Loan Trader Demonstration

November 13, 2023

1 Loan Trader Demonstration

[1]:
```
import Agents.Loan as Loan
import Agents.LoanInvestor as LoanInvestor
import Agents.LoanTrader as LoanTrader
import matplotlib.pyplot as plt
```

2 Setting up a universe of loans and investors

The general goal here will be to demonstrate the process for a trade to clear step by step with a loan that the trader is holding for sale.

2.0.1 Creating the setup for loans to start trading

We will start with a universe in which there are a number of investors and a fewer number of loans

[2]:
```
num_loans = 50
num_investors = 10


# setting up a universe of loans and investors and randomly allocating them
loans_list = [Loan.Loan() for _ in range(num_loans)]
investors_list = [LoanInvestor.LoanInvestor() for _ in range(num_investors)]
trader = LoanTrader.LoanTrader()

# randomly allocating loans to investors
for investor in investors_list:
    available_loans = [loan for loan in loans_list if loan.current_owner == "no␣
 ↪owner"]
    investor.initialize_portfolio(available_loans)

# setting up all the investors to have our trader as their trader
trader.add_investors(investors_list)

loans_for_sale = [loan for loan in loans_list if loan.current_owner == "no␣
 ↪owner"]
loans_for_sale.extend([Loan.Loan() for _ in range(1)])
```

```
trader.update_loans_for_sale(loans_for_sale)
```

[3]:
```
for investor in investors_list:
    investor.update(cycle = 1)

trader.run_auction(show_bids=True)
```

```
Investor I1366 bids 83.91370163552608 for loan 618bd
Investor Ie244 bids 87.47236182751814 for loan 618bd
Investor I4b0a bids 82.55874774340819 for loan 618bd
Investor I7443 bids 81.98785918539997 for loan 618bd
Investor I5ab4 bids 84.43258652601018 for loan 618bd
Investor I4cec bids 64.00098942882649 for loan 618bd
Investor I7b3e bids 42.71014783684683 for loan 618bd
Investor Ifa80 bids 81.29626476939734 for loan 618bd
Investor Ie8db bids 85.52519263141284 for loan 618bd
Investor I390a bids 76.07487119439195 for loan 618bd
Purchased:  True
Top bidder is Ie244 with bid price 87.47236182751814 for $3855251.952378224
```

3 Looking at the Last Loan For Sale

[4]:
```
loans_for_sale[-1].as_dict()
```

[4]:
```
{'id': '618bd385-5af7-4520-9ede-8c595bdbea8f',
 'maturity': 84,
 'current_cycle': 0,
 'starting_cycle': 0,
 'ending_cycle': 84,
 'time_to_maturity': 84,
 'pd': 0.07657589327611096,
 'size': 4407394.372156293,
 'interest_rate': 0.006667256728879449,
 'fair_value': 98.85749308835244,
 'market_price': 87.47236182751814,
 'current_owner': <Agents.LoanInvestor.LoanInvestor at 0x137d0b790>,
 'maturity_bool': False,
 'fair_value_history': [98.85749308835244],
 'market_price_history': [98.85749308835244, 87.47236182751814],
 'ownership_history': ['no owner',
  'Td6c68e6e-1fbf-41ea-88a7-560dccb479fa',
  'Ie244cb8c-2026-4e9b-9395-3801478f12f1'],
 'sale_price_history': [None, 87.47236182751814],
 'reserve_price': 79.08599447068195}
```

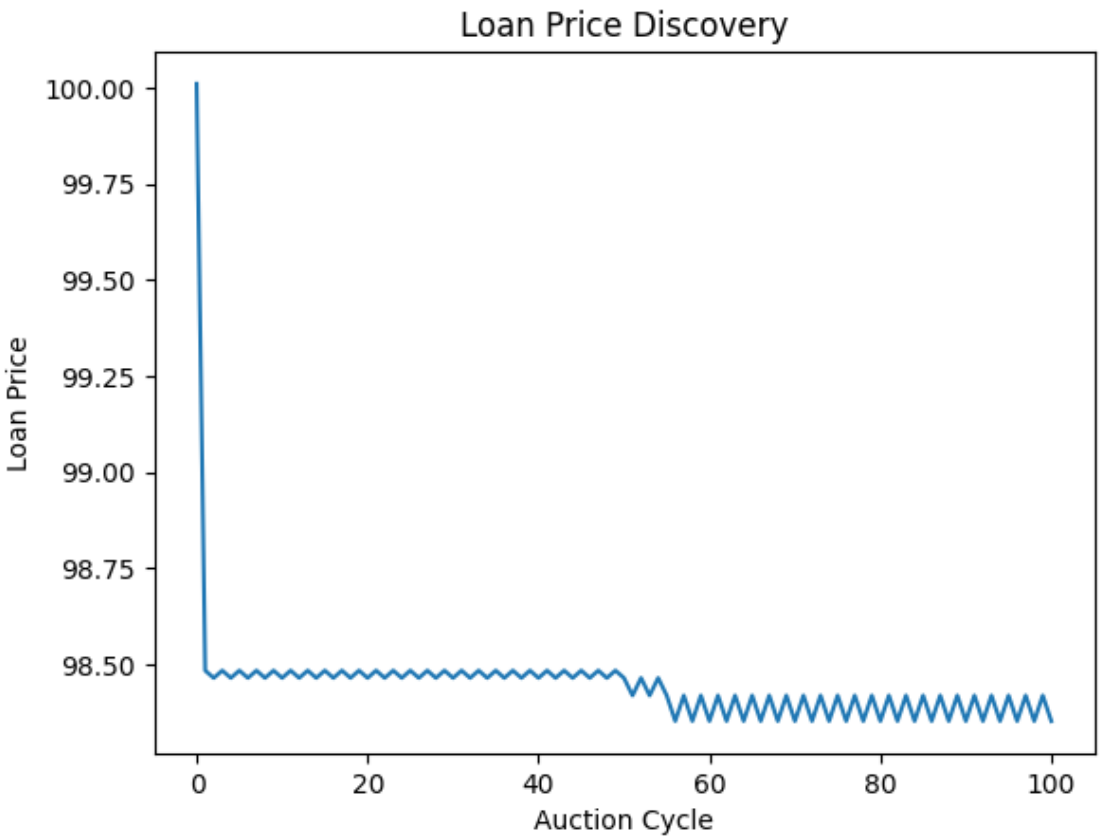
3.1 Trading Loans Between Investors

The trader will have the ability to collect the loans for sale from investors and then run an auction to sell the loans. We'll use a similar set up from above and a limited pool of investors as a demonstration.

[5]:
```
num_loans = 1
num_investors = 500
trader = LoanTrader.LoanTrader(max_investors=500)


# setting up a universe of loans and investors and randomly allocating them
loans_list = [Loan.Loan() for _ in range(num_loans)]
investors_list = [LoanInvestor.LoanInvestor() for _ in range(num_investors)]

# randomly allocating loans to investors
for investor in investors_list:
    available_loans = [loan for loan in loans_list if loan.current_owner == "no␣
 ↪owner"]
    investor.initialize_portfolio(available_loans)

# setting up all the investors to have our trader as their trader
trader.add_investors(investors_list)
```

[6]:
```
for _ in range(100):
    trader.collect_loans_for_sale(print_outputs=False, num_investors = 500)
    trader.run_auction(show_bids=False)
```

[7]:
```
# plot to show loan price discovery in this universe of investors
plt.plot([loan.market_price_history for loan in loans_list][0])
plt.title("Loan Price Discovery")
plt.xlabel("Auction Cycle")
plt.ylabel("Loan Price")
plt.show()
```



From the plot above, we can see how there is eventially an ideal price that the loan will eventually reach after getting auctioned a number of times to match the most ideal investor for that particular loan based on it's score and the max bid. The idea of a reserve price allows this disovery to happen in a way that is not too costly for the originaly seller creating this slow decline type effect.

3.2 Adding in Multiple Loans

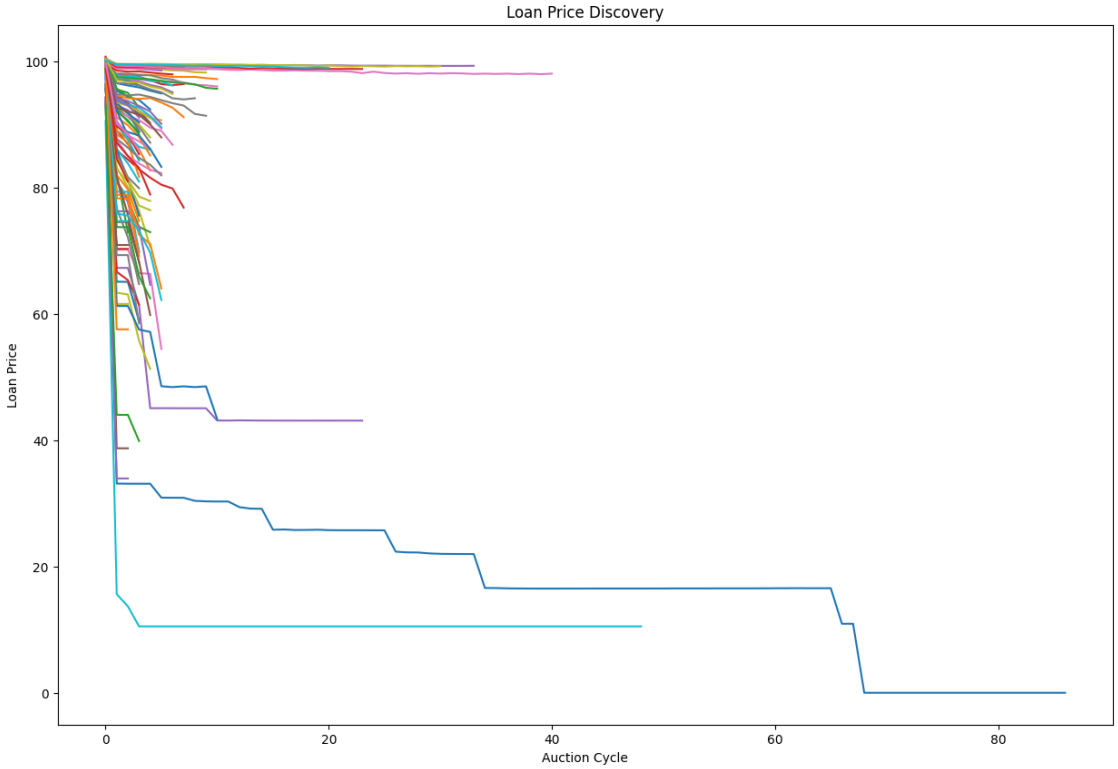
Below we see how price discovery becomes more complex for an individual loan when investors have a built portfolio. This is all under the assumption of a release at the initial cycle where no loans are updated until maturity.

[8]:
```
num_loans = 100
num_investors = 25
trader = LoanTrader.LoanTrader(max_investors=500)


# setting up a universe of loans and investors and randomly allocating them
loans_list = [Loan.Loan() for _ in range(num_loans)]
investors_list = [LoanInvestor.LoanInvestor() for _ in range(num_investors)]

# randomly allocating loans to investors
```

```
for investor in investors_list:
    available_loans = [loan for loan in loans_list if loan.current_owner == "no␣
 ↪owner"]
    investor.initialize_portfolio(available_loans)

# setting up all the investors to have our trader as their trader
trader.add_investors(investors_list)
```

[9]:
```
for _ in range(100):
    trader.collect_loans_for_sale(print_outputs=False, num_investors = 25)
    trader.run_auction(show_bids=False)
```

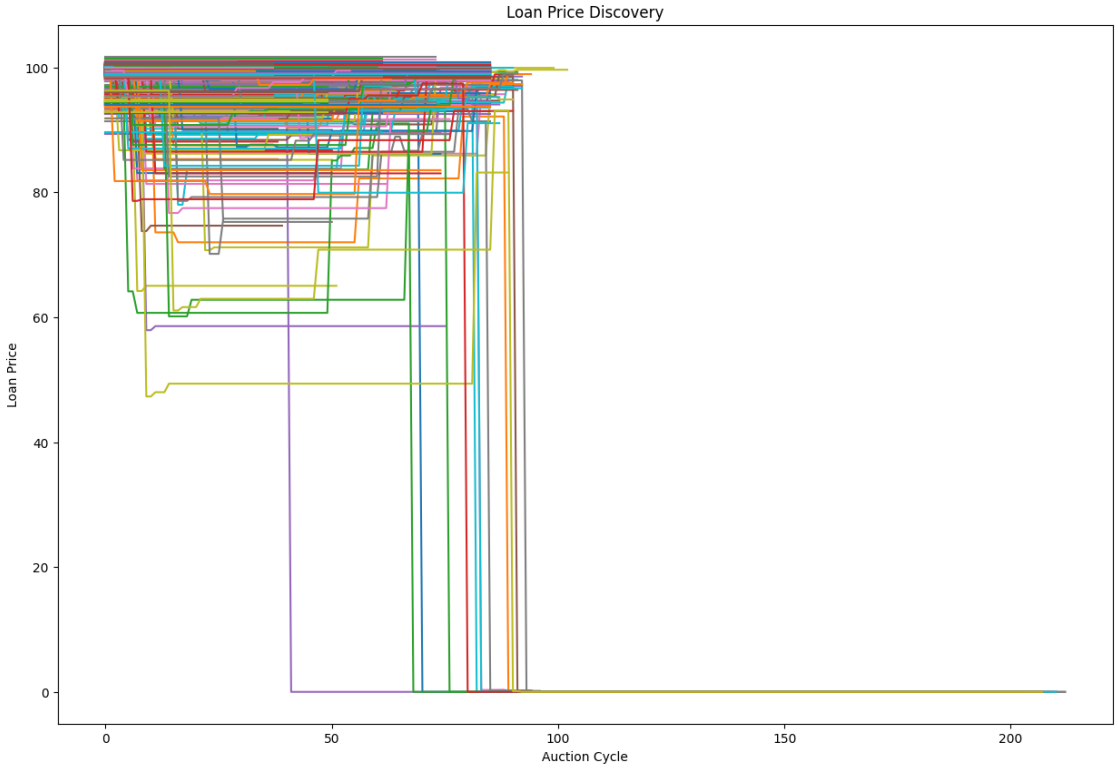
[10]:
```
# Plotting
plt.figure(figsize=(15, 10))

# Plot each sublist as a separate line in the line plot
for i, sublist in enumerate([loan.market_price_history for loan in loans_list]):
    plt.plot(sublist, label=f'Series {i+1}')

plt.title("Loan Price Discovery")
plt.xlabel("Auction Cycle")
plt.ylabel("Loan Price")
plt.show()
```



3.2.1 Adding in Loan Updates

This is the most basic implementation of the entire loan market in which prices are discovered and loans are able to mature. In this case there is only one trader and 1000 loans across 25 investors.

[11]:
```
num_loans = 1000
num_investors = 25
trader = LoanTrader.LoanTrader(max_investors=500)

# setting up a universe of loans and investors and randomly allocating them
loans_list = [Loan.Loan() for _ in range(num_loans)]
investors_list = [LoanInvestor.LoanInvestor() for _ in range(num_investors)]

# randomly allocating loans to investors
for investor in investors_list:
    available_loans = [loan for loan in loans_list if loan.current_owner == "no␣
 ↪owner"]
    investor.initialize_portfolio(available_loans)

# setting up all the investors to have our trader as their trader
trader.add_investors(investors_list)
for _ in range(200):
    for investor in investors_list:
        investor.update(cycle=_)
    for loan in loans_list:
        loan.update(current_cycle=_)
    trader.collect_loans_for_sale(print_outputs=False, num_investors=25)
    trader.run_auction(show_bids=False)
# Plotting
plt.figure(figsize=(15, 10))

# Plot each sublist as a separate line in the line plot
for i, sublist in enumerate([loan.market_price_history for loan in loans_list]):
    plt.plot(sublist, label=f'Series {i + 1}')

plt.title("Loan Price Discovery")
plt.xlabel("Auction Cycle")
plt.ylabel("Loan Price")
plt.show()
```



[12]:
```
# TO DO: Matured Loans are sneaking into the auction process
# The matured loans are coming from traders, so the money they have is not␣
 ↪updating when the loans mature. Also the loans for sale are not being␣
 ↪appropriately purged.
# Update the loan market class to comprehensively run the simulation.
```