Bid Curve Toolkit (Python)
some explanation

August 9, 2025

Contents

1 Overview

This library loads Japanese power market bid curves from row-wise data, builds cumulative supply/demand curves on a common price grid, and provides:

- market-clearing metrics (clearing_price, clearing_demand),

- diagnostics per time slot (`timeslot_summary`),
- trading cost and marginal price impact (`trading_cost`, `marginal_price`),
- simple plotting helpers.

Core containers are `BidCurve` (single day) and `MultiBidCurve` (many timestamps).

Requirements Python 3.10+, `numpy`, `pandas`, `matplotlib`. Data columns expected (customizable via loader args): `area`, `dt`, `order` (`buy`/`sell`), `jpy_kwh`, `volume_mw`, optional `group`.

2 Quick Start

Listing 1: Load, compute, plot

```python
from jp_da_imb.trading_costs.bid_curve_loader import load_curves_rows
from jp_da_imb.trading_costs.bid_curve_stats import clearing_price, timeslot_summary
from jp_da_imb.trading_costs.trading_cost_estimation import trading_cost
from jp_da_imb.trading_costs.bid_curve_plotting import plot_supply_demand

curve = load_curves_rows(
    path="path/to/jepx_bid_curve.parquet",
    area="tokyo", price_step=0.01, tz="Asia/Tokyo",
    vol_col="volume_mw", price_col="jpy_kwh",
    order_col="order", area_col="area", dt_col="dt",
)

ts = "2023-10-01 02:30:00+09:00"
slice_df = curve.slice_time(ts)

print("Clearing price:", clearing_price(slice_df))
print(timeslot_summary(curve).head())

fig = plot_supply_demand(curve, ts=ts, ylim=60)
fig.savefig("supply_demand.png", dpi=150)
```

3 Data Structures

3.1 BidCurve

Represents one trading day (48 half-hour slots) for a region.

- **Attributes**: `region`, `date`, `bins` (price grid), `supply`, `demand`, `df_raw`.
- `slice_time(time_code:int)` → `DataFrame` (index=price, columns=`supply_cum`, `demand_cum`).
- `to_long()` → long table with columns `date, region, time_code, side, price, cum_vol`.

3.2 MultiBidCurve

Panel over many timestamps (index are exact 30-min `Timestamps`).

- **Attributes**: `region`, `bins`, `supply`, `demand`, `groups` (optional group-ID per timestamp), `df_raw`.
- `slice_time(ts)` → curve at timestamp.
- `slice_day(date)` → `BidCurve`.
- `to_long()` → long table with `date, region, time_code, timestamp, side, price, cum_vol`.
- Indexing: `obj[ts]` is sugar for `slice_time(ts)`; iteration yields timestamps.

4 Loading & Aggregation (`bid_curve_loader.py`)

4.1 load_curves_rows(path, area, price_step=0.01, tz=...)

Read a row-wise file and return a MultiBidCurve for one `area`.

- **Builds**: a common ascending price grid across all slices; cumulative arrays for `supply_cum` and `demand_cum`.
- **Returns**: MultiBidCurve.
- **Raises**: `ValueError` if columns missing or area not found.

4.2 group_slices_from_parquet(parquet_path, areas, price_step, tz)

Load multiple areas and aggregate *per timestamp* by `group`-ID.

- **Returns**: `(time_dict, all_ts)` where `time_dict[ts]` is a list of `(combined_name, slice_df)`.
- **Note**: ensures identical price grids across areas; resample first if they differ.

5 Clearing & Diagnostics (`bid_curve_stats.py`)

5.1 clearing_price(slice_df) clearing_demand(slice_df)

Market-clearing intersection of curves via linear interpolation inside the first crossing bin. Demand ≡ Supply at the clearing point (alias clearing_supply).

5.2 residual_volume(slice_df, price_threshold, side)

Remaining cumulative volume above a price threshold. `side` in {`"supply"`, `"demand"`}.

5.3 imbalance(slice_df, integrated=False)

Supply minus demand at each price. If `integrated=True`, returns trapezoidal area (MWh · ¥).

5.4 iter_slices(container)

Yields `(label, slice_df)` over a BidCurve (labels 1–48) or MultiBidCurve (timestamp labels).

5.5 clearing_series(container, return_dataframe=True)

Vectorised clearing price/volume across all slices. Helpers: clearing_price_series, clearing_volume_series.

5.6 timeslot_summary(container, vwap_side="supply")

Per-slot diagnostics:

- `clearing_price` [¥/kWh], `clearing_volume` [MWh],
- `vwap` on selected side using incremental volumes,
- `price_min`, `price_max` active bins,
- `imbalance_integral` (area between curves).

6 Trading Cost & Impact (`trading_cost_estimation.py`)

6.1 trading_cost(slice_df, vol_mwh, side)

Uniform-price auction cost to buy/sell `vol_mwh`. Returns `(total_yen, clearing_price)`. Validates available volume.

6.2 trading_cost_series(container, vol_mwh, side)

Vectorised total cost and clearing price per slot.

6.3 marginal_price(slice_df, qty_mwh, side)

Shift demand/supply by `qty_mwh`, recompute clearing, and return `(old_cp, new_cp, delta_price, extra_cost)` where `extra_cost = delta_price × qty_mwh × 1000` [¥].

6.4 marginal_price_series(container, qty_mwh, side, metric)

Vectorised `delta_price` [¥/kWh] and `extra_cost` [¥] across slots; returns `DataFrame` (or a single `Series` when requested).

7 Plotting (`bid_curve_plotting.py`)

Lightweight, Matplotlib-based helpers that accept either a BidCurve slice or a MultiBidCurve+timestamp.

- plot_supply_demand(container, ts, ylim=None, xlow=None, xhigh=None)
- plot_trading_cost_series(container, vol_mwh, side, metric="total_cost")
- plot_metric_series(container, metric="clearing_price", vwap_side="supply")
- plot_marginal_price(container, qty_mwh, side, metric)

8 Examples

Aggregate multiple regions by group-ID

```
areas = ["chubu","chugoku","hokkaido","hokuriku","kansai",
        "kyushu","shikoku","tohoku","tokyo"]
time_dict, all_ts = group_slices_from_parquet(
    parquet_path="path/to/jepx_bid_curve.parquet",
    areas=areas, price_step=0.01, tz="Asia/Tokyo"
)

ts = all_ts[0]
name, slice_df = time_dict[ts][0] # first combined group at ts
print(name, clearing_price(slice_df))
```

9 Design Notes

- All slices share one exact price grid to avoid FP jitter.
- Cumulative arrays are filled monotone: supply increases with price; demand decreases with price.
- VWAP uses incremental volumes derived from cumulative curves.

10 API Index

Module	Functions / Classes
bid_curve_struc	BidCurve, MultiBidCurve
bid_curve_loader	load_curves_rows, group_slices_from_parquet
bid_curve_stats	clearing_price, clearing_demand, residual_volume, imbalance, timeslot_summary, clearing_series
trading_cost_estimation	trading_cost, trading_cost_series, marginal_price, marginal_price_series
bid_curve_plotting	plot_supply_demand, plot_trading_cost_series, plot_metric_series, plot_marginal_price