backward(const double spot,
    const arr2& paths,
    const int w, const double strike, const double maturity,
    const double interest_rate, MCRegression& mc_regression,
    const MonteCarloRegressionMethod& method);
```

Let's start by taking a look at some of the arguments.

`method`: Assume that this is set to `Tsitsiklis_VanRoy` for now, we will discuss the `Longstaff_Schwartz` case later.

`w`: The payoff of the option will be $\max(w(S-K),0)$, so $w=1$ means call, and $w=-1$ means put.

`mc_regression`: An instance of `PolynomialMCRegression` or `HermiteMCRegression`. This object will estimate the continuation values $C_i(S)$ and store the coefficients β_{ij} seen in equation (4).

`paths`: An array with N rows and M columns. Each of the N rows of `paths` represents a randomly generated path. More formally, its entries are:

$$\begin{aligned} \text{paths}_{k,j-1} &= S_{j\delta t,k} \\ &= S(0)\exp\left\{(r-q-\sigma^2/2)j\delta t + \sigma\sqrt{\delta t}\sum_{i=1}^{j}\text{noise}_{Mk+i}\right\} \end{aligned} \tag{5}$$

for $j = 1\ldots M$ and $k = 0\ldots N-1$, and $\delta t = T/M$. The noise array above will contain an array with NM i.i.d. standard normal samples.

It is recommended to use a "cumulative sum" function to implement this.

In the body of `regression_pricer_backward`, we will need the following arrays, each of length N:

- E will be the immediate **E**xercise value
- C will be the estimated **C**ontinuation value, computed at each step by regression
- P will be the actual **P**ayoff value

The Tsitsiklis–Van Roy algorithm can be summarised by three equations:

$$E_t = \max(w(S_t-K),0) \tag{6}$$

$$C_t = \text{proj}(P_{t+\delta t}e^{-r\delta t}|\text{span}(f_1(S_t), f_2(S_t), \ldots f_B(S_t))) \tag{7}$$

$$P_t = \begin{cases} E_t & \text{if } t = T \\ \max(E_t, C_t) & \text{if } 0 < t < T \end{cases} \tag{8}$$

The proj operator in equation 7 means "calculate the fitted values of a linear regression". To compute this, please use the `fit_predict` method of the `mc_regression` argument.

When $t = 0$, the stock price is known: $S(0)$. In this case, the orthogonal projection makes C_0 the average of $P_{\delta t}e^{-r\delta t}$ across paths. In this case the computation should be done in the `fit_predict_at_0` method.

The algorithm returns a single value, $P_0 := \max(E_0, C_0)$.

Implement `regression_pricer_backward` with a `for` loop, using the recursive formulation above.

With a small code change, your function can also compute the Longstaff–Schwartz value. The only difference is equation (8) which should be modified to:

$$P_t = \begin{cases} E_t & \text{if } t = T \\ E_t & \text{if } E_t \geq C_t \text{ and } 0 < t < T \\ P_{t+\delta t}e^{-r\delta t} & \text{if } E_t < C_t \text{ and } 0 < t < T \end{cases}$$

Use the `regression_pricer_backward` function's `method` argument to enable the caller to switch between the two algorithms.

3.3 Forward pricer

When running `regression_pricer_backward`, it not only prices the option, but also modifies the supplied `MCRegression` object[2]. After pricing, the regression object contains all the fitted coefficients, so all the information needed to calculate the continuation values $C_i(S)$.

This allows us to easily implement the "forward version" of our pricer:

```
double regression_pricer_forward(const double spot,
    const arr2& paths,
    const int w, const double strike, const double maturity,
    const double interest_rate,
    const MCRegression& mc_regression);
```

As before we will use an array P of length N for the actual payoff values.

For each path k, this method starts at time $t_0 = 0$, and step by step compares the immediate exercise value $K - S$ to the estimated continuation value $C_i(S)$. If at time $j\delta t$ we find that $K - S_{j\delta t,k}$ exceeds $C_j(S_{j\delta t,k})$, the option is *exercised*, we set $P_k = e^{-j\delta t}(K - S_{j\delta t,k})$, break the forward loop and move on to the next path, starting at time 0.

Return the average payoff value across paths: $\mathrm{mean}(P)$.

Your implentation should use both `MCRegression::predict` and `MCRegression::predict_at_0`.

3.4 Calculations

In the attached spreadsheet, you will be asked to price an option using the methods `regression_pricer_backward` and `regression_pricer_forward` with various parameters. Consider a six-month monthly Bermudan put with spot 40.5, strike 44, volatility 20%. Assume that the risk free rate is 4%, dividend rate is 0%.

We will run $N_e = 100$ experiments, each with 10,000 paths.

[2] As could be seen from the lack of `const` in front of that argument

For the first experiment, generate 10,000 paths with 6 timesteps each using the LCG + inverse transform method with seed $x_0 = 100$. Refer to equation (5) for details.

For each following experiment, generate another 10,000 paths using the last seed from the previous experiment. If you started with $x_0 = 100$, the second seed should be

$$x'_0 := x_{60,000} = 1185163621$$

Record the prices $V_i = 1 \dots N_e$ for each experiment and report[3]

$$\hat{V} = \frac{1}{N_e} \sum_{i=1}^{N_e} V_i$$

$$\text{s.e.}(\hat{V}) = \frac{1}{\sqrt{N_e}} \sqrt{\frac{1}{N_e} \sum_{i=1}^{N_e} \left(V_i - \hat{V}\right)^2}$$

For the "out-of-sample" paths start the first experiment at $x_0 = 200$ instead.

Compare your results with those of the binomial tree method with 6,000 timesteps: 4.079018801027898. Report only the differences `regression_price - binomial_tree_price`.

As a sanity check, make sure that for any given Longstaff-Schwartz setup, the in-sample backward and forward pricers produce the same output. Understand why this should be the case and why it's not true for Tsitsiklis–Van Roy.

3.5 Discussion

1. [Writing] Having filled in the spreadsheet, answer the following question about your results:

   (a) Which one tends to be more accurate: in-sample or out-sample pricing? Is there a bias one way or the other between the two? Give an intuitive explanation for the bias.

   (b) Do out-sample results have a bias one way or the other (compared to the true value)? Explain where this bias comes from and why it is less noticeable in-sample.

   (c) Which of Tsitsiklis–Van Roy and Longstaff–Schwartz produce more accurate results in-sample? Is there a bias one way or the other between the two? Find an intuitive explanation for the bias.

   (d) Does a polynomial regression with the monomial basis benefit from preconditioning? Explain your answer by looking at the difference between backward and forward prices in-sample.

   (e) Does a polynomial regression with the Hermite basis benefit from preconditioning? Does it benefit from standardization? Or both?

[3] The standard errors help us understand how much variation to expect from running the pricer with different seeds. They do not feature in pricing.

(f) Explain why the pricers typically produce higher values as you increase the degree of the polynomial.

(g) Which configuration would you use (basis, degree, preconditioning, TVR/LS, forward/backward, in-sample, out-of-sample) in production and why?

2. [Writing] Read the Section 1 and the beginning of Section 2 of Longstaff and Schwartz's original paper [LS01].

   They use a non-polynomial basis for their regressions. Is this basis orthogonal under any probability distribution? If so, name the distribution. Does their simulation data come from such a distribution?

   Would their choice of basis would be appropriate for a pricing call option?

3. [Writing] Let's assume you've generated paths and priced an American option.

   How would you calculate a confidence interval that contains the option's true value with 95% probability. (Without repeatedly running the pricer)

   How would you calculate the option's delta?

References

[CLP02] Emmanuelle Clément, Damien Lamberton, and Philip Protter. *An analysis of a least squares regression method for American option pricing.* Berlin, 2002. URL: `https://hdl.handle.net/1813/9176`.

[Gla13] Paul Glasserman. *Monte Carlo methods in financial engineering.* Vol. 53. Springer Science & Business Media, 2013.

[LS01] Francis A Longstaff and Eduardo S Schwartz. "Valuing American options by simulation: a simple least-squares approach". In: *The review of financial studies* 14.1 (2001), pp. 113–147.

[TB97] Lloyd N. Trefethen and David Bau. *Numerical linear algebra.* SIAM, 1997. ISBN: 978-0-89871-361-9. DOI: `10.1137/1.9780898719574`.

[TV01] J. N. Tsitsiklis and B. Van Roy. "Regression methods for pricing complex American-style options." In: *IEEE transactions on neural networks* 12 (4 2001), pp. 694–703. ISSN: 1045-9227. DOI: `10.1109/72.935083`. ppublish.