

**Михаил Шардин**

личный блог



19 марта 2025, 04:44

[+ Подписаться](#)

## Тестировании торговой системы Александра Резвякова для фьючерсов Московской биржи с использованием Python

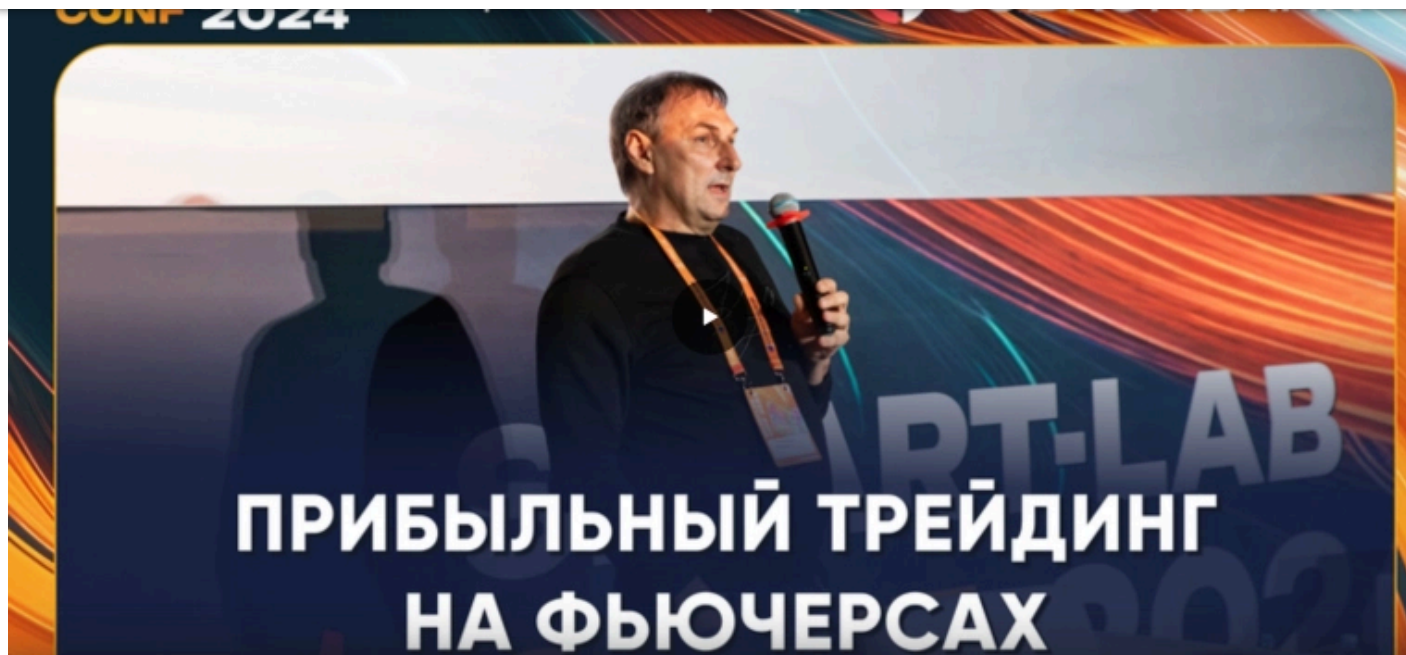
В этой статье расскажу о том, как воспроизвел и протестировал торговую систему для фьючерсов Московской биржи, основанную на идеях Александра Резвякова. Недавно, просматривая раздел алготрейдинга на Смартлабе, я наткнулся на видео с его выступления на конференции 2024 года под названием "[5-6 идей для построения прибыльной торговой системы на фьючерсах](#)". Меня привлекла четкость и понятность предложенных им правил торговли.

Поскольку я активно занимаюсь автоматизацией процессов и стремлюсь глубже изучить возможности Python библиотеки `backtesting.py`, мне показалось это хорошей идеей для практического применения.

Хотя я лично не знаком с Александром, полагаю, что публичное представление идеи предполагает возможность её независимого анализа и тестирования сообществом трейдеров и программистов.

**БИЛЕТОВ  
ВСЕ МЕНЬШЕ****УСПЕТЬ КУПИТЬ**

Введите текст комментария



## Обзор стратегии Александра Резвякова на фьючерсах

Основная идея — открывать сделки в строго определенное время и использовать структуру рынка последних дней для принятия решений.

### Правила входа

- Время входа: сделки совершаются только в 10:00.

Лонг:

- Должно быть не менее трех дневных свечей в наличии.
- За последние два дня минимумы (Low) и максимумы (High) должны повышаться.
- Вероятность входа — 50%.

Шорт:

- Также минимум три дневных бара.
- За последние два дня максимумы и минимумы снижаются.
- Вероятность входа — 50%.

### Правила выхода

- По времени: позиция закрывается в 20:40, независимо от результата.

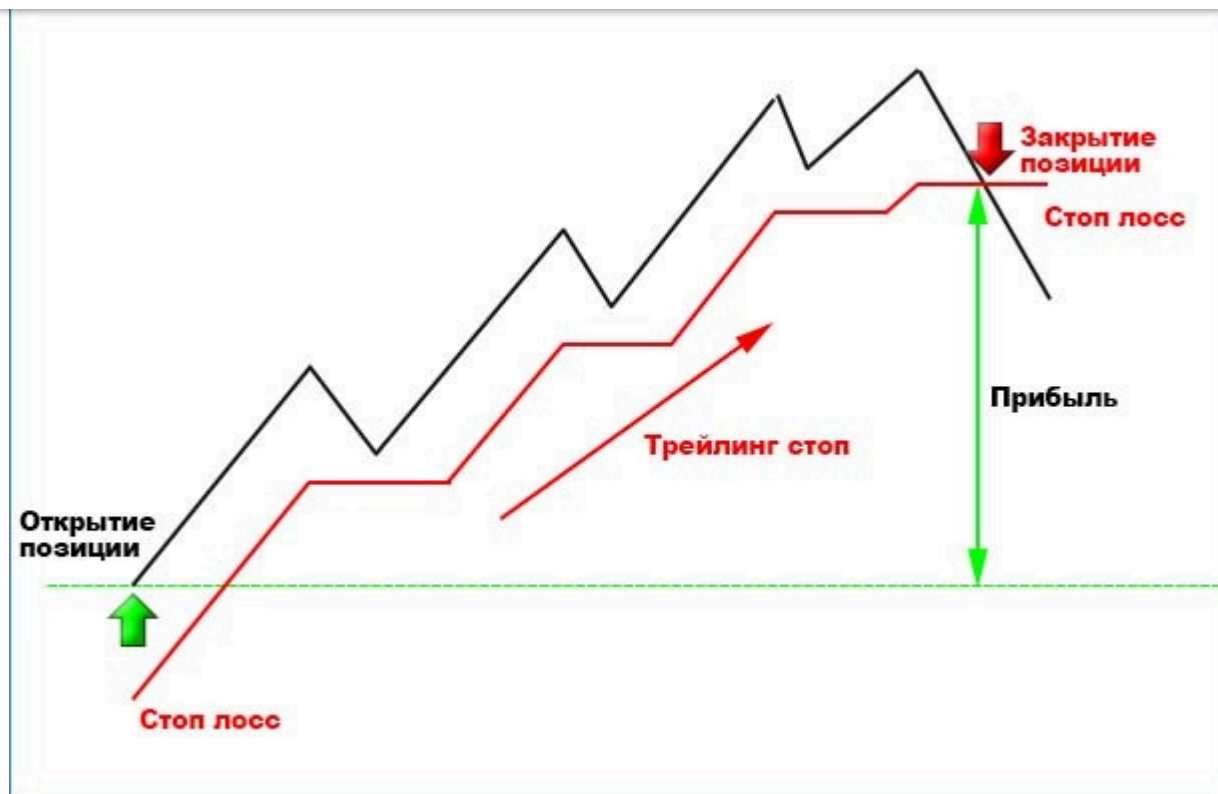
По трейлинг-стопу:

- Для ЛОНГА: если цена закрытия предыдущего бара падает больше чем на 2% от уровня покупки, то продажа. Если цена закрытия бара увеличивается, то этот уровень 2% убытка отчитывается уже от этой нового максимального уровня цены закрытия бара.
- Для ШОРТА: обратно лонгу — если цена растет на определенный процент от

**БИЛЕТОВ  
ВСЕ МЕНЬШЕ**



**УСПЕТЬ КУПИТЬ**



Для лонга

### Подготовка среды для тестирования

Для реализации и тестирования стратегии Александра Резвякова я выбрал Python и библиотеку `backtesting.py`. Этот выбор обусловлен несколькими факторами: Python предоставляет гибкие возможности для работы с данными, а `backtesting.py` позволяет эффективно моделировать торговые стратегии.

**БИЛЕТОВ  
ВСЕ МЕНЬШЕ**



**УСПЕТЬ КУПИТЬ**

```

14 class RandomEntryStrategy(Strategy):
15     # Параметры стратегии
16     trail_percent = 0.002 # Процент трейлинг-стопа 2%
17     entry_probability = 0.5 # Вероятность входа в рынок 50/50
18     entry_time = "10:00" # Время для входа в позицию
19     exit_time = "20:00" # Время для выхода из позиции
20
21     def init(self):
22         """
23         Инициализация индикаторов и переменных
24         """
25         # Подготовка данных для дневных баров
26         self.daily_data = self.resample_to_daily()
27
28         # Создаем индикатор для отображения трейлинг-стопа на графике
29         # Используем тот же тип данных, что и у Close, и заполним нулями
30         self.trailing_stops = self.I(lambda: rp.full_like(self.data.Close, np.nan, dtype=float),
31                                     overlay=True, name="трейлинг-стоп")
32
33         # Для отслеживания, активен ли трейлинг-стоп (для правильного отображения)
34         self.is_trailing_stop_active = False
35
36         # Для хранения цены входа и максимальной/минимальной цены закрытия
37         self.entry_prices = {} # Словарь для хранения цен входа по ID позиции
38         self.max_close_prices = {} # Максимальные цены закрытия для лонгов
39         self.min_close_prices = {} # Минимальные цены закрытия для шортов
40
41         # Создаем время входа и выхода как объекты time
42         self.entry_time_obj = pd.to_datetime(self.entry_time).time()
43         self.exit_time_obj = pd.to_datetime(self.exit_time).time()
44
45     def resample_to_daily(self):
46         """
47         Пересемплирование данных в дневные бары
48         """
49         # Создаем копию данных и устанавливаем индекс как дату
50         daily_df = pd.DataFrame(index=self.data.index)
51         daily_df['Open'] = self.data.Open
52         daily_df['High'] = self.data.High

```

В своем проекте применил модульный подход, разделив функциональность на несколько основных компонентов:

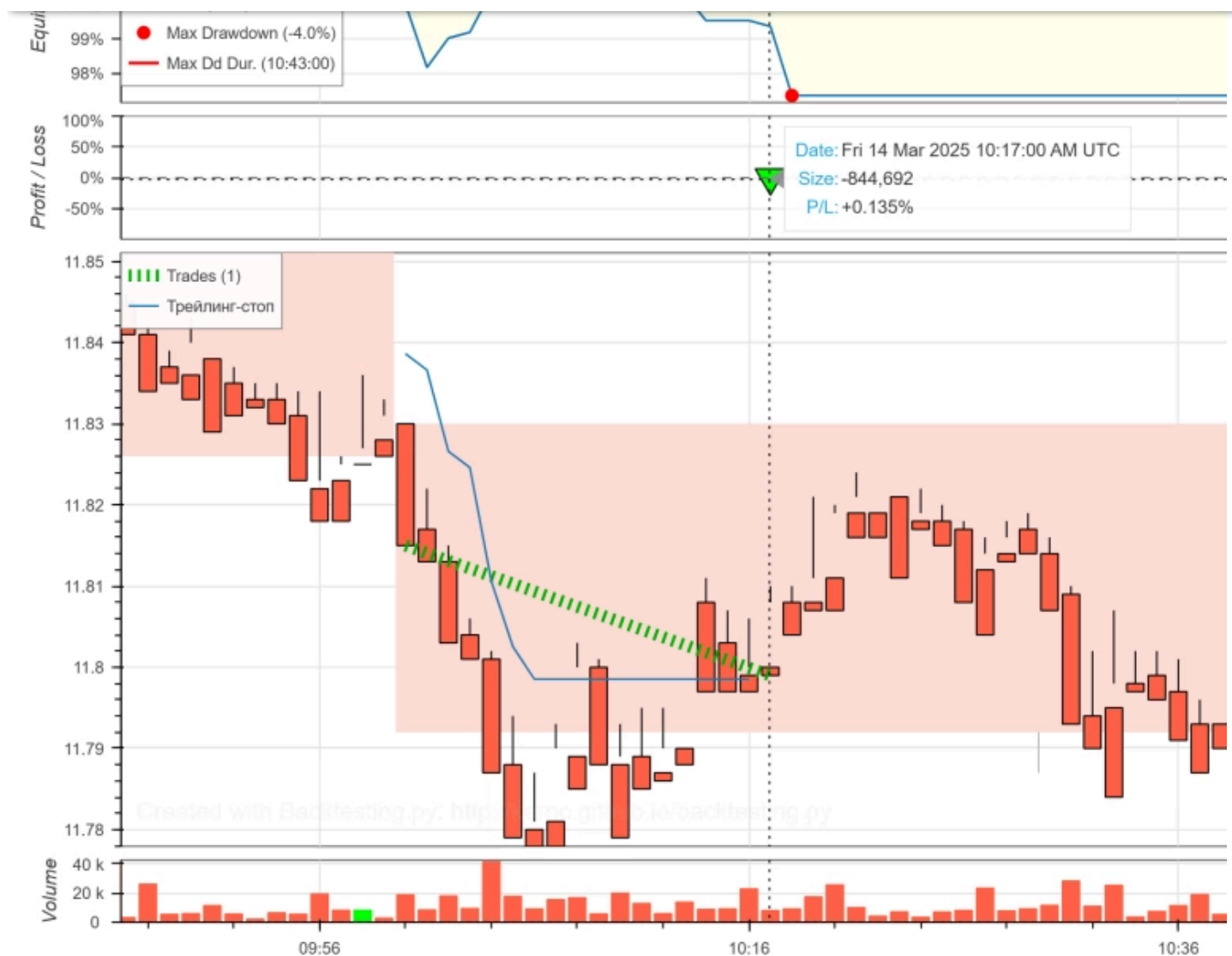
- main.py — центральный модуль, запускающий весь процесс сканирования данных и бэктестирования стратегии
- resample\_data\_1min.py — отвечает за фильтрацию и преобразование данных, получаемых через API; выбирает только необходимый временной интервал котировок
- data\_loader.py — преобразует данные из CSV-формата в структуру, пригодную для работы с backtesting.py
- backtester.py — содержит основную логику запуска и настройки бэктеста
- strategy\_Random\_1min.py — реализует саму торговую систему Резвякова с описанными правилами входа и выхода

Хотя в оригинальном видео Александр Резвяков рекомендовал работать на 5-минутном таймфрейме, я принял решение использовать 1-минутные свечи. Это позволяет более детально анализировать движение цены и точнее отслеживать срабатывание трейлинг-стопов. Благодаря более гранулярным данным, становится проще визуализировать и понять всю логику выхода из позиции, особенно в критических моментах, когда цена колеблется вблизи уровня стопа.

**БИЛЕТОВ  
ВСЕ МЕНЬШЕ**



**УСПЕТЬ КУПИТЬ**



Шорт фьючерса CRH5 (CNY-3.25)

[Полный исходный код всех модулей доступен в моем репозитории на GitHub](#) для тех, кто захочет воспроизвести эксперимент или улучшить предложенную реализацию.

### Реализация стратегии на Python с использованием `backtesting.py`

Вот подробное описание каждого шага с точки зрения программирования:

### Подготовка данных:

Первым шагом является загрузка и предобработка исторических данных о фьючерсах. Модуль `resample_data_1min.py` отвечает за получение минутных данных с использованием API брокера (в данном случае, T-Invest API) и выборку необходимых временных интервалов. Он загружает данные из CSV-файлов, содержащих котировки, и фильтрует их по заданному диапазону дат. Ключевым моментом является функция `filter by date`, которая обеспечивает

**БИЛЕТОВ  
ВСЕ МЕНЬШЕ**



**УСПЕТЬ КУПИТЬ**



```

1  ticker;timestamp;Open;High;Low;Close;Volume
2  CRH5;2025-03-10 05:59:00;12.123;12.113;12.123;12.112;353
3  CRH5;2025-03-10 06:00:00;12.113;12.134;12.134;12.111;19132
4  CRH5;2025-03-10 06:01:00;12.133;12.137;12.139;12.133;7061
5  CRH5;2025-03-10 06:02:00;12.136;12.125;12.141;12.125;10509
6  CRH5;2025-03-10 06:03:00;12.125;12.131;12.136;12.125;6342
7  CRH5;2025-03-10 06:04:00;12.131;12.133;12.138;12.13;3322
8  CRH5;2025-03-10 06:05:00;12.133;12.144;12.146;12.133;8200
9  CRH5;2025-03-10 06:06:00;12.143;12.141;12.144;12.136;4279
10 CRH5;2025-03-10 06:07:00;12.141;12.138;12.142;12.137;3213
11 CRH5;2025-03-10 06:08:00;12.137;12.137;12.138;12.137;1630
12 CRH5;2025-03-10 06:09:00;12.136;12.126;12.136;12.125;3562
13 CRH5;2025-03-10 06:10:00;12.126;12.125;12.129;12.125;10635
14 CRH5;2025-03-10 06:11:00;12.125;12.129;12.134;12.108;6272
15 CRH5;2025-03-10 06:12:00;12.13;12.125;12.13;12.121;2230
16 CRH5;2025-03-10 06:13:00;12.125;12.125;12.13;12.124;980
17 CRH5;2025-03-10 06:14:00;12.125;12.13;12.13;12.124;1612
18 CRH5;2025-03-10 06:15:00;12.13;12.136;12.137;12.129;5119
19 CRH5;2025-03-10 06:16:00;12.136;12.133;12.137;12.129;1528
20 CRH5;2025-03-10 06:17:00;12.133;12.131;12.133;12.13;365
21 CRH5;2025-03-10 06:18:00;12.132;12.132;12.132;12.131;482
22 CRH5;2025-03-10 06:19:00;12.132;12.131;12.132;12.13;579
23 CRH5;2025-03-10 06:20:00;12.132;12.132;12.132;12.13;456

```

## Загрузка данных из CSV:

Модуль `data_loader.py` отвечает за загрузку исторических данных из CSV-файлов и преобразование их в формат, пригодный для использования с библиотекой `backtesting.py`. Функция `load_data_for_ticker` считывает данные для указанного тикера, выполняет парсинг дат и устанавливает столбец `timestamp` в качестве индекса. Это позволяет `backtesting.py` правильно интерпретировать временные ряды.

## Реализация условий для входа и выхода:

В модуле `strategy_Random_1min.py` реализована основная логика торговой стратегии. Класс `RandomEntryStrategy` наследуется от класса `Strategy` из библиотеки `backtesting.py`. В методе `init` происходит инициализация необходимых параметров и индикаторов.

Метод `resample_to_daily` пересчитывает минутные данные в дневные бары, необходимые для принятия решений о входе в позицию.

В методе `next` реализуются правила входа и выхода из позиции, описанные Александром Резвяковым. Время входа строго фиксировано (10:00), а решение о входе в лонг или шорт принимается на основе анализа предыдущих двух дневных баров. Вероятность входа в

сделку устанавливается как 50%. Время выхода установлено на 20:40

**БИЛЕТОВ  
ВСЕ МЕНЬШЕ**



**УСПЕТЬ КУПИТЬ**



Интерактивный HTML

## Настройка трейлинг-стопов:

Для реализации трейлинг-стопов в `strategy_Random_1min.py` используются методы `update_long_position` и `update_short_position`. В этих методах вычисляется уровень трейлинг-стопа на основе текущей цены и заданного процента (2%).

Трейлинг-стоп автоматически подтягивается вверх для длинных позиций и вниз для коротких, обеспечивая фиксацию прибыли и ограничение убытков. При достижении ценой уровня трейлинг-стопа позиция автоматически закрывается.

## Тестирование стратегии

Библиотека `backtesting.py` позволяет не только моделировать сделки, но и визуализировать их в интерактивных HTML-отчетах. Это помогает детально анализировать точки входа и выхода, проверять корректность реализации логики и выявлять возможные ошибки в коде.

Чтобы избежать нагромождения данных в интерактивных HTML-отчетах решил загружать данные по неделям.

Кроме того, я задал комиссию в размере 0,2%, поскольку комиссии оказывают заметное влияние на итоговую доходность:

```
<code>
```

```
bt = Backtest(
    df,
    DynamicStrategyClass
```

**БИЛЕТОВ  
ВСЕ МЕНЬШЕ**



**УСПЕТЬ КУПИТЬ**

```
hedging=False,  
)  
</code>
```

Так как стратегия использует случайный вход в сделки (вероятность 50% на покупку или продажу), результаты тестирования также носят случайный характер. Однако, уже на первых тестах удалось увидеть интересные закономерности.

Вот пример удачного теста на фьючерсе NGH5:



Шорт

```
<code>
```

```
=====
🚀 Запуск бэктеста для NGH5
=====
```

📄 Технические данные (для справки):

Колонки в данных: ['ticker', 'Open', 'High', 'Low', 'Close', 'Volume']

Первые 5 строк данных:

timestamp	ticker	Open	High	Low	Close	Volume
2025-03-10 05:59:00	NGH5	4.610	4.610	4.610	4.576	8743
2025-03-10 06:00:00	NGH5	4.610	4.605	4.629	4.605	10655
2025-03-10 06:01:00	NGH5	4.605	4.592	4.607	4.592	8855
2025-03-10 06:02:00	NGH5	4.592	4.597	4.598	4.592	5293

БИЛЕТОВ  
ВСЕ МЕНЬШЕ

УСПЕТЬ КУПИТЬ



```
Open      float64
High      float64
Low       float64
Close     float64
Volume    int64
dtype: object
```

🕒 Запуск бэктеста...

ПРОДАЖА на 2025-03-12: Цена: 4.33, Стоп: 4.33

ВЫКУП (трейлинг-стоп) на 2025-03-12: Цена: 4.31, Стоп: 4.31, Вход: 4.33, Мин.зак

ПРОДАЖА на 2025-03-14: Цена: 4.08, Стоп: 4.09

ВЫКУП (трейлинг-стоп) на 2025-03-14: Цена: 4.02, Стоп: 4.02, Вход: 4.08, Мин.зак

📊 Результаты бэктеста:

⚙️ Стратегия: NGH5\_RandomEntryStrategy

📅 Период тестирования: с 2025-03-10 05:59:00 по 2025-03-14 20:49:00

💰 Начальный капитал: 100,000 руб.

📈 Конечный капитал: 1104162.00 руб.

📈 Общая доходность: 10.42%

📊 Годовая доходность: 14652.07%

📈 Коэффициент Шарпа: 1.11

📉 Максимальная просадка: -6.51%

🔄 Количество сделок: 2

✅ Процент выигрышных сделок: 100.00%

👉 Лучшая сделка: +1.40%

😞 Худшая сделка: 0.44%

🕒 Средняя продолжительность сделки: 0 days 00:40:00

📊 Построение графика результатов...

✅ График успешно построен!

```
=====
🎯 Бэктест для NGH5 завершен
=====
</code>
```

**БИЛЕТОВ  
ВСЕ МЕНЬШЕ**



**УСПЕТЬ КУПИТЬ**

некоторых тестах условия накатаивались быстрее, чем приливные сделки перекрывали потери. В будущем стоит рассмотреть добавление дополнительных фильтров для входа.

## Рекомендации по улучшению стратегии

### Наращивание позиций в выигрышных сделках

Автор рекомендует увеличивать объем позиции в прибыльных сделках. Однако в текущей реализации теста этот момент не был учтен. Основная сложность в том, как правильно идентифицировать момент для добавления к позиции и реализовать это в `backtesting.py`.

Если у кого-то из читателей есть идеи или готовые решения, буду рад обсудить их.

### Возможные улучшения стратегии

- Оптимизация трейлинг-стопа

В текущей версии трейлинг-стоп фиксирован на уровне 2%, однако более гибкий подход, основанный на волатильности актива, может улучшить результаты. Например, можно использовать ATR (Average True Range) для динамического определения уровня стопа.

- Добавление фильтров для входа

Вход в 10:00 производится без дополнительных условий, что делает стратегию чувствительной к случайным колебаниям рынка. Можно добавить индикаторы тренда (например, SMA, VWAP) или фильтры объема, чтобы подтверждать сигналы на вход.

- Добавление адаптивного модуля управления рисками

Вместо фиксированного размера позиции можно вводить динамическое управление рисками на основе максимальной просадки (drawdown) или волатильности актива.

Все эти идеи требуют детального тестирования, но их внедрение может значительно повысить эффективность стратегии.

## Заключение

В рамках этой статьи я воспроизвел и протестировал торговую систему Александра Резвякова на фьючерсах Московской биржи, используя Python и библиотеку `backtesting.py`.

Несмотря на кажущуюся простоту (случайные входы в 10:00 с вероятностью 50%), стратегия показала интересные результаты благодаря четким правилам выхода из позиций: трейлинг-стопа и закрытие по времени в 20:40.

Данный опыт может быть особенно полезен начинающим алгоритмическим трейдерам.

- Во-первых, он наглядно демонстрирует, что даже базовая стратегия со случайными входами может приносить прибыль при грамотном управлении рисками.
- Во-вторых, проект служит отличным примером структурирования кода для торговых

**БИЛЕТОВ  
ВСЕ МЕНЬШЕ**



**УСПЕШЬ КУПИТЬ**

Что касается перспектив развития, система обладает значительным потенциалом для улучшения. Добавление динамических трейлинг-стопов на основе ATR, интеграция дополнительных фильтров для входа и реализация механизма наращивания позиций в прибыльных сделках могут существенно повысить эффективность стратегии. Python и backtesting.py предоставляют все необходимые инструменты для таких экспериментов, открывая широкие возможности для дальнейших исследований и оптимизации.

**Автор:** Михаил Шардин

[Моя онлайн-визитка](#)

[Telegram «Умный Дом Инвестора»](#)

19 марта 2025 г.

Александр Резвяков

торговые системы

торговые роботы

трейдинг

6.4K 

 47

 69

 45



**Михаил Шардин**

 Пермь

 161  1 414

 с 23 января 2019

 +HreHDn1F5CZjN...

+ Подписаться

69 КОММЕНТАРИЕВ

Сначала старые 



**Sergey Pavlov**

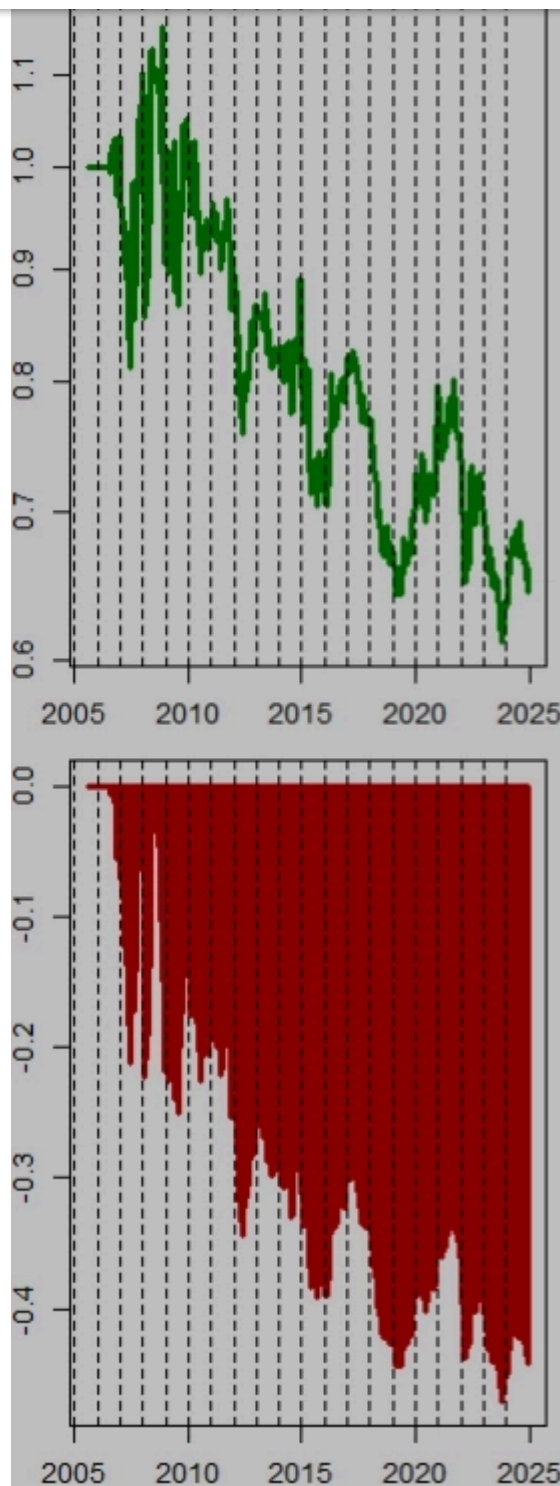
19 марта 2025, 05:48



**БИЛЕТОВ  
ВСЕ МЕНЬШЕ**



**УСПЕТЬ КУПИТЬ**



— Показать 2 ответа



Sergey Pavlov

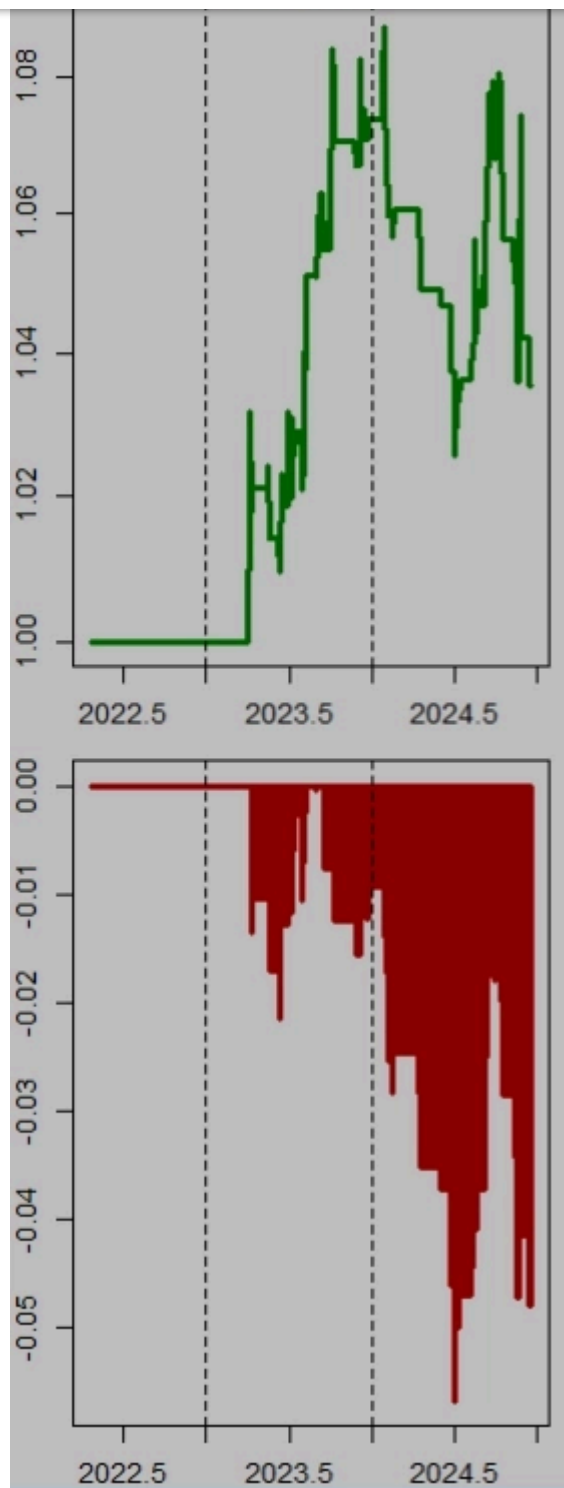
19 марта 2025, 05:49



**БИЛЕТОВ  
ВСЕ МЕНЬШЕ**

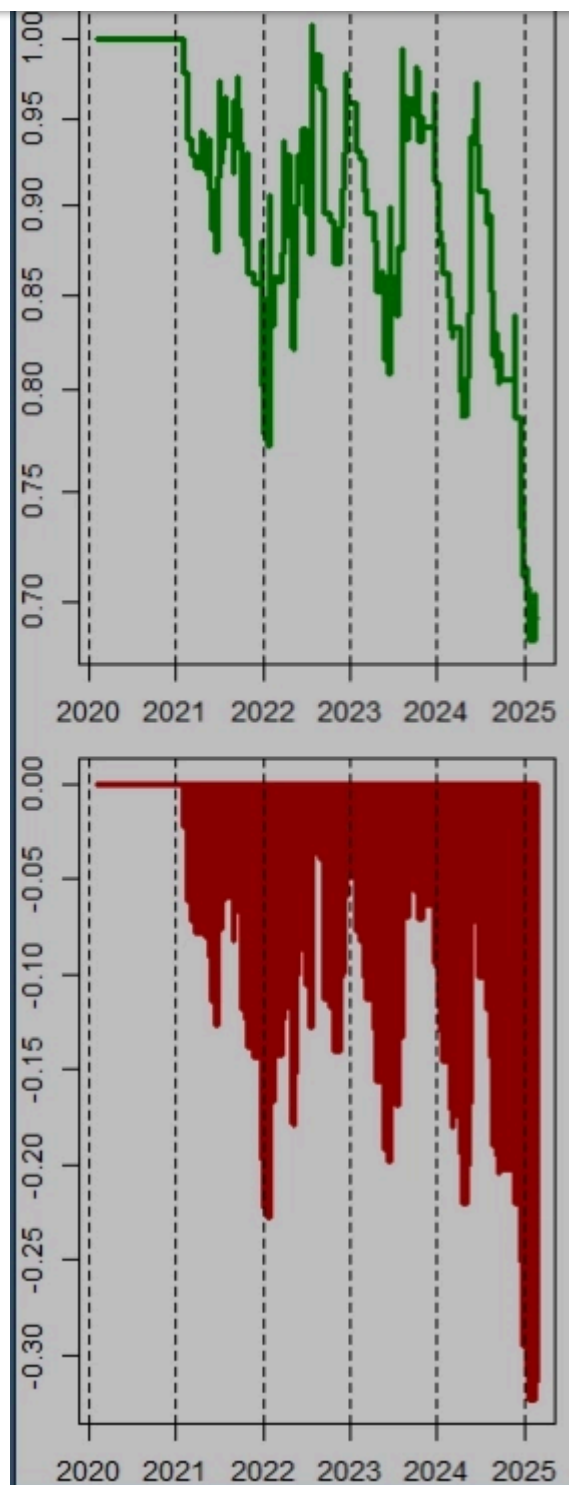


**УСПЕТЬ КУПИТЬ**

**Sergey Pavlov**

19 марта 2025, 05:50

**БИЛЕТОВ  
ВСЕ МЕНЬШЕ****УСПЕТЬ КУПИТЬ**



— Показать 9 ответов



Sergey Pavlov

19 марта 2025, 05:51



**БИЛЕТОВ  
ВСЕ МЕНЬШЕ**



**УСПЕТЬ КУПИТЬ**



откуда вот это всё взялось?

— Показать 1 ответ



Ещё 13 комментариев

Напишите комментарий...



ОТПРАВИТЬ

Установите приложение Смартлаба:



RuStore



AppGallery



App Store



[О смартлабе](#)

[Реклама](#)

[Полная версия](#)



[Московская Биржа](#) является спонсором ресурса smart-lab.ru

**БИЛЕТОВ  
ВСЕ МЕНЬШЕ**



**УСПЕТЬ КУПИТЬ**