POSITION-BUILDING IN COMPETITION WITH REAL-WORLD CONSTRAINTS

Neil A. Chriss[1]

[1]neil.chriss@gmail.com

September 13, 2024

***K*eywords** Trading, position-building, game theory, Euler-Lagrange, trading strategies, equilibrium, strategy selection under uncertainty

1 Introduction

In [Chriss(2024)] this author introduce a framework for calculating an optimal trading strategy that builds position in a stock in the face of competition. When a trader wants to buy a target quantity of stock (scaled to one unit) over a period of time $t = 0$ to time $t = 1$, during which one or more adversaries are also buying the same stock, the trader contends with the aggregate market impact of their own trading and that of their adversaries. Traders face two distinct kinds of market impact, namely, temporary impact, the transitory price pressure due to a premium charged for immediacy, and permanent impact, the effect of a information leakage and residual impact from persistent buying in the stock. The focused analyzed the game-theoretic aspect of trader choosing a strategy as a best response to another trader's strategy, where the object is to minimize the total cost of trading due to market impact arising from the aggregate trading of all traders at each moment in time[1].

In [Chriss(2024)] many scenarios are analyzed, but the key features of the framework may be understood when there are two traders, A and B with B trading a strategy represented by a twice-differentiable function $b(t)$ mapping the unit interval $[), 1]$ to the real number $\mathbf{R}$ such that $b(0) = 0$ and $b(1) = 1$, representing the quantity of stock held by B at each point in time scaled to a final quantity of 1, which B is actually trading λ units of stock so that the actual quantity traded is $\lambda b(t)$. The most important scenario we analyze is where for a *known* strategy $b(t)$, A wishes to find the strategy $a(t)$, also satisfying the boundary conditions $a(0) = 0, a(1) = 1$, that is the best-response to $b(t)$ in the sense that it minimizes the total cost of trading described by the loss function $L(t) = (\dot{a} + \dot{b})\dot{a} + \kappa(a + \lambda)b\dot{a}$ representing the sum of temporary and permanent impact:

$$\underset{a:[0,1]\to\mathbf{R}}{\textbf{minimize}} \int_0^1 (\dot{a} + \lambda\dot{b})\dot{a} + \kappa(a + \lambda b)\dot{a}\,\mathtt{dt} \tag{1}$$

We call Eq. (1) the *continuous, unconstrained* best-response optimization problem. This is a *variational problem* that seeks to find a path minimizing the cost function $\text{Cost}(a, b)$ representing the total cost of trading for A's strategy a while in competition with the trader B trading the strategy b, scaled to λ units of stock.

$$\text{Cost}(a; b) = \int_0^1 (\dot{a} + \lambda\dot{b})\dot{a} + \kappa(a + \lambda b)\dot{a}\,\mathtt{dt} \tag{2a}$$

$$= \int_0^1 L(t)\,\mathtt{dt} \tag{2b}$$

[1]Draft: Sept 13, 2024, 12:32pm.

1.1 Position-building in the real world

The premise behind optimal-position building with competition is that a trader or portfolio manager has identified a catalyst in a stock that will occur on a future date that is expected to cause a revaluation in the stock from the current price p_0 to a new, higher price, p_1. The trader wishes to buy the stock ahead of the catalyst to profit from the revaluation and the trade is said to have an *embedded profit* of $p_1 - p_0$ and the price p_0 is called the *arrival price* of the stock[2], and is usually taken to be the price directly prior to the commencement of trading. Given that A is targeting the acquisition of one unit of stock, the *actual* profit of the trade will be reduced by the cost of trading.

The paper [Chriss(2024)] uses techniques from the calculus of variations which impose limitations on imposing various necessary constraints on the problem, such as:

- **Limitations on-overbuying**: in optimal position-building when there is significant permanent market impact there can be a considerable amount of *over-buying* of the name and providing liquidity to the adversary. Suppose, however, that A does not unlimited over-buying due to some *real-world constraints*, e.g., capital or risk limits) and wants to limits purchases to less than ρ-percent of the target quantity. Then this would mean a *trajectory constraint* of $a(t) \leq 1 + \rho$;
- **Channel constraints:**
- **No selling:** suppose that A never wants to sell the stock, this translates to a constraint on the first-derivative, namely, $\dot{a}(t) \geq 0$ for all t.

To solve such problems requires solving a *constrained* version of Eq. (1), along the lines of:

$$\begin{aligned} \underset{a:[0,1]\to\mathbf{R}}{\textbf{minimize}} \quad & \int_0^1 (\dot{a} + \lambda\dot{b})\dot{a} + \kappa(a + \lambda b)\dot{a}\,\mathtt{dt} \\ \text{subject to} \quad & L(t) \leq \dot{a}(t) \leq U(t) \\ & a(t) \leq 1 + \rho \end{aligned} \tag{3}$$

where $L(t)$ and $U(t)$ define lower- and upper-bounds for the stategy $a(t)$ at each time t.

2 Notation

In this section we set forth the notation and terminology to be used through the rest of the paper. Write $\mathcal{C}2$ for the space of twice-differentiable functions from the unit interval to the real-numbers:

$$\mathcal{C}^2[0,1] = \{x : [0,1] \to \mathbf{R} \,|\, x \text{ is twice-differentiable}\} \tag{4}$$

Trading strategies function $a \in \mathcal{C}^2[0,1]$ satisfying the boundary conditions $a(0) = 0$ and $a(1) = 1$. We call the value of $a(1)$ the *target quantity*. When a strategy has a target quantity other than one, denoted by $\lambda > 0$, we call the strategy λ-scaled use a normalized strategy, say $b(t)$, and note that we will always represent the strategy as $b \in \mathcal{C}^2[0,1]$ with $b(0) = 0, b(1) = 1$ and note that the actual quantity held at a given time t is $\lambda b(t)$ so that its target quantity is λ. We typically write $b(t)$ for a λ-scaled strategy and call its graph the *shape* of the strategy. We write $\dot{a}$ for the time derivative of a and note that this represnts the *rate* of trading in the stock by a.

We write $\mathbf{N}$ for the natural numbers, $\mathbf{N}^p$ for the positive integers and $\mathbf{N}^+_{\text{odd}}$, $\mathbf{N}^+_{\text{even}}$ for the odd and respectively even positive, integers.

3 Approximating the total cost of trading using Fourier series

In this section we show that the total cost of trading a strategy $a(t)$ in competition with a second λ-scaled strategy $b(t)$ may be approximated by replacing the functions $a(t)$ and $b(t)$ by (effectively) Fourier series and that with the series we may estimate the cost function as a quadratic function of the coefficients of the Fourier series.

[2]This terminology is taken from the market impact and optimal execution literature.

3.1 Brief review of Fourier series of trading strategies

Let $a \in C^2[0,1]$ with such that $a(0) = 0, a(1) = 1$ be a trading strategy. We want to approximate a by a trigonometric series in N terms plus a linear term as follows:

$$a_{\mathcal{F},N}(t) = t + \sum_{1}^{N} a_n \sin(n\pi t) \tag{5}$$

Here the subscript $\mathcal{F}$ signifies that this is related to a Fourier series (in particular a sine series) and the N indicates there are N terms in the sum. We note that $\sum_1^N a_n \sin(n\pi t)$ is the N-th partial sum of of the Fourier series of $a(t) - t$ on $[0,1]$. We gather a few definitions and a bit of terminology. We will typically drop the subscript N in Eq. (5) and write:

$$a_{\mathcal{F}}(t) = t + \sum_{1}^{N} a_n \sin(n\pi t) \tag{6}$$

We call Eq. (6) the N-term Fourier series of the trading strategy a and the $a_n, n = 1, \ldots, N$ the *Fourier coefficients* of the strategy. From this we immediately know the following facts about $a_{\mathcal{F}}$.

Proposition 3.1 (Fourier expansion of trading strategies)**.** Given $a_{\mathcal{F}}(t)$ as above:

- $a_{\mathcal{F}}(0) = 0, a_{\mathcal{F}}(1) = 1, (a_{\mathcal{F}} - t)(0) = 0, (a_{\mathcal{F}} - t)(1) = 0,$;
- $\sum_1^N a_n \sin(n\pi t)$ converges uniformly to $a(t) - t$ as $N \to \infty$;
- $a_{\mathcal{F}}(t) \to a(t)$ uniformly as $N \to \infty$; and
- The coefficient a_n may be calculated as the n-th Fourier coefficient of $a(t) - t$:

$$a_n = 2 \int_0^1 (a(t) - t) \sin(n\pi t) \, \mathtt{dt} \tag{7}$$

- the coefficients a_n decay *at least* like $1/n^2$, more specifically $|a_n| \leq C/n^2$.

Fourier expansions of trading strategies

$$a_{\mathcal{F}}(t) = t + \sum_{n=1}^{N} a_n \, \sin(n\pi t)$$
$$\dot{a}_{\mathcal{F}}(t) = 1 + \sum_{n=1}^{N} a_n \, n\pi \cos(n\pi t)$$
$$b_{\mathcal{F}}(t) = t + \sum_{n=1}^{N} b_n \, \sin(n\pi t)$$
$$\dot{b}_{\mathcal{F}}(t) = 1 + \sum_{n=1}^{N} b_n \, n\pi \cos(n\pi t)$$

Figure 1: *The key expansions to be used in this paper.*

Finally in Box 3 we provide the key trigonometric identities we will use in this paper when dealing with Fourier series of trading strategies.

Key trigonometric identities

$$\int_0^1 \sin(n\pi t)\,\mathtt{dt} = \begin{cases} \frac{2}{n\pi} & \text{if } n \in \mathbf{N}^+_{\text{odd}} \\ 0 & \text{otherwise} \end{cases}$$

$$\int_0^1 \cos(n\pi t)\,\mathtt{dt} = \begin{cases} 1 & \text{if } n = 0 \\ 0 & \text{if } n \neq 0 \end{cases}$$

$$\int_0^1 t\cos(n\pi t)\,\mathtt{dt} = \begin{cases} -\frac{2}{n^2\pi^2} & \text{if } n \in \mathbf{N}^+_{\text{odd}} \\ 0 & \text{otherwise} \end{cases}$$

$$\int_0^1 \cos(n\pi t)\cos(m\pi t)\,\mathtt{dt} = \begin{cases} \frac{1}{2} & \text{if } n = m \\ 0 & \text{if } n \neq m \end{cases}$$

$$\int_0^1 \cos^2(n\pi t)\,\mathtt{dt} = \frac{1}{2}$$

$$\int_0^1 \cos(n\pi t)\sin(m\pi t)\,\mathtt{dt} = 0 \quad \text{for all } n \text{ and } m$$

Figure 2: *The key trigonometric identities we will use in this paper.*

4 Approximating the cost function with Fourier series

Our strategy is to approximate the cost function $\text{Cost}(a;b) = \int_0^1 (\dot{a} + \lambda\dot{b})a + \kappa(a + \lambda b)\dot{a}\,\mathtt{dt}$ using the key expansions in Box 1, that is, replace $\text{Cost}(a;b)$ with $\text{Cost}_{\mathcal{F}}(a;b)$ defined by:

The Fourier approximation of the cost function

$$\text{Cost}_{\mathcal{F}}(a;b) =$$

$$(1+\lambda) + \frac{\pi^2}{2}\sum_{n=1}^{N}\left(a_n^2 + \lambda b_n a_n\right)n^2 + \kappa\left\{\frac{1+\lambda}{2} - \sum_{n\in\mathbf{N}^+_{\text{odd}}}^{N}\frac{2(1+\lambda)a_n}{\pi^2 n^2} + \lambda\sum_{n\in\mathbf{N}^+_{\text{odd}}}^{N}\frac{2b_n}{n\pi}\right\}$$

Figure 3: *The Fourier approximation to the cost function Cost$(a;b)$ takes the Fourier coefficients of b given by $b_{\mathcal{F}}(t) = t + \sum_1^N b_n \sin(n\pi t)$ as a given and provides an estimate of Cost$(a;b)$ in terms of the Fourier coefficients of a in $a_{\mathcal{F}} = t + a_n\sum_1^N \sin(n\pi t)$. The resulting fuction Cost$_{\mathcal{F}}(a;b)$ is a quadratic function in $a_1 \cdots a_N$.*

We provide the derivation of this formula in Section 8 and note here:

- $\text{Cost}_{\mathcal{F}}(a;b)$ is a *quadratic function* of the Fourier coefficients $a_1 \cdots a_n$; and
- $\text{Cost}_{\mathcal{F}}(a;b) \to \text{Cost}(a,b) = \int_0^1 (\dot{a} + \lambda\dot{b})a + \kappa(a + \lambda b)\dot{b}\,\mathtt{dt}$ and $\text{Cost}_{\mathcal{F}}(a;b) \to \text{Cost}(a,b)$ as the number of Fourier coefficients goes to infinity, namely as $N \to \infty$

5 Using the approximate cost function for optimal trading

In this section we use the cost function equation to build robust-best response strategies to adversary's trading. The setup is as follows. Let $a(t)$ be a trading strategy and $b(t)$ a λ-scaled trading strategy. We want to find the optimal best-response strategy to $b(t)$ (see [Chriss(2024)]) but want to use the cost function approximation to re-cast optimal

trading strategies in a non-linear programming framework. For starters note that the cost function Eq. (71) is a *functional* mapping the space of twice-differentiable functions:

$$\text{Cost} : \mathcal{C}^2[0,1] \times \mathcal{C}^2[0,1] \to \mathbf{R} \tag{8}$$

When b is known and fixed we can re-write Cost as:

$$\text{Cost}_b : \mathcal{C}^2[0,1] \to \mathbf{R} \tag{9}$$

and think of finding best-response strategies as an optimization problem taking place in $\mathcal{C}^2$, the province of calculus of variations which was the main tool in [Chriss(2024)]. The main message of this section is that to have a practical method for adding real-world constraints to we have to re-cast the problem from that of a *variational* problem involving functions and functionals to an optimization problem over the real numbers so that we may readily add constraints.

Recall that the approximate cost function $\text{Cost}_{\mathcal{F}}(a;b)$ is a quadratic function in the Fourier coefficients of a given by $a_1 \dots a_N$, therefore the following is a *unconstrained quadratic program*:

$$\underset{a_1 \cdots a_N}{\text{minimize}} \quad \text{Cost}_{\mathcal{F}}(a;b) \tag{10}$$

Note that the `arg-min` of Eq. (10) is a set of Fourier coefficients $a_1 \cdots a_N$, which then determines $a_{\mathcal{F}}(a_1 \cdots a_N)$. Note that this Eq. (10) acts as a stand-in for the functional minimization Eq. (1), and thus the `arg-min` of Eq. (10) behaves as an approximation to the `arg-min` of Eq. (1). Put differently, solving Eq. (1) yields $a_1 \cdots a_N$ and

$$(a_1 \cdots a_N) \mapsto a_{\mathcal{F}}(t) = t + \sum_1^N a_n \sin(n\pi t) \tag{11}$$

With this setup we can begin to add constraints to optimal position-building strategies in competition.

5.1 The no overbuying constraint

As a first example of adding constraints to the optimization program we start with the overbuying constraint. To review we want to find the optimal best-response strategy $a(t)$ (where $a(0) = 0$ and $a(1) = 1$ trading in competition with the λ-scaled strategy $b(t)$. We want to insure that for some $\rho > 1$, $a(t) \leq 1 + \rho$ for all $t \in [0,1]$. We call ρ the *overbuying threshold*. We depict the over-buying constraint in Figure 4.

Figure 4: *An example of a over-buying strategy (blue line) exceeds the target quantity of one and therefore violates the over-buying constraint.*

.

We accomplish this by adding constraints to the unconstrained minimization problem Eq. (10) and note that each N-tuple of Fourier coefficients $a_1 \dots a_N$ corresponds to the Fourier expansion in N terms $a_{\mathcal{F}} : [0,1] \to \mathbf{R}$, and $a_{\mathcal{F}} \approx a$, that is the Fourier expansion is approximately equal to the strategy a:

$$(a_1 \dots a_N) \mapsto a_{\mathcal{F}}(t) = t + \sum a_n \sin(n\pi t) \tag{12}$$

Given thus, to add the over-buying constraint for $\rho > 0$ to the quadratic program Eq. (10) we simply :

$$\begin{aligned} \underset{a_1 \dots a_N}{\text{minimize}} \quad & \text{Cost}_{\mathcal{F}}(a; b) \\ \text{subject to} \quad & t + \sum_1^N a_n \sin(n\pi t) \leq 1 + \rho, \text{ for all } t \in [0,1] \end{aligned} \tag{13}$$

Note that Eq. (13) is *not* implementable *as such* because the constraint involves at $t \in [0,1]$. However, if we replace the constraint for all t with K times $0 < t_1 \dots t_K < 1$ we obtain:

$$\begin{aligned} \underset{a_1 \cdots a_N}{\text{minimize}} \quad & \text{Cost}_{\mathcal{F}}(a; b) \\ \text{subject to} \quad & t_1 + \sum_1^N a_n \sin(n\pi t_1) \leq \ 1 + \rho \\ & \qquad \vdots \\ & t_K + \sum_1^N a_n \sin(n\pi t_K) \leq 1 + \rho \end{aligned} \tag{14}$$

and we see that the program Eq. (14) is a quadratic program in N variables with K linear constraints. We note that given the low level of oscillations in the Fourier expansions the function $a_{\mathcal{F}}(t_1) \dots a_{\mathcal{F}}(t_K)$ will come very close to actual mean of a for moderate levels of the number of Fourier coefficients N and number of grid points K.

5.2 Trajectory channel constraints

A trajectory channel constraint seeks to keep the trading trajectory within some bounds. For example, suppose a traders wants the optimal best-best response trading in competition with $b(t)$ but wants to be no more risk averse than the optimal risk-averse channel building strategy $\sinh(4t)/\sinh(4)$ and no more risky than the risk-neutral strategy (see Figure 5 for a plot of this).

Using the quadratic programming methodology to solve this problem requires specifying two *channel bounds*:

$$L : [0,1] \to \mathbf{R}, \quad L(0) = 0, L(1) = 1 \tag{15a}$$
$$U : [0,1] \to \mathbf{R}, \quad U(0) = 0, U(1) = 1 \tag{15b}$$
$$L(t) < U(t) \text{ for all } t \in (0,1) \tag{15c}$$

and then solving the program for grid points $t_1 \cdots t_K$:

$$\begin{aligned} \underset{a_1 \cdots a_N}{\text{minimize}} \quad & \text{Cost}_{\mathcal{F}}(a; b) \\ \text{subject to} \quad & t + \sum_1^N a_n \sin(n\pi t) \geq L(t), \qquad t = t_1 \cdots t_K \\ & t + \sum_1^N a_n \sin(n\pi t) \leq U(t), \qquad t = t_1 \cdots t_K \end{aligned} \tag{16}$$

This will limit strategies those with non-negative first derivatives, namely those that do not sell at any point in time.

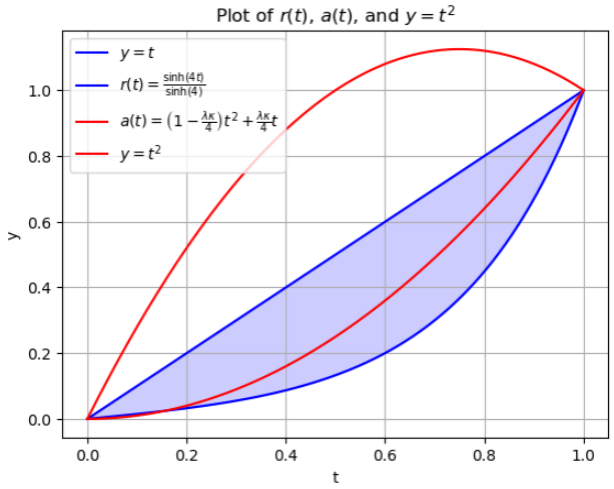

Figure 5: *The trajectory channel bounded by* $\sinh(4t)/\sinh(4)$ *below and* t *above. We may seek a constrained optimization that restricts to strategies that are within this channel and see that the risk averse strategy* $y = t^2$ *is always within the channel while the eager strategy is never within the channel.*

6 Near-completion constraints

Suppose a trader wants to implement a constraint something like, the strategy will have acquired and continue to hold at least 95% of the target quantity by time time τ. We depict this constraint in Figure 6.

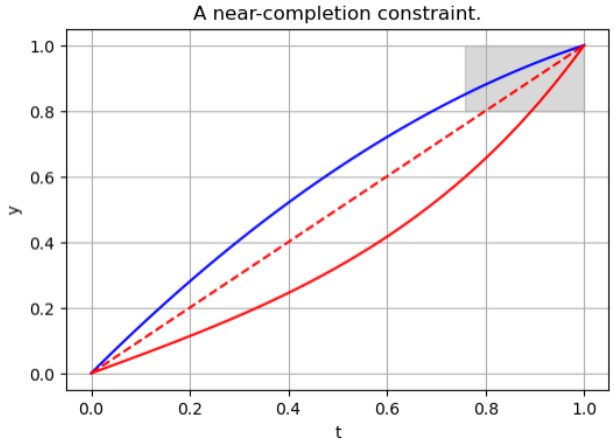

Figure 6: *A visual depiction of near-completion constraints. The constraint requires that at time* $t = 0.75$ *the trading strategy is at least 80% completed and does not exceed 100% of the target quantity. In this plot, the eager strategy (blue line) satisfies this constraint while the risk-averse strategy (read line) does not.*

We can view this as a *partial channel constraint* and solve this as follows.

Let $0 < \tau, c < 1$ and then choose time points

$$\tau = t_1 < \cdots < t_K < 1 \tag{17}$$

and then we solve

$$\begin{aligned} \underset{a_1 \cdots a_N}{\text{minimize}} \quad & \text{Cost}_{\mathcal{F}}(a;b) \\ \text{subject to} \quad & t + \sum_{1}^{N} a_n \sin(n\pi t) \geq c, \qquad t = t_1 \cdots t_K \end{aligned} \tag{18}$$

If we want to insure that the strategy remains between c and 1 from $t \in [0, \tau]$ we simply add a single new constraint:

$$\begin{aligned} \underset{a_1 \cdots a_N}{\text{minimize}} \quad & \text{Cost}_{\mathcal{F}}(a;b) \\ \text{subject to} \quad & t + \sum_{1}^{N} a_n \sin(n\pi t) \geq c, \qquad t = t_1 \cdots t_K \\ & t + \sum_{1}^{N} a_n \sin(n\pi t) \leq 1, \qquad t = t_1 \cdots t_K \end{aligned} \tag{19}$$

6.1 Implementing a no-sell constraint

In [Chriss(2024)] it was noted that a strategy shape that arises in the study of position building with competition is the *bucket strategy* depicted in Figure 4. Suppose that the trader wishes to constrain solutions to optimal best-response strategies to ones that do not trade bucket strategies.

There are two approaches to this. The first is simply to constrain the first derivative of the best response strategy to be non-negative, that is, to insure that $\dot{a}(t) \geq 0$ for all $t \in [0, 1]$. We can achieve this using the Fourier approximation of $\dot{a}$:

$$\dot{a}_{\mathcal{F}}(t) = 1 + \sum_{n=1}^{N} a_n \, n\pi \cos(n\pi t) \tag{20}$$

and for a set of times $t_1 \cdots t_K$ require $\dot{a}_{\mathcal{F}}(t_k) \geq 0$ for each k. This can then be solved using

$$\begin{aligned} \underset{a_1 \cdots a_N}{\text{minimize}} \quad & \text{Cost}_{\mathcal{F}}(a;b) \\ \text{subject to} \quad & t_1 + \sum_{n=1}^{N} a_n \, n\pi \cos(n\pi t_1) \geq \ 0 \\ & \qquad \vdots \\ & t_K + \sum_{n=1}^{N} a_n \, n\pi \cos(n\pi t_K) \geq 0 \end{aligned} \tag{21}$$

Note: the no-sell constraint by definition constrains the strategy from selling at *any point* in time along the trajectory, therefore, it will automatically prevent the overbuying constraint as in Section 5.1. However, the no-selling constraint is *stronger* than the no-overbuy constraint because it prohibits selling even when the strategy is holding less than the target quantity.By contrast, the no overbuying constraint does not directly address selling at y all, rather it prevents selling only because of the boundary condition $a(1) = 1$. If a strategy holds more than the target quantity of 1 at any time, then selling is required to ultimately satisfy the boundary condition.

7 Examples of Fourier series approximations of various best-response and equilibrium strategies

In [Chriss(2024)] we derived various strategies and in this section we demonstrate what their Fourier expansions look like.,

7.1 Best-response to a risk-neutral adversary

$$a(t) = \left(1 - \frac{\lambda\kappa}{4}\right) t^2 + \frac{\lambda\kappa}{4} t \tag{22}$$

Here are the Fourier coefficients:

$$a_1 = \frac{2(-4 + \lambda\kappa)}{\pi^3}, \tag{23}$$

$$a_2 = 0, \tag{24}$$

$$a_3 = \frac{2(-4 + \lambda\kappa)}{27\pi^3}, \tag{25}$$

$$a_4 = 0. \tag{26}$$

The Fourier series approximation for $a(t)$ using the first four terms is:

$$a(t) \approx t + \frac{2(-4 + \lambda\kappa)}{\pi^3} \sin(\pi t) + \frac{2(-4 + \lambda\kappa)}{27\pi^3} \sin(3\pi t) \tag{27}$$

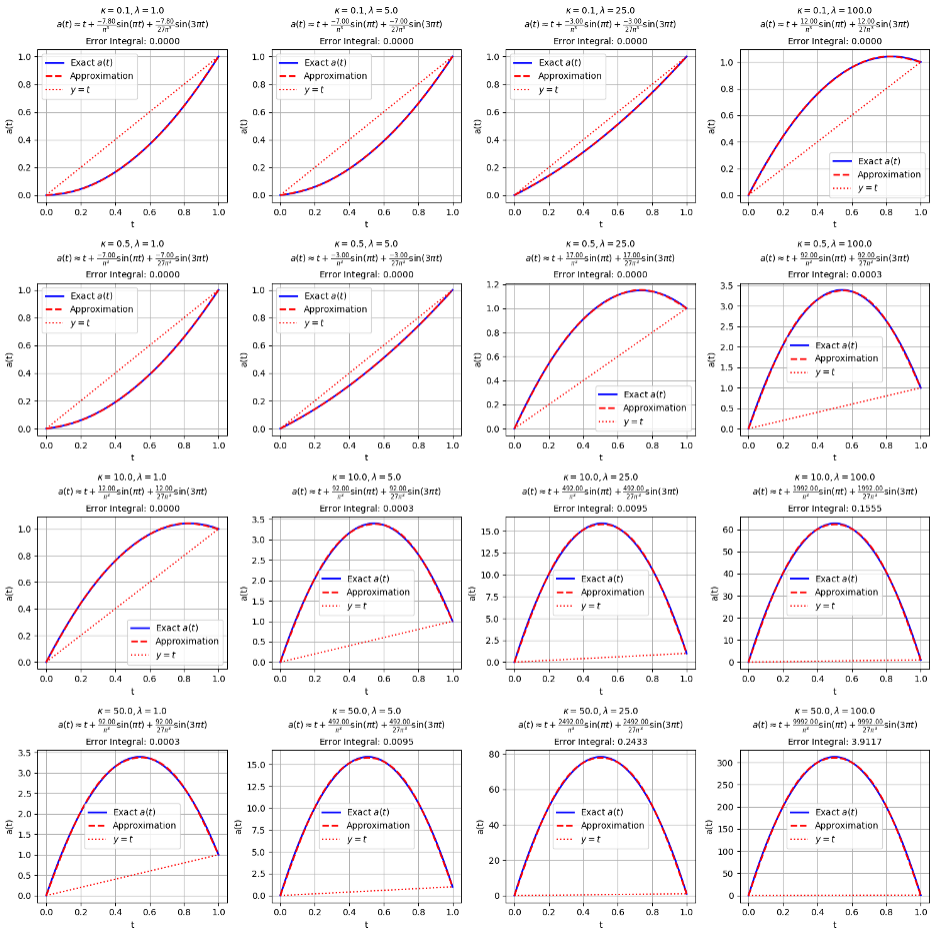

Figure 7: *The approximations in Eq.* (27)

7.2 Computing the Equilibrium strategies:

We now repeat the approximations for equilibrium.

$$a(t) = -\frac{\left(1 - e^{-\frac{\kappa t}{3}}\right)\left(-e^{\kappa/3}\left(e^{\kappa/3} + e^{2\kappa/3} + 1\right)(\lambda+1) + (\lambda-1)e^{\frac{\kappa t}{3}} + (\lambda-1)e^{\frac{2\kappa t}{3}} + (\lambda-1)e^{\kappa t}\right)}{2\left(e^{\kappa} - 1\right)} \tag{28}$$

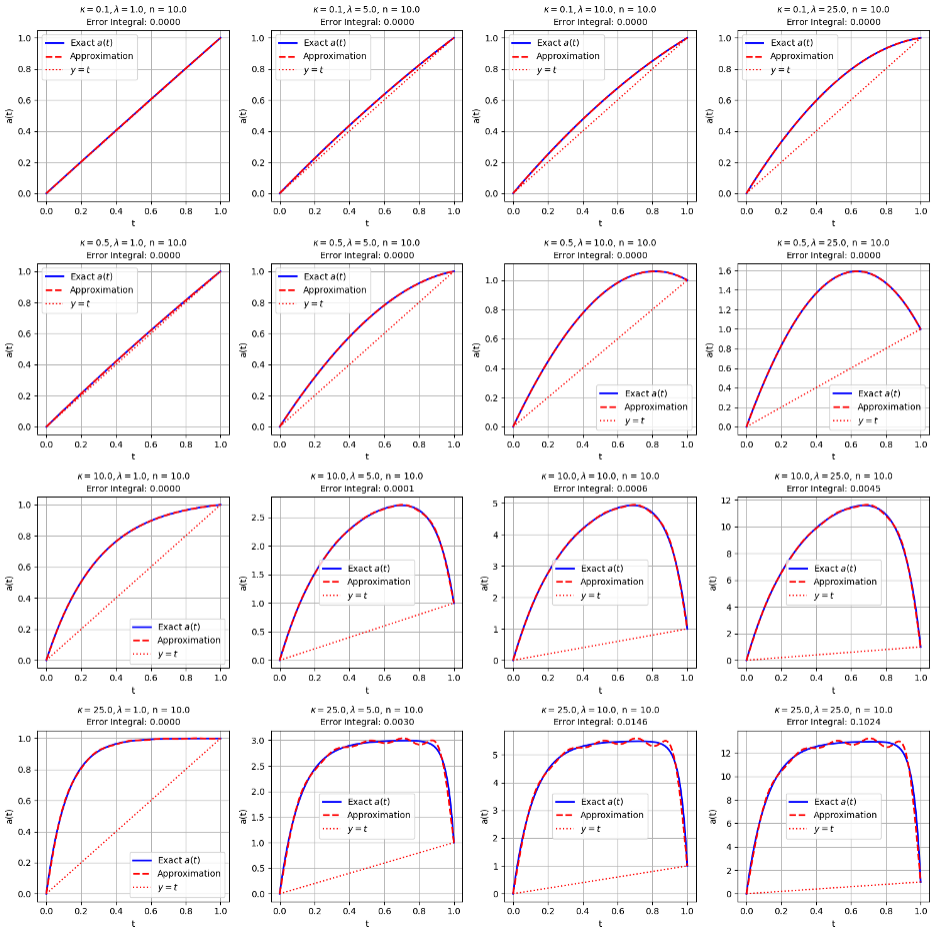

Figure 8: *Ten coefficient approximation*

8 Derivation of cost function approximation

8.1 Compute the temporary market impact component of the cost function

We now proceed along similar lines and compute the temporary impact component of the cost function in terms of the Fourier coefficients:

$$\int_0^1 (\dot{a} + \lambda \dot{b})\dot{a}\, \mathtt{dt} \tag{29}$$

First we compute $\int_0^1 \dot{a}\,\dot{a}\,\mathtt{dt}$ by expanding $\dot{a}\dot{a}$:

$$\dot{a}\,\dot{a} = \left(1 + \sum_{n=1}^{N} a_n n\pi \cos(n\pi t)\right) \cdot \left(1 + \sum_{n=1}^{N} a_n n\pi \cos(n\pi t)\right) \tag{30}$$

$$= 1 + 2\sum_{n=1}^{N} a_n n\pi \cos(n\pi t) + \sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m nm\pi^2 \cos(n\pi t)\cos(m\pi t). \tag{31}$$

We now integrate each term in the expansion starting with $\int_0^1 1\,\mathtt{dt}$:

$$\int_0^1 1\,\mathtt{dt} = 1. \tag{32}$$

Next we compute the integral of the second term $2\sum_{n=1}^{N} a_n n\pi \cos(n\pi t)$. Since the integral of $\cos(n\pi t)$ over $[0,1]$ is zero for any integer n: $\int_0^1 \cos(n\pi t)\,\mathtt{dt} = 0$.

$$\int_0^1 2\sum_{n=1}^{N} a_n n\pi \cos(n\pi t)\,\mathtt{dt} = 0. \tag{33}$$

Finally we compute the third term $\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m nm\pi^2 \cos(n\pi t)\cos(m\pi t)$. Using the trigonometric identity:

$$\cos(n\pi t)\cos(m\pi t) = \frac{1}{2}\left(\cos((n+m)\pi t) + \cos((n-m)\pi t)\right), \tag{34}$$

the integral of these cosine terms is zero unless $n = m$. When $n = m$, the term becomes $\cos^2(n\pi t)$, and $\int_0^1 \cos^2(n\pi t)\,\mathtt{dt} = \frac{1}{2}$. Thus, the integral of $\dot{a}\dot{a}$ becomes:

$$\int_0^1 \sum_{n=1}^{N} a_n^2 n^2 \pi^2 \cos^2(n\pi t)\,\mathtt{dt} = \sum_{1}^{N} \pi^2 \int_0^1 a_n^2 n^2 \cos^2(n\pi t)\,\mathtt{dt} \tag{35}$$

$$= \sum_{n=1}^{N} \pi^2 \int_0^1 a_n^2 n^2 \cos^2(n\pi t)\,\mathtt{dt} \tag{36}$$

$$= \frac{\pi^2}{2}\sum_{n=1}^{N} a_n^2 n^2. \tag{37}$$

Summing the contributions from all terms we obtain the integral of the first term in the temporary impact function:

$$\boxed{\int_0^1 \dot{a}(t)\dot{a}(t)\,\mathtt{dt} = 1 + \frac{\pi^2}{2}\sum_{n=1}^{N} a_n^2 n^2} \tag{38}$$

8.2 Computation of the cost of trading from temporary impact

We proceed by computing

$$\lambda \int_0^1 \dot{b}(t)\dot{a}(t)\,\mathtt{dt} \tag{39}$$

where we recall that:

$$\dot{a} = 1 + \sum_{n=1}^{N} a_n n\pi \cos(n\pi t), \tag{40}$$

$$\dot{b} = 1 + \sum_{n=1}^{N} b_n n\pi \cos(n\pi t). \tag{41}$$

Substituting the Fourier expansions into the first term in the integral, $\dot{b} \cdot \dot{a}$, we obtain:

$$\dot{b} \cdot \dot{a} = \left(1 + \sum_{n=1}^{N} b_n n\pi \cos(n\pi t)\right)\left(1 + \sum_{n=1}^{N} a_n n\pi \cos(n\pi t)\right) \tag{42}$$

$$= 1 + \sum_{n=1}^{N} b_n n\pi \cos(n\pi t) + \sum_{n=1}^{N} a_n n\pi \cos(n\pi t) \tag{43}$$

$$+ \sum_{n=1}^{N}\sum_{m=1}^{N} b_n a_m nm\pi^2 \cos(n\pi t)\cos(m\pi t). \tag{44}$$

The entirety of the expression we need to integrate is

$$\dot{b} \cdot \dot{a} = 1 + \sum_{n=1}^{N}(b_n + a_n)n\pi \cos(n\pi t) + \sum_{n=1}^{N}\sum_{m=1}^{N} b_n a_m nm\pi^2 \cos(n\pi t)\cos(m\pi t) \tag{45}$$

We note that to compute Eq. (45) we need to use the following identities:

$$\int_0^1 \cos(n\pi t)\,\mathtt{dt} = 0, \qquad n > 0, \tag{46}$$

$$\int_0^1 \cos(n\pi t)\cos(m\pi t)\,\mathtt{dt} = \begin{cases} \frac{1}{2} & \text{if } n = m \\ 0 & \text{if } n \neq m \end{cases} \tag{47}$$

We now integrate each term in the expansion. The first term is the constant 1 and $\int_0^1 1\,\mathtt{dt} = 1$. The integral of the second term is a multiple of $\cos(n\pi t)$ which is zero for all n. The third term is the sum of products $\cos(n\pi t)\cos(m\pi t)$ whose integrals are zero unless $n = m$ in which case it is 1/2. Thus the integral of the third term is given by

$$\int_0^1 \sum_{n=1}^{N}\sum_{m=1}^{N} b_n a_m nm\pi^2 \cos(n\pi t)\cos(m\pi t) = \int_0^1 \sum_{n=1}^{N} a_n b_n n^2\pi^2 \cos^2(n\pi t)\,\mathtt{dt} \tag{48}$$

$$= \sum_{1}^{n} \pi^2 \int_0^1 a_n b_n n^2 \cos^2(n\pi t)\,\mathtt{dt} \tag{49}$$

$$= \frac{\pi^2}{2}\sum_{n=1}^{N} a_n b_n n^2 \tag{50}$$

Summing all contributions the second term in temporary impact becomes:

$$\boxed{\lambda \int_0^1 \dot{b}(t)\dot{a}(t)\,\mathtt{dt} = \lambda\Big(1 + \frac{\pi^2}{2}\sum_{n=1}^{N} b_n a_n n^2\Big).}$$

To complete the calculation of the temporary impact term we add in $\int_0^1 \dot{a} \cdot \dot{a}\,\mathtt{dt}$ to obtain:

$$\int_0^1 (\dot{a} + \lambda\dot{b})\dot{a}\,\mathtt{dt} = \int_0^1 \dot{a} \cdot \dot{a}\,\mathtt{dt} + \lambda \int_0^1 \dot{b} \cdot a\,\mathtt{dt} \tag{51}$$

$$= \Big(1 + \frac{\pi^2}{2}\sum_{n=1}^{N} a_n^2 n^2\Big) + \lambda\Big(1 + \frac{\pi^2}{2}\sum_{n=1}^{N} b_n a_n n^2\Big) \tag{52}$$

$$= 1 + \frac{\pi^2}{2}\sum_{n=1}^{N} a_n^2 n^2 + \lambda + \lambda\frac{\pi^2}{2}\sum_{n=1}^{N} b_n a_n n^2 \tag{53}$$

$$= (1+\lambda) + \frac{\pi^2}{2}\sum_{n=1}^{N} \left(a_n^2 + \lambda b_n a_n\right) n^2. \tag{54}$$

and here is the final result for the *temporary impact term*:

$$\boxed{\int_0^1 (\dot{a} + \lambda\dot{b})\dot{a}\,\mathtt{dt} = (1+\lambda) + \frac{\pi^2}{2}\sum_{n=1}^{N} \left(a_n^2 + \lambda b_n a_n\right) n^2} \tag{55}$$

Equation 1: *Temporary impact in terms of Fourier expansions where where $a_1 \cdots a_N$ and $b_1, \ldots b_N$ are the Fourier coefficients for the trading strategies $a(t)$ and $b(t)$, respectively.*

8.3 Permanent impact component of cost function

The integral defining the permanent impact term is $\int_0^1 \kappa\big(a(t) + \lambda b(t)\big)\dot{a}(t)\,\mathtt{dt}$ can be split into two parts:

$$\kappa \int_0^1 a(t)\dot{a}(t)\,\mathtt{dt} + \kappa\lambda \int_0^1 b(t)\dot{a}(t)\,\mathtt{dt}. \tag{56}$$

First let's compute $\int_0^1 a(t)\dot{a}(t)dt$ by substituting the Fourier expansions for $a(t)$ and $\dot{a}(t)$ into $a(t)\dot{a}(t)$:

$$a(t) \cdot \dot{a}(t) = \left(t + \sum_{n=1}^{N} a_n \sin(n\pi t)\right)\left(1 + \sum_{n=1}^{N} a_n n\pi \cos(n\pi t)\right). \tag{57}$$

This gives four terms:

$$\begin{aligned} a(t) \cdot \dot{a}(t) = t + t\sum_{n=1}^{N} a_n n\pi \cos(n\pi t) + \sum_{n=1}^{N} a_n \sin(n\pi t) \\ + \sum_{n=1}^{N} a_n \sin(n\pi t) \sum_{n=1}^{N} a_n n\pi \cos(n\pi t). \end{aligned} \tag{58}$$

We now integrate each term over $[0, 1]$. The first term is simple:

$$\int_0^1 t\,\mathtt{dt} = \frac{1}{2}. \tag{59}$$

The second term is:

$$\int_0^1 \sum_{n=1}^{N} a_n n\pi\, t\, \cos(n\pi t)\, \mathtt{dt}. \tag{60}$$

Recalling the identity:

$$\int_0^1 t\cos(n\pi t)\, \mathtt{dt} = \begin{cases} -\frac{2}{n^2\pi^2} & \text{if } n \in \mathbf{N}^+_{\text{odd}} \\ 0 & \text{otherwise} \end{cases} \tag{61}$$

we only get contributions from odd n, leading to:

$$\int_0^1 \sum_{n=1}^{N} a_n n\pi\, t\, \cos(n\pi t)\, \mathtt{dt} = -\sum_{n\in\mathbf{N}^+_{\text{odd}}}^{N} \frac{2a_n}{n^2\pi^2}. \tag{62}$$

The third term is $\int_0^1 \sum_{n=1}^{N} a_n \sin(n\pi t)\, \mathtt{dt}$ and the integral of $\sin(n\pi t)$ over $[0,1]$ is zero for even and $\frac{2}{n\pi}$ for n odd, and thus:

$$\int_0^1 \sum_{n=1}^{N} a_n \sin(n\pi t)\, \mathtt{dt} = \sum_{n\in\mathbf{N}^+_{\text{odd}}}^{N} \frac{2a_n}{n\pi}. \tag{63}$$

The fourth term involves the product $\sin(n\pi t)\cos(m\pi t)$, and its integral over $[0,1]$ is zero for all n and m. Thus, the total integral is:

$$\int_0^1 a(t)\dot{a}(t)\, \mathtt{dt} \approx \frac{1}{2} - \sum_{n\in\mathbf{N}^+_{\text{odd}}}^{N} \frac{2a_n}{\pi^2 n^2} + \sum_{n\in\mathbf{N}^+_{\text{odd}}}^{N} \frac{2a_n}{n\pi}. \tag{64}$$

Now we integrate the second component of the permanent impact function, $\kappa \int_0^1 b(t)\dot{a}(t)\, \mathtt{dt}$. The integral for $b(t)\dot{a}(t)$ is computed as:

$$\int_0^1 b(t)\dot{a}(t)\, \mathtt{dt} = \frac{1}{2} - \sum_{n\in\mathbf{N}^+_{\text{odd}}}^{N} \frac{2a_n}{\pi^2 n^2} + \sum_{n\in\mathbf{N}^+_{\text{odd}}}^{N} \frac{2b_n}{n\pi}. \tag{65}$$

Final result for permanent impact term: $\kappa \int_0^1 (a+b)\dot{a}\, \dot{\mathtt{dt}}$:

$$\int_0^1 \kappa(a+\lambda b)\dot{a} = \int_0^1 \kappa\Big(a\dot{a} + \lambda b\dot{a}\Big)\, \mathtt{dt} \tag{66}$$

$$= \kappa\Big(\frac{1}{2} - \sum_{n\in\mathbf{N}^+_{\text{odd}}}^{N} \frac{2a_n}{\pi^2 n^2} + \sum_{n\in\mathbf{N}^+_{\text{odd}}}^{N} \frac{2a_n}{n\pi}\Big) \tag{67}$$

$$+ \kappa\lambda\Big(\frac{1}{2} - \sum_{n\in\mathbf{N}^+_{\text{odd}}}^{N} \frac{a_{2n}}{\pi^2 n^2} + \sum_{n\in\mathbf{N}^+_{\text{odd}}}^{N} \frac{2b_n}{n\pi}\Big) \tag{68}$$

$$= \kappa\Big(\frac{1+\lambda}{2} - \sum_{n\in\mathbf{N}^+_{\text{odd}}}^{N} \frac{2(1+\lambda)a_n}{\pi^2 n^2} + \lambda \sum_{n\in\mathbf{N}^+_{\text{odd}}}^{N} \frac{2b_n}{n\pi}\Big). \tag{69}$$

Finally the permanent impact term:

$$\boxed{\int_0^1 \kappa(a+\lambda b)\dot{a} = \kappa \left\{ \frac{1+\lambda}{2} - \sum_{n \in \mathbf{N}^+_{\text{odd}}}^{N} \frac{2(1+\lambda)a_n}{\pi^2 n^2} + \lambda \sum_{n \in \mathbf{N}^+_{\text{odd}}}^{N} \frac{2b_n}{n\pi} \right\}} \tag{70}$$

Equation 2: *The permanent impact term of the cost function in terms of Fourier coefficients, where $a_1, \ldots, a_N$ and $b_1, \ldots b_N$ are the Fourier coefficients for the strategies $a(t)$ and $b(t)$ respectively.*

8.4 Final approximation of the cost function

Putting together the entire set of calculation the approximation of the cost function in terms of the Fourier Coefficients of a and b:

$$\text{Cost}(a;b) = \int_0^1 (\dot{a} + \lambda \dot{b})\dot{a} + \kappa(a+\lambda b)\dot{b}\,\texttt{dt} \tag{71}$$

We can now write the cost function as a function of the Fourier coefficients of strategy $a(t)$ further parameterized by the Fourier coefficients of strategy $b(t)$ and write $\text{Cost}_{\mathcal{F}}(a;b)$ to indicate that this is the *approximate cost function*, specifically approximated by the Fourier coefficients of a and further parameterized by the Fourier coefficients of b:

$$\boxed{\begin{aligned}&\text{Cost}_{\mathcal{F}}(a;b) = \\ &\quad (1+\lambda) + \frac{\pi^2}{2}\sum_{n=1}^{N}\left(a_n^2 + \lambda b_n a_n\right) n^2 + \kappa \left\{ \frac{1+\lambda}{2} - \sum_{n \in \mathbf{N}^+_{\text{odd}}}^{N} \frac{2(1+\lambda)a_n}{\pi^2 n^2} + \lambda \sum_{n \in \mathbf{N}^+_{\text{odd}}}^{N} \frac{2b_n}{n\pi} \right\}\end{aligned}}$$

Equation 3: *Cost function $\int_0^1 (\dot{a} + \lambda \dot{b})\dot{a} + \kappa(a+\lambda b)\dot{b}\,\texttt{dt}$ in terms of Fourier coefficients, where $a_1 \cdots a_N$ and $b_1, \ldots b_N$ are the Fourier coefficients for the trading strategies $a(t)$ and $b(t)$, respectively. The first term is the* temporary impact term *and the second is the* permanent impact term.

9 Code

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import quad

# Define the exact function a(t)
def a_exact(t, kappa, lambda_):
    return (1 - (lambda_ * kappa) / 4) * t**2 + (lambda_ * kappa) / 4 * t

# Define the Fourier series approximation for a(t)
def a_approx(t, kappa, lambda_):
    a1 = (2 * (-4 + lambda_ * kappa)) / np.pi**3
    a3 = (2 * (-4 + lambda_ * kappa)) / (27 * np.pi**3)
    return t + a1 * np.sin(np.pi * t) + a3 * np.sin(3 * np.pi * t)

# Function to compute the squared difference
def squared_difference(t, kappa, lambda_):
    return (a_exact(t, kappa, lambda_) - a_approx(t, kappa, lambda_))**2

# Create the grid of values for kappa and lambda
kappa_values = [0.1, 0.5, 10, 50]
lambda_values = [1, 5, 25, 100]

# Create a grid of plots
fig, axs = plt.subplots(len(kappa_values), len(lambda_values), figsize=(15,
15))
t = np.linspace(0, 1, 500)

# Loop over the grid to plot for each (kappa, lambda) combination
for i, kappa in enumerate(kappa_values):
    for j, lambda_ in enumerate(lambda_values):
        # Compute the exact and approximated values of a(t)
        a_exact_vals = a_exact(t, kappa, lambda_)
        a_approx_vals = a_approx(t, kappa, lambda_)

        # Plot the exact and approximated functions
        axs[i, j].plot(t, a_exact_vals, label='Exact $a(t)$', color='blue',
linewidth=2)
        axs[i, j].plot(t, a_approx_vals, label='Approximation', color='red',
linestyle='--', linewidth=2)

        # Plot the dashed red line for y=t
        axs[i, j].plot(t, t, label='$y = t$', color='red', linestyle=':',
linewidth=1.5)

        # Compute the squared error integral
        error_integral, _ = quad(squared_difference, 0, 1, args=(kappa,
lambda_))

        # Set plot titles with the Fourier approximation formula in LaTeX
        a1 = (2 * (-4 + lambda_ * kappa)) / np.pi**3
        a3 = (2 * (-4 + lambda_ * kappa)) / (27 * np.pi**3)

        # LaTeX-style formula for title
        title_formula = (
            r'$a(t) \approx t + \frac{%.2f}{\pi^3} \sin(\pi t) + \frac{%.2f
}{27 \pi^3} \sin(3 \pi t)$'
            % (2 * (-4 + lambda_ * kappa), 2 * (-4 + lambda_ * kappa))
        )

        # Set title combining kappa, lambda, formula, and the error integral
value
```

```
            axs[i, j].set_title(r'$\kappa = %.1f, \lambda = %.1f$' % (kappa, lambda_) + '\n'
                                + title_formula + '\n' + r'Error Integral: $%.4f$' % error_integral, fontsize=10)

            axs[i, j].set_xlabel('t')
            axs[i, j].set_ylabel('a(t)')
            axs[i, j].legend()
            axs[i, j].grid(True)  # Add gridlines

    # Adjust layout for better spacing
    plt.tight_layout()
    plt.show()
```

Listing 1: *Python code for plotting a(t) and its Fourier approximation*

```
    import numpy as np
    import matplotlib.pyplot as plt
    from scipy.integrate import quad

    # Create the grid of values for kappa and lambda
    kappa_values = [0.1, 0.5, 10, 25]
    lambda_values = [1, 5, 10, 25]
    n_terms = 10


    # Define the given function a(t)
    def a_func(t, kappa, lambda_):
        term1 = (1 - np.exp(-kappa * t / 3))
        term2 = -np.exp(kappa / 3) * (np.exp(kappa / 3) + np.exp(2 * kappa / 3) +
    1) * (lambda_ + 1)
        term3 = (lambda_ - 1) * np.exp(kappa * t / 3)
        term4 = (lambda_ - 1) * np.exp(2 * kappa * t / 3)
        term5 = (lambda_ - 1) * np.exp(kappa * t)
        return -(term1 * (term2 + term3 + term4 + term5)) / (2 * (np.exp(kappa) -
    1))

    # Define the integrand for the Fourier coefficient a_n
    def integrand(t, n, kappa, lambda_):
        return (a_func(t, kappa, lambda_) - t) * np.sin(n * np.pi * t)


    # Define the Fourier series approximation for a(t) using provided coefficients
    def a_approx(t, kappa, lambda_, n_terms):

        def compute_approx(t, kappa, lambda_, n_terms):
            approx = 0
            for n in range(1, n_terms+1):
                coeff, _ = quad(integrand, 0, 1, args=(n, kappa, lambda_))
                approx += 2 *coeff * np.sin(n * np.pi * t)
            return approx

        return t + compute_approx(t, kappa, lambda_, n_terms)

    # Function to compute the squared difference between exact and approximate
    functions
    def squared_difference(t, kappa, lambda_, n_terms):
        return (a(t, kappa, lambda_) - a_approx(t, kappa, lambda_, n_terms))**2


    # Create a grid of plots
    fig, axs = plt.subplots(len(kappa_values), len(lambda_values), figsize=(15,
    15))
    t = np.linspace(0, 1, 500)
    z = np.zeros(500)

    # Loop over the grid to plot for each (kappa, lambda) combination
    for i, kappa in enumerate(kappa_values):
        for j, lambda_ in enumerate(lambda_values):
            # Compute the exact and approximated values of a(t)
            a_exact_vals = a(t, kappa, lambda_)
            a_approx_vals = a_approx(t, kappa, lambda_, n_terms)

            # Plot the exact and approximated functions
            axs[i, j].plot(t, a_exact_vals, label='Exact $a(t)$', color='blue',
    linewidth=2)
            axs[i, j].plot(t, a_approx_vals, label='Approximation', color='red',
    linestyle='--', linewidth=2)

            # Plot the dashed red line for y=t
```

```
            axs[i, j].plot(t, t, label='$y = t$', color='red', linestyle=':',
    linewidth=1.5)

            # Compute the squared error integral
            error_integral, _ = quad(squared_difference, 0, 1, args=(kappa,
    lambda_, n_terms))

            # Set title combining kappa, lambda, formula, and the error integral
    value
            axs[i, j].set_title(r'$\kappa = %.1f, \lambda = %.1f$, n = %.1f' % (
    kappa, lambda_, n_terms)
```

Listing 2: *Python code for plotting a(t) and its Fourier approximation for equilibrium best-response strategy*

```
    import numpy as np
    import matplotlib.pyplot as plt
    from scipy.integrate import quad

    # Define the exact function a(t)
    def a_exact(t, kappa, lambda_):
        return (1 - (lambda_ * kappa) / 4) * t**2 + (lambda_ * kappa) / 4 * t

    # Define the Fourier series approximation for a(t)
    def a_approx(t, kappa, lambda_):
        a1 = (2 * (-4 + lambda_ * kappa)) / np.pi**3
        a3 = (2 * (-4 + lambda_ * kappa)) / (27 * np.pi**3)
        return t + a1 * np.sin(np.pi * t) + a3 * np.sin(3 * np.pi * t)

    # Function to compute the squared difference
    def squared_difference(t, kappa, lambda_):
        return (a_exact(t, kappa, lambda_) - a_approx(t, kappa, lambda_))**2

    # Create the grid of values for kappa and lambda
    kappa_values = [0.1, 0.5, 10, 50]
    lambda_values = [1, 5, 25, 100]

    # Create a grid of plots
    fig, axs = plt.subplots(len(kappa_values), len(lambda_values), figsize=(15,
    15))
    t = np.linspace(0, 1, 500)

    # Loop over the grid to plot for each (kappa, lambda) combination
    for i, kappa in enumerate(kappa_values):
        for j, lambda_ in enumerate(lambda_values):
            # Compute the exact and approximated values of a(t)
            a_exact_vals = a_exact(t, kappa, lambda_)
            a_approx_vals = a_approx(t, kappa, lambda_)

            # Plot the exact and approximated functions
            axs[i, j].plot(t, a_exact_vals, label='Exact $a(t)$', color='blue',
    linewidth=2)
            axs[i, j].plot(t, a_approx_vals, label='Approximation', color='red',
    linestyle='--', linewidth=2)

            # Plot the dashed red line for y=t
            axs[i, j].plot(t, t, label='$y = t$', color='red', linestyle=':',
    linewidth=1.5)

            # Compute the squared error integral
            error_integral, _ = quad(squared_difference, 0, 1, args=(kappa,
    lambda_))

            # Set plot titles with the Fourier approximation formula in LaTeX
            a1 = (2 * (-4 + lambda_ * kappa)) / np.pi**3
            a3 = (2 * (-4 + lambda_ * kappa)) / (27 * np.pi**3)

            # LaTeX-style formula for title
            title_formula = (
                r'$a(t) \approx t + \frac{%.2f}{\pi^3} \sin(\pi t) + \frac{%.2f
    }{27 \pi^3} \sin(3 \pi t)$'
                % (2 * (-4 + lambda_ * kappa), 2 * (-4 + lambda_ * kappa))
            )

            # Set title combining kappa, lambda, formula, and the error integral
    value
            axs[i, j].set_title(r'$\kappa = %.1f, \lambda = %.1f$' % (kappa,
    lambda_) + '\n'
```

```
                                + title_formula + '\n' + r'Error Integral: $%.4f$'
    % error_integral, fontsize=10)

            axs[i, j].set_xlabel('t')
            axs[i, j].set_ylabel('a(t)')
            axs[i, j].legend()
            axs[i, j].grid(True)  # Add gridlines

    # Adjust layout for better spacing
    plt.tight_layout()
    plt.show()
```

Listing 3: *Python code for plotting a(t) and its Fourier approximation*

```
    import numpy as np
    import matplotlib.pyplot as plt
    from scipy.integrate import quad

    # Create the grid of values for kappa and lambda
    kappa_values = [0.1, 0.5, 10, 25]
    lambda_values = [1, 5, 10, 25]
    n_terms = 10


    # Define the given function a(t)
    def a_func(t, kappa, lambda_):
        term1 = (1 - np.exp(-kappa * t / 3))
        term2 = -np.exp(kappa / 3) * (np.exp(kappa / 3) + np.exp(2 * kappa / 3) +
    1) * (lambda_ + 1)
        term3 = (lambda_ - 1) * np.exp(kappa * t / 3)
        term4 = (lambda_ - 1) * np.exp(2 * kappa * t / 3)
        term5 = (lambda_ - 1) * np.exp(kappa * t)
        return -(term1 * (term2 + term3 + term4 + term5)) / (2 * (np.exp(kappa) -
    1))

    # Define the integrand for the Fourier coefficient a_n
    def integrand(t, n, kappa, lambda_):
        return (a_func(t, kappa, lambda_) - t) * np.sin(n * np.pi * t)


    # Define the Fourier series approximation for a(t) using provided coefficients
    def a_approx(t, kappa, lambda_, n_terms):

        def compute_approx(t, kappa, lambda_, n_terms):
            approx = 0
            for n in range(1, n_terms+1):
                coeff, _ = quad(integrand, 0, 1, args=(n, kappa, lambda_))
                approx += 2 *coeff * np.sin(n * np.pi * t)
            return approx

        return t + compute_approx(t, kappa, lambda_, n_terms)

    # Function to compute the squared difference between exact and approximate
    functions
    def squared_difference(t, kappa, lambda_, n_terms):
        return (a(t, kappa, lambda_) - a_approx(t, kappa, lambda_, n_terms))**2


    # Create a grid of plots
    fig, axs = plt.subplots(len(kappa_values), len(lambda_values), figsize=(15,
    15))
    t = np.linspace(0, 1, 500)
    z = np.zeros(500)

    # Loop over the grid to plot for each (kappa, lambda) combination
    for i, kappa in enumerate(kappa_values):
        for j, lambda_ in enumerate(lambda_values):
            # Compute the exact and approximated values of a(t)
            a_exact_vals = a(t, kappa, lambda_)
            a_approx_vals = a_approx(t, kappa, lambda_, n_terms)

            # Plot the exact and approximated functions
            axs[i, j].plot(t, a_exact_vals, label='Exact $a(t)$', color='blue',
    linewidth=2)
            axs[i, j].plot(t, a_approx_vals, label='Approximation', color='red',
    linestyle='--', linewidth=2)

            # Plot the dashed red line for y=t
```

```
            axs[i, j].plot(t, t, label='$y = t$', color='red', linestyle=':',
    linewidth=1.5)

            # Compute the squared error integral
            error_integral, _ = quad(squared_difference, 0, 1, args=(kappa,
    lambda_, n_terms))

            # Set title combining kappa, lambda, formula, and the error integral
    value
            axs[i, j].set_title(r'$\kappa = %.1f, \lambda = %.1f$, n = %.1f' % (
    kappa, lambda_, n_terms)
                                + '\n' + r'Error Integral: $%.4f$' %
    error_integral, fontsize=10)

            axs[i, j].set_xlabel('t')
            axs[i, j].set_ylabel('a(t)')
            axs[i, j].legend()
            axs[i, j].grid(True)  # Add gridlines

    # Adjust layout for better spacing
    plt.tight_layout()
    plt.show()
```

Listing 4: *Python code for plotting a(t) and its Fourier approximation for equilibrium best-response strategy*

10 Acknowledgements

I extend my sincere thanks to the Machine Learning Research Group at Morgan Stanley, to whom I gave two seminars on this work in August of 2024 during which they made many insightful suggests which greatly improved this work. I also extend my gratitude to Mike Shelley of the Courant Institute and Flatiron Institute for his invaluable guidance and Jim Gatheral of Baruch college for many illuminating discussions on transaction costs.

References

[Chriss(2024)] Neil A. Chriss. Optimal position-building in competition, 2024. URL https://arxiv.org/abs/2409.03586.