## **Overview**

The MarketModels class provides tools for market data analysis and trading cost estimation in cryptocurrency markets. It implements several models for estimating trading costs, including slippage, market impact, maker/taker ratios, and fees.

## **Class: MarketModels**

### **Constructor**

python

def \_\_init\_\_(self, buffer\_size=100)

Initializes a new instance of the MarketModels class.

**Parameters:**

* buffer\_size (int, optional): Maximum number of data points to store in history buffers. Default is 100.

**Attributes:**

* buffer\_size: Maximum size of history buffers
* price\_history: Deque of recent price data
* spread\_history: Deque of recent spread data
* volume\_history: Deque of recent volume data
* volatility: Current volatility estimate (as percentage)

### **Methods**

#### **update\_market\_data**

python

def update\_market\_data(self, price, spread, volume)

Updates internal market data buffers with new tick data and recalculates volatility.

**Parameters:**

* price (float): Latest market price
* spread (float): Latest bid-ask spread
* volume (float): Latest trading volume

#### **estimate\_slippage**

python

def estimate\_slippage(self, quantity, side, orderbook)

Estimates expected slippage for a trade using linear regression or a fallback method.

**Parameters:**

* quantity (float): Order size in quote currency (e.g., USD)
* side (str): Order side ('buy' or 'sell')
* orderbook (dict): Order book data with 'asks' and 'bids' keys

**Returns:**

* Estimated slippage as a percentage (float)

#### **\_estimate\_basic\_slippage**

python

def \_estimate\_basic\_slippage(self, quantity, side, orderbook)

Provides a basic slippage estimation when regression isn't possible.

**Parameters:**

* quantity (float): Order size in quote currency
* side (str): Order side ('buy' or 'sell')
* orderbook (dict): Order book data

**Returns:**

* Estimated slippage as a percentage (float)

#### **calculate\_market\_impact**

python

def calculate\_market\_impact(self, quantity, side, price, volatility\_override=None)

Calculates expected market impact using a simplified Almgren-Chriss model.

**Parameters:**

* quantity (float): Order size in quote currency
* side (str): Order side ('buy' or 'sell')
* price (float): Current market price
* volatility\_override (float, optional): Override for volatility value

**Returns:**

* Estimated market impact in absolute currency value (float)

#### **predict\_maker\_taker\_ratio**

python

def predict\_maker\_taker\_ratio(self, order\_type, quantity, volatility\_override=None)

Predicts the maker/taker ratio for a given order.

**Parameters:**

* order\_type (str): Type of order ('market' or 'limit')
* quantity (float): Order size in quote currency
* volatility\_override (float, optional): Override for volatility value

**Returns:**

* Probability of the order being a maker order (0.0-1.0)

#### **calculate\_fees**

python

def calculate\_fees(self, quantity, fee\_tier, maker\_ratio=0.0)

Calculates expected trading fees based on OKX fee structure and maker/taker ratio.

**Parameters:**

* quantity (float): Order size in quote currency
* fee\_tier (str): Fee tier identifier ('1', '2', '3')
* maker\_ratio (float, optional): Portion of order expected to be maker (0.0-1.0)

**Returns:**

* Tuple of (fee\_amount, fee\_rate)

#### **calculate\_net\_cost**

python

def calculate\_net\_cost(self, quantity, slippage\_pct, market\_impact, fee\_amount, side)

Calculates the net cost of a trade including slippage, market impact, and fees.

**Parameters:**

* quantity (float): Order size in quote currency
* slippage\_pct (float): Slippage as a percentage
* market\_impact (float): Market impact in currency units
* fee\_amount (float): Fee amount in currency units
* side (str): Order side ('buy' or 'sell')

**Returns:**

* Net cost for buy orders or net proceeds for sell orders (float)

#### **update**

python

def update(self, price=None, spread=None, volume=None, reset=False)

Generic method to update internal data buffers.

**Parameters:**

* price (float, optional): New price data point
* spread (float, optional): New spread data point
* volume (float, optional): New volume data point
* reset (bool, optional): If True, clears all buffers

## **Usage Examples**

### **Basic Usage**

python

*# Initialize market models*  
market\_models = MarketModels(buffer\_size=100)  
  
*# Update with new market data*  
market\_models.update\_market\_data(price=50000.0, spread=5.0, volume=10.0)  
  
*# Estimate slippage for a buy order*  
orderbook = {  
 "asks": [["50000", "1.5"], ["50010", "2.0"], ["50020", "3.0"]],  
 "bids": [["49995", "1.2"], ["49990", "2.1"], ["49980", "2.8"]]  
}  
slippage = market\_models.estimate\_slippage(quantity=10000, side="buy", orderbook=orderbook)  
  
*# Calculate market impact*  
impact = market\_models.calculate\_market\_impact(quantity=10000, side="buy", price=50000.0)  
  
*# Predict maker/taker ratio*  
maker\_ratio = market\_models.predict\_maker\_taker\_ratio(order\_type="limit", quantity=10000)  
  
*# Calculate fees*  
fees, fee\_rate = market\_models.calculate\_fees(quantity=10000, fee\_tier="2", maker\_ratio=maker\_ratio)  
  
*# Calculate net cost*  
net\_cost = market\_models.calculate\_net\_cost(  
 quantity=10000,  
 slippage\_pct=slippage,   
 market\_impact=impact,  
 fee\_amount=fees,  
 side="buy"  
)

### **Comprehensive Trade Cost Analysis**

python

def analyze\_trade\_costs(market\_models, quantity, side, orderbook, price, fee\_tier):  
 """Analyze all components of trading costs for a potential trade."""  
   
 *# Calculate slippage*  
 slippage\_pct = market\_models.estimate\_slippage(quantity, side, orderbook)  
   
 *# Calculate market impact*  
 market\_impact = market\_models.calculate\_market\_impact(quantity, side, price)  
   
 *# Predict maker/taker ratio for limit orders (0 for market orders)*  
 maker\_ratio = market\_models.predict\_maker\_taker\_ratio("limit", quantity)  
   
 *# Calculate fees*  
 fees, fee\_rate = market\_models.calculate\_fees(quantity, fee\_tier, maker\_ratio)  
   
 *# Calculate net cost*  
 net\_cost = market\_models.calculate\_net\_cost(  
 quantity, slippage\_pct, market\_impact, fees, side  
 )  
   
 *# Return comprehensive analysis*  
 return {  
 "slippage\_pct": slippage\_pct,  
 "slippage\_amount": quantity \* (slippage\_pct / 100),  
 "market\_impact": market\_impact,  
 "maker\_ratio": maker\_ratio,  
 "fees": fees,  
 "fee\_rate": fee\_rate,  
 "net\_cost": net\_cost  
 }

## **Technical Notes**

* The class uses deque from collections to efficiently manage fixed-size buffers
* Linear regression is used for slippage estimation when sufficient data is available
* The Almgren-Chriss model is used for market impact estimation
* Logistic regression is used to predict maker/taker ratios for limit orders
* Fee calculations are based on the OKX fee structure