Assignment 2 Report

Fuzzy logic in trading securities

Tanat Tangun 630610737

September 2022

This report is about the result of my fuzzy logic implementation on Rust language for 261456 - INTRO COMP INTEL FOR CPE class assignment. This report will be about how fuzzy logic can help in trading securities and other assets. If you are interested to know how I implement the fuzzy logic and use it to help in trading securities and other assets , you can see the source code on my Github repository or in this document appendix.

Introduction

The term "securities and other assets" in our report include securities such as stock, bonds, currencies, and other assets i.e. crypto-currency, house price, etc. In this report, we will focus on crypto-currency such as Ethereum. Before we delve further, let's understand that we are not trying to create more profit than any other techniques but we are trying to learn how to create fuzzy logic and explore how it might be useful for trading.

Technical Indicators

According to [1], "Technical indicators are heuristic or pattern-based signals produced by the price, volume, and/or open interest of a security or contract used by traders who follow technical analysis. By analyzing historical data, technical analysts use indicators to predict future price movements."

We are trying to use technical indicators to help decide when to entry or exit to make a profit trade. Techinal indicators that we will use are RSI (Relative Strength Index) and Bollinger Bands.

RSI - Relative Strength Index

According to [2] "The relative strength index (RSI) is a momentum indicator used in technical analysis. RSI measures the speed and magnitude of a security's recent price changes to evaluate overvalued or undervalued conditions in the price of that security." RSI is a value in range $[0, 100]$ and the value at time t is defined as [4] shown:

$$\text{RSI} = 100 - \frac{100}{1 + \text{RS}}$$

where RS is the relative strength of the last n sessions and its defined as:

$$\text{RS} = \frac{\text{AverageGain}_t(n)}{\text{AverageLoss}_t(n)}$$

where $\text{AverageGain}_t(n)$ and $\text{AverageLoss}_t(n)$ are the average of the gains ($\text{price}_t > \text{price}_{t-1}$) and losses ($\text{price}_t < \text{price}_{t-1}$), obtained in the last n sessions. That is, from time $t - (n - 1)$ to time t. However, theses values are usually from estimated using the following smoothing equations:

$$\text{AverageGain}_t(n) = \frac{\text{AverageGain}_{t-1}(n) \cdot (n - 1) + \text{gain}_t}{n}$$

$$\text{AverageLoss}_t(n) = \frac{\text{AverageLoss}_{t-1}(n) \cdot (n - 1) + \text{loss}_t}{n}$$

If a session t result in gain then $\text{loss}_t = 0$ and, if results in loss then $\text{gain}_t = 0$. Commom number of sessions are $n = 14$ and a common interpretation of the RSI index is that it suggests oversold at value < 30, and overbought for value > 70

Bollinger Bands

According to [3], "A Bollinger Band is a technical analysis tool defined by a set of trendlines plotted two standard deviations (positively and negatively) away from a simple moving average (SMA) of a security's price." Bollinger Bands (BOLU for upper band, BOLD for lower band) at time t are defined as:

$$\text{BOLU} = \text{MA}(n) + m\sigma(n)$$

$$\text{BOLD} = \text{MA}(n) - m\sigma(n)$$

where m is number of standard deviations (usually 2), and both $\text{MA}(n)$ and $\sigma(n)$ of the last n sessions are defined as:

$$\text{MA}(n) = \frac{\sum_{i=1}^{n} p_i}{n}$$

$$\sigma(n) = \sqrt{\frac{\sum_{i=1}^{n} (p_i - \text{MA}(n))^2}{n}}$$

where each p_i is a typical price calulated as $p_i = \frac{(\text{high}_i + \text{low}_i + \text{close}_i)}{3}$

Common number of sessions are $n = 20$ and a common interpretation is the closer the prices move to BOLU, the more overbought the market, and the closer the prices move to the lower band, the more oversold the market. For using in our fuzzy logic, we will use BOLU and BOLD as a 100% mark from MA and we will compute the diffrence of price from MA to use it e.g. price = 100, MA = 89, and $\sigma = 10$ then we calculate $100 \times \frac{\text{price} - \text{MA}}{2\sigma} = 55$ which we will use on our fuzzy logic.

Entry rules

Entry rule tell you that at a current time, should you open a position or not? and it is what we are going to make with fuzzy logic. For clarifying, a position in this report will be in these 2 types:

1. LONG, which we gain profit from price increasing.
2. SHORT, which we gain profit from price decreasing.

Classic rules

The classic rules is based on set of fixed rules where the input are exact number and the outputs are binary values (1 - yes, 0 - no). The condition is from the trader's belief which should have some uncertainty, but the classic rules can't include that uncertainty in the rules. An examples of classic rules can be seen on table 1 and table 2 which could be more complex and more practical by adding more "useful" indicators. In those examples, on the Bollinger Bands column we will use $\beta = \frac{\text{price} - \text{MA}}{2\sigma}$ to determine how close is the price to BOLU and BOLD.

	RSI		Bollinger Bands		LONG
If	< 30	&	$\beta < -0.9$	then	1
elseif	< 30	&	$\beta \geq -0.9 \,\&\, \beta < 0.0$	then	1
else				then	0

Table 1: Examples for classic rule (LONG signal).

	RSI		Bollinger Bands		SHORT
If	> 70	&	$\beta > 0.9$	then	1
elseif	> 70	&	$\beta \leq 0.9 \,\&\, \beta > 0.0$	then	1
else				then	0

Table 2: Examples for classic rule (SHORT signal).

Fuzzy rules

Fuzzy rule can tolerate uncertainty better than classic rule due to its nature. So, we can use imprecise linguistic terms on trading indicator e.g. rsi → HIGH, MEDIUM, LOW to help us decide when to entry. Thus, a trader can create entry rule which reflect on trader's belief or represent expert's belief on market behaviour that is vague and often lack certainty.

In this report, techincal indicators and output conditions have been fuzzified as shown on fig. 1 and below is an explanation for each fuzzy set.

- RSI linguistic variable is descriped by 3 fuzzy sets:
  - LOW: indicating oversold situation, favor long position.
  - MEDIUM: not favor any position.
  - HIGH: indicating overbougth situation, favor short position.
- Bollinger Bands linguistic variable is described by 3 fuzzy sets:
  - LONG: indicating oversold situation, favor long position.
  - WAIT: not favor any position.
  - SHORT: indicating oversold situation, favor short position.
- Long, Short linguistic variable is described by 3 fuzzy sets:
  - WEAK: we should not enter the market
  - STRONG & VERYSTRONG: we should enter the market but with different level of confidence.

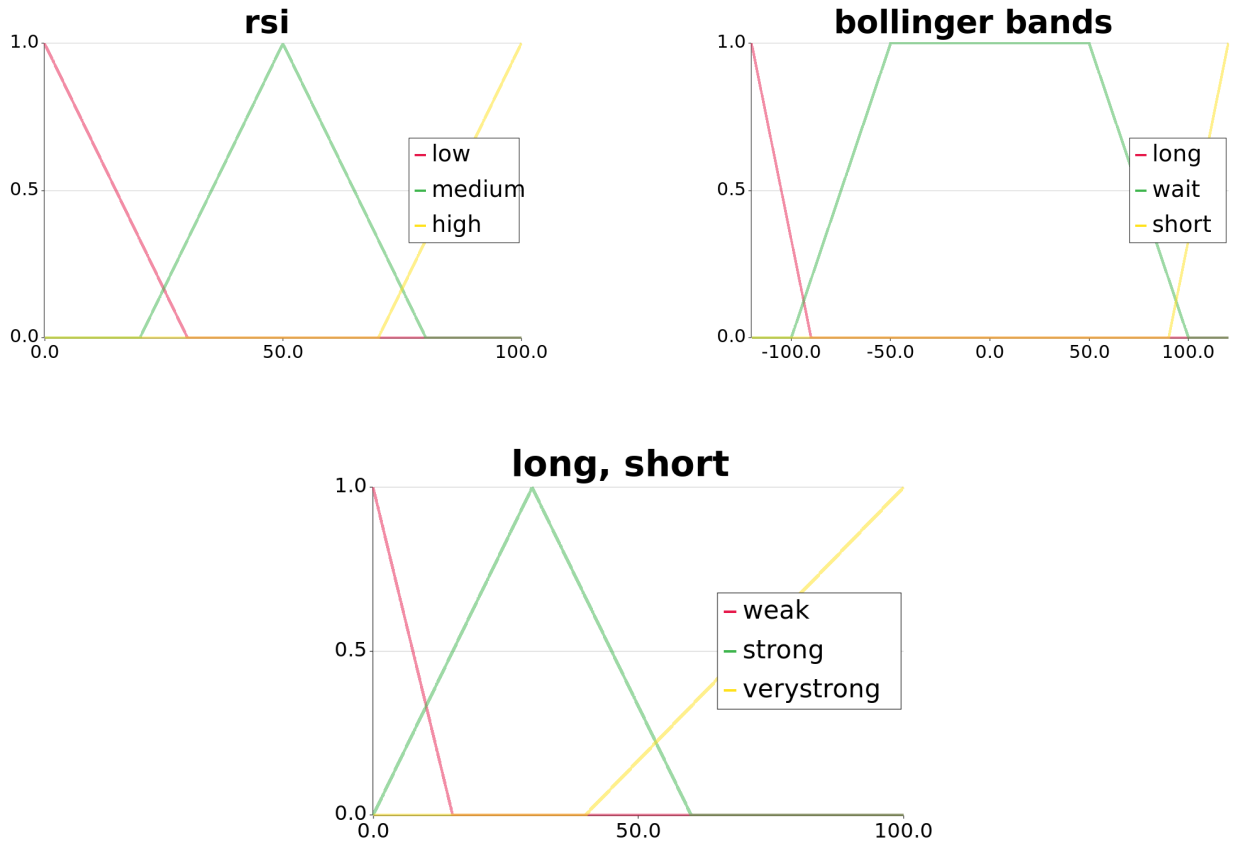

Figure 1: Fuzzifications of the indicators and outputs

Follwing the common interpretation of both RSI and Bollinger Bands, we create fuzzy rules to match with those interpretation as shown on table 3. A Mamdani fuzzy system and centroid defuzzification is used in building the rules.

RSI	Bollinger Bands	LONG	SHORT
HIGH	LONG	WEAK	WEAK
HIGH	WAIT	WEAK	STRONG
HIGH	SHORT	WEAK	VERYSTRONG
MEDIUM	LONG	WEAK	STRONG
MEDIUM	WAIT	WEAK	WEAK
MEDIUM	SHORT	STRONG	WEAK
LOW	LONG	VERYSTRONG	WEAK
LOW	WAIT	STRONG	WEAK
LOW	SHORT	WEAK	WEAK

Table 3: Fuzzy rules.

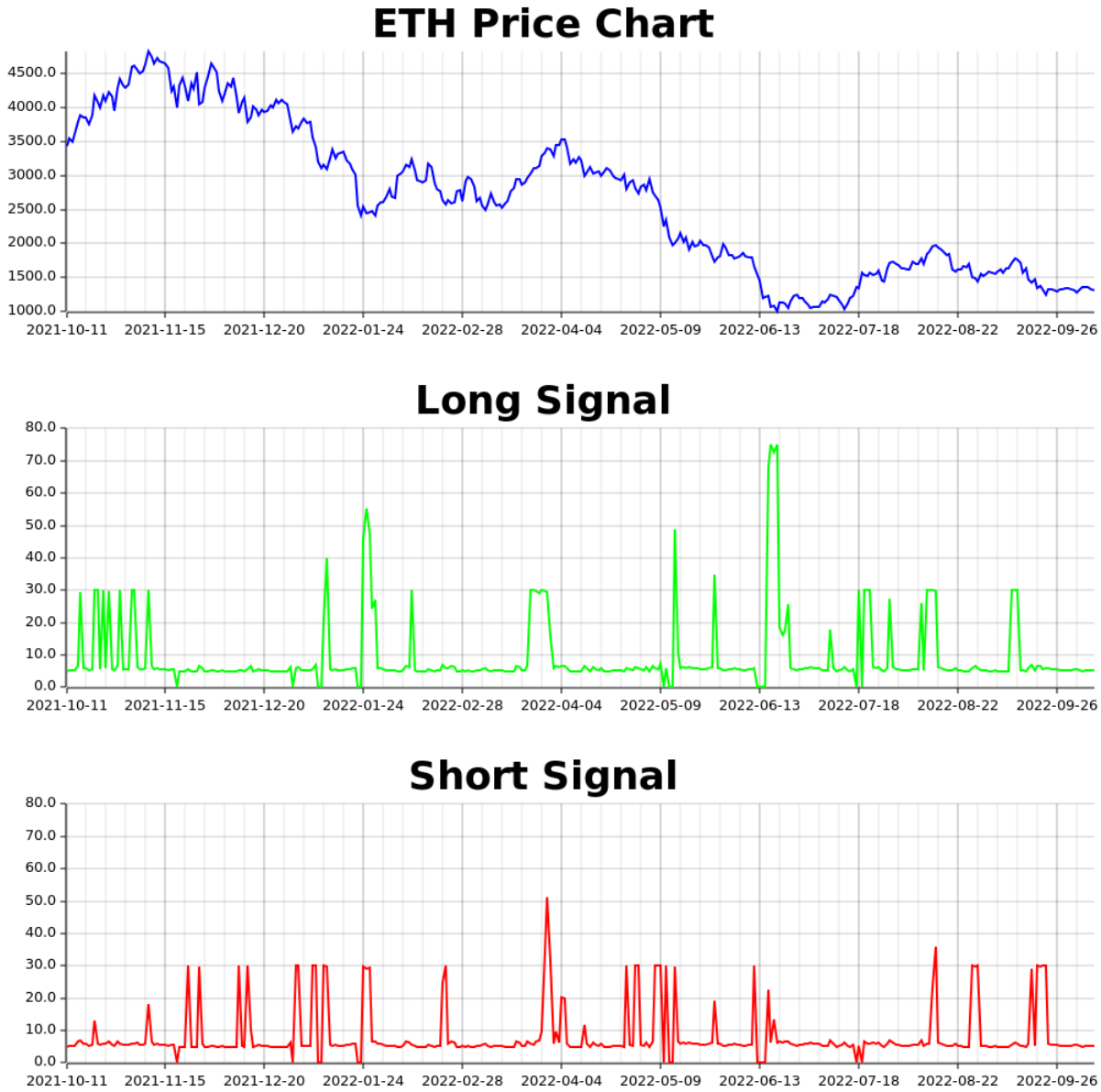

Figure 2: Result of fuzzy rules apply on each day from 2021-10-11 to 2022-10-09 where ETH price chart is the price in USD (obtained from coingecko), long signal is a defuzzification result of output LONG, and short signal is a defuzzification result of output SHORT.

Experiment & Analysis

We get ETH price data from coingecko, and apply our fuzzy rules using RSI and Bollinger Bands as inputs as we described in Technical Indicators on each day from 2022-10-11 to 2022-10-09. The result can be seen on fig. 2 which we can see the spikes on both long and short signal graph, those spikes is what we will be interest on as an entry point. Next, we set up backtesting (the general method for seeing how well a strategy or model would have done) for both classic and fuzzy rules with these rules:

1. Initial capital is 1000$ and we are not reinvesting any trade profit/loss into capital.
2. We iterate over all data from 2022-10-11 to 2022-10-09 and entry a position of 100$ following these conditions:
    - For fuzzy rules, if either long or short signal is greater than 40 we enter that position.
    - For classisc rule, we use the exact same rules as shown on table 1 and table 2
3. For each position, we take profit if the price difference (profit side) is greater than 20% and stop-loss if the price difference (loss side) is greater than 10%

The backtesting result is shown on table 4, which we can see that if we combine both fuzzy long and short signals we get more profit than both classic long and short signals combined. But, this might be from my poor design of classic rules examples however classic rules can't provide us with the signal graph that is intuitive like fuzzy rules (as shown on fig. 2) because the results of classic rules can be only 1 or 0. The fuzzy signal graph can visualize how likely you shoukd enter a market e.g. the big spike on long signal, we can consider entering a position with more confidence because of it (which is likely to make us a profit as we can see that the price is getting up considerably higher).

Signal used	Total trade	Profit trade	Loss trade	Total profit ($)	Total loss ($)	Net profit ($)
Fuzzy Long	9	6	3	139.26	-40.132	99.131
Fuzzy Short	1	1	0	20.624	0.000	20.624
Classic Long	10	5	5	111.541	-85.499	26.043
Classic Short	3	3	0	61.244	0.000	61.244

Table 4: Backtesting result.

Conclusion

Fuzzy logic can be useful for trading securities because of its ability to deal with uncertainty and vagueness we can use more than just for entry rules for example we can use it for exit rules, setting a position size, etc. And the ability to visualize how fuzzy rules result in the signal graph can be a huge difference when deciding to enter a market. Furthermore, we can add more linguistic variables which we tailor to our likes or more rules to make fuzzy rules more practical and useful.

References

[1] James Chen. *Technical Indicator*. URL: https://www.investopedia.com/terms/t/technicalindicator.asp#:~:text=What%20Is%20a%20Technical%20Indicator,to%20predict%20future%20price%20movements.. (accessed: 02.10.2022).

[2] Jason Fernado. *Relative Strength Index (RSI) Indicator Explained With Formula*. URL: https://www.investopedia.com/terms/r/rsi.asp. (accessed: 02.10.2022).

[3] Adam Hayes. *Bollinger Bands®: Calculations and Indications*. URL: https://www.investopedia.com/terms/b/bollingerbands.asp. (accessed: 09.10.2022).

[4] Rodrigo Naranjo et al. *An Intelligent Trading System with Fuzzy Rules and Fuzzy Capital Management*. 2015. URL: https://eprints.ucm.es/id/eprint/30730/1/JIS_FuzzyTradingSystem.pdf.

Appendix

main.rs

```rust
pub mod backtest;
pub mod data;
pub mod rule;
pub mod set;
pub mod shape;

use chrono::offset::{Local, TimeZone};
use chrono::Date;
use data::read_csv;
use plotters::prelude::*;
use rule::FuzzyEngine;
use set::{arange, LinguisticVar};
use shape::{trapezoidal, triangular};
use std::error::Error;

fn parse_time(t: &str) -> Date<Local> {
    Local
        .datetime_from_str(t, "%Y-%m-%d %H:%M:%S %Z")
        .unwrap()
        .date()
}

fn max_of_vec(vec: &Vec<f64>) -> f64 {
    vec.iter().fold(f64::NAN, |max, &val| val.max(max))
}

fn min_of_vec(vec: &Vec<f64>) -> f64 {
    vec.iter().fold(f64::NAN, |min, &val| val.min(min))
}

fn plot(date: &Vec<Date<Local>>, data: [&Vec<f64>; 3], path: &str) -> Result<(), Box<dyn Error>> {
    let root = SVGBackend::new(path, (1024, 1024)).into_drawing_area();
    root.fill(&WHITE)?;
    let area = root.split_evenly((3, 1));

    let colors = [&BLUE, &GREEN, &RED];
    let names = ["ETH Price Chart", "Long Signal", "Short Signal"];
    let limits = [
        (min_of_vec(&data[0]), max_of_vec(&data[0])),
        (0.0, 80.0),
        (0.0, 80.0),
    ];
    for (i, draw_area) in area.iter().enumerate() {
        let mut chart = ChartBuilder::on(&draw_area)
            .caption(names[i], ("Hack", 44, FontStyle::Bold).into_font())
            .set_label_area_size(LabelAreaPosition::Left, 70)
            .set_label_area_size(LabelAreaPosition::Bottom, 60)
            .margin_right(20)
            .build_cartesian_2d(date[0]..date[date.len() - 1], limits[i].0..limits[i].1)?;

        chart
            .configure_mesh()
            .y_max_light_lines(0)
            .x_labels(11)
            .x_label_formatter(&|v| format!("{}", v.format("%Y-%m-%d")))
```

```
            .x_label_style(("Hack", 16).into_font())
            .y_label_style(("Hack", 16).into_font())
            .draw()?;

        chart.draw_series(LineSeries::new(
            date.iter().zip(data[i].iter()).map(|(d, p)| (*d, *p)),
            colors[i].stroke_width(2),
        ))?;
    }
    root.present()?;
    Ok(())
}

fn main() -> Result<(), Box<dyn std::error::Error>> {
    let rsi_var = LinguisticVar::new(
        vec![
            (&triangular(0f64, 1.0, 30f64), "low"),
            (&triangular(50f64, 1.0, 30f64), "medium"),
            (&triangular(100f64, 1.0, 30f64), "high"),
        ],
        arange(0f64, 100f64, 0.01),
    );
    let bb_var = LinguisticVar::new(
        vec![
            (&triangular(-120f64, 1.0, 30f64), "long"),
            (&trapezoidal(-100f64, -50f64, 50f64, 100f64, 1.0), "wait"),
            (&triangular(120f64, 1.0, 30f64), "short"),
        ],
        arange(-150f64, 150f64, 0.01),
    );
    let long = LinguisticVar::new(
        vec![
            (&triangular(0f64, 1.0, 15f64), "weak"),
            (&triangular(30f64, 1.0, 30f64), "strong"),
            (&triangular(100f64, 1.0, 60f64), "verystrong"),
        ],
        arange(0f64, 100f64, 0.01),
    );
    let short = LinguisticVar::new(
        vec![
            (&triangular(0f64, 1.0, 15f64), "weak"),
            (&triangular(30f64, 1.0, 30f64), "strong"),
            (&triangular(100f64, 1.0, 60f64), "verystrong"),
        ],
        arange(0f64, 100f64, 0.01),
    );

    //rsi_var.plot("rsi".into(), "img/rsi.svg".into())?;
    //bb_var.plot("bollinger bands".into(), "img/bb.svg".into())?;
    //long.plot("long, short".into(), "img/ls.svg".into())?;

    let mut f_engine = FuzzyEngine::new([rsi_var, bb_var], [long, short]);

    f_engine.add_rule(["high", "long"], ["weak", "weak"]);
    f_engine.add_rule(["high", "wait"], ["weak", "strong"]);
    f_engine.add_rule(["high", "short"], ["weak", "verystrong"]);

    f_engine.add_rule(["medium", "long"], ["weak", "strong"]);
    f_engine.add_rule(["medium", "wait"], ["weak", "weak"]);
    f_engine.add_rule(["medium", "short"], ["strong", "weak"]);

    f_engine.add_rule(["low", "long"], ["verystrong", "weak"]);
    f_engine.add_rule(["low", "wait"], ["strong", "weak"]);
    f_engine.add_rule(["low", "short"], ["weak", "weak"]);

    let data = read_csv("eth.csv");
    let rsi = data::rsi(&data, 14)[2256..].to_vec();
    let bb = data::bb(&data, 20)[2256..].to_vec();
    let price: Vec<f64> = data[2256..].iter().map(|x| x.price).collect();
    let bb_inputs: Vec<f64> = price
        .iter()
        .zip(bb.iter())
        .map(|(p, y)| 100.0 * (p - y.0) / (2.0 * y.1))
        .collect();

    let date: Vec<Date<Local>> = data[2256..]
```

```
        .iter()
        .map(|x| parse_time(x.snapped_at.as_str()))
        .collect();

    let mut long_singal: Vec<f64> = vec![];
    let mut short_singal: Vec<f64> = vec![];
    for i in 0..price.len() {
        let result = f_engine.calculate([rsi[i], bb_inputs[i]]);
        long_singal.push(result[0].centroid_defuzz());
        short_singal.push(result[1].centroid_defuzz());
    }
    /*
    plot(
        &date,
        [&price, &long_singal, &short_singal],
        "img/chart.svg",
    )?;
    */

    backtest::f_backtest(&price, &long_singal, false);
    backtest::f_backtest(&price, &short_singal, true);
    backtest::c_backtest(&price, &rsi, &bb, false);
    backtest::c_backtest(&price, &rsi, &bb, true);

    Ok(())
}
```

rule.rs

```
use crate::set::*;

pub struct FuzzyEngine<const N: usize, const M: usize> {
    inputs_var: [LinguisticVar; N],
    outputs_var: [LinguisticVar; M],
    rules: Vec<(Vec<String>, Vec<String>)>, // list of ([input1_term, input2_term, ...] -> output_term)
}

impl<const N: usize, const M: usize> FuzzyEngine<N, M> {
    pub fn new(
        inputs_var: [LinguisticVar; N],
        output_var: [LinguisticVar; M],
    ) -> FuzzyEngine<N, M> {
        FuzzyEngine {
            inputs_var,
            outputs_var: output_var,
            rules: Vec::<(Vec<String>, Vec<String>)>::new(),
        }
    }

    pub fn add_rule(&mut self, cond: [&str; N], res: [&str; M]) {
        for i in 0..self.inputs_var.len() {
            self.inputs_var[i].term(&cond[i]); // check if term "cond[i]" exist
        }
        for i in 0..self.outputs_var.len() {
            self.outputs_var[i].term(&res[i]); // term() check if term "res" is exist
        }

        let conditions: Vec<String> = cond.iter().map(|x| x.to_string()).collect();
        let results: Vec<String> = res.iter().map(|x| x.to_string()).collect();
        self.rules.push((conditions, results));
    }

    pub fn calculate(&self, inputs: [f64; N]) -> Vec<FuzzySet> {
        let mut temp: Vec<Vec<FuzzySet>> = vec![];
        for j in 0..self.rules.len() {
            let mut aj = f64::MAX;
            for i in 0..self.rules[j].0.len() {
                let fuzzy_set = self.inputs_var[i].term(&self.rules[j].0[i]);
                let v = fuzzy_set.degree_of(inputs[i]);
                aj = aj.min(v);
            }

            let mut t: Vec<FuzzySet> = vec![];
            for i in 0..self.rules[j].1.len() {
                t.push(
                    self.outputs_var[i]
```

```
                        .term(&self.rules[j].1[i])
                        .min(aj, format!("f{}", j)),
                );
            }

            temp.push(t);
        }
        let mut res: Vec<FuzzySet> = vec![];
        for i in 0..M {
            res.push(temp[0][i].std_union(&temp[0][i], "".into()));
        }
        for j in 1..temp.len() {
            for i in 0..M {
                res[i] = res[i].std_union(&temp[j][i], "".into());
            }
        }
        res
    }
}

#[cfg(test)]
mod tests {
    use super::*;
    use crate::shape::*;
    use std::error::Error;

    #[test]
    #[should_panic]
    fn test_adding_rule() {
        let rsi = LinguisticVar::new(
            vec![
                (&triangular(20f64, 1.0, 20f64), "low"),
                (&triangular(80f64, 1.0, 20f64), "high"),
            ],
            arange(0f64, 100f64, 0.01),
        );

        let mut f_engine = FuzzyEngine::new([rsi.clone()], [rsi]);
        f_engine.add_rule(["medium".into()], ["low".into()]);
    }

    #[test]
    fn basic_test() -> Result<(), Box<dyn Error>> {
        let rsi = LinguisticVar::new(
            vec![
                (&triangular(20f64, 1.0, 20f64), "low"),
                (&triangular(80f64, 1.0, 20f64), "high"),
            ],
            arange(0f64, 100f64, 0.01),
        );

        let ma = LinguisticVar::new(
            vec![(&triangular(30f64, 0.8, 20f64), "low")],
            arange(0f64, 100f64, 0.01),
        );

        let trend = LinguisticVar::new(
            vec![(&triangular(30f64, 1f64, 30f64), "weak")],
            arange(0f64, 100f64, 0.01),
        );

        rsi.plot("rsi".into(), "img/test/rsi.svg".into())?;
        ma.plot("ma".into(), "img/test/ma.svg".into()).unwrap();
        ma.plot("ma".into(), "img/test/ma.svg".into()).unwrap();
        trend
            .plot("trend".into(), "img/test/trend.svg".into())
            .unwrap();

        let mut f_engine = FuzzyEngine::new([rsi, ma], [trend]);

        f_engine.add_rule(["low", "low"], ["weak"]);
        f_engine.add_rule(["high", "low"], ["weak"]);

        let res = f_engine.calculate([15f64, 25f64]);
        res[0].plot("t2".into(), "img/test/t2.svg".into())?;
```

```
        Ok(())
    }
}
```

set.rs

```
use crate::shape::*;
use plotters::prelude::*;
use std::error::Error;

pub fn arange(start: f64, stop: f64, interval: f64) -> Vec<f64> {
    if stop < start {
        panic!("end can not be less than start");
    } else if interval <= 0f64 {
        panic!("interval must be > 0");
    }

    let mut members: Vec<f64> = vec![];
    let r = 1.0 / interval;
    let mut n = start;
    while n <= stop {
        members.push(n);
        n += interval;
        if interval < 1.0 {
            n = (n * r).round() / r;
        }
    }
    members
}

#[derive(Clone)]
pub struct LinguisticVar {
    pub sets: Vec<FuzzySet>,
    pub universe: Vec<f64>,
}

impl LinguisticVar {
    pub fn new(inputs: Vec<(&dyn Shape, &str)>, universe: Vec<f64>) -> LinguisticVar {
        let mut sets: Vec<FuzzySet> = vec![];
        for item in inputs {
            sets.push(FuzzySet::new(&universe, item.0, item.1.to_string()));
        }
        LinguisticVar { sets, universe }
    }

    pub fn term(&self, name: &str) -> &FuzzySet {
        match self.sets.iter().find(|x| x.name == name.to_string()) {
            Some(x) => x,
            None => panic![
                "there're no fuzzy set name {} in this linguistic variable",
                name
            ],
        }
    }

    pub fn plot(&self, name: String, path: String) -> Result<(), Box<dyn Error>> {
        let root = SVGBackend::new(&path, (1024, 768)).into_drawing_area();
        root.fill(&WHITE)?;

        let mut chart = ChartBuilder::on(&root)
            .caption(name, ("Hack", 70, FontStyle::Bold).into_font())
            .set_label_area_size(LabelAreaPosition::Left, 60)
            .set_label_area_size(LabelAreaPosition::Bottom, 60)
            .margin(60)
            .build_cartesian_2d(
                self.universe[0]..self.universe[self.universe.len() - 1],
                0f64..1f64,
            )?;

        chart
            .configure_mesh()
            .disable_x_mesh()
            .y_max_light_lines(0)
            .x_labels(5)
            .y_labels(5)
            .x_label_style(("Hack", 40).into_font())
```

```
            .y_label_style(("Hack", 40).into_font())
            .draw()?;

        for i in 0..self.sets.len() {
            let color = Palette99::pick(i);
            chart
                .draw_series(LineSeries::new(
                    self.universe
                        .iter()
                        .zip(self.sets[i].membership.iter())
                        .map(|(x, y)| (*x, *y)),
                    color.mix(0.5).stroke_width(4),
                ))?
                .label(self.sets[i].name.clone())
                .legend(move |(x, y)| {
                    PathElement::new([(x, y), (x + 20, y)], color.filled().stroke_width(4))
                });
            // there're a problem with styling legend
        }

        chart
            .configure_series_labels()
            .label_font(("Hack", 50).into_font())
            .background_style(&WHITE)
            .border_style(&BLACK)
            .draw()?;

        root.present()?;
        Ok(())
    }
}

#[derive(Debug, Clone)]
pub struct FuzzySet {
    pub name: String,
    pub universe: Vec<f64>, // universe of discourse that own this set
    pub membership: Vec<f64>,
}

impl FuzzySet {
    pub fn new(universe: &Vec<f64>, fuzzy_f: &dyn Shape, name: String) -> FuzzySet {
        let mut membership: Vec<f64> = vec![];
        for i in 0..universe.len() {
            membership.push(fuzzy_f.function(universe[i]));
        }
        FuzzySet {
            name: name.to_string(),
            universe: universe.clone(),
            membership,
        }
    }

    pub fn plot(&self, name: String, path: String) -> Result<(), Box<dyn Error>> {
        let root = SVGBackend::new(&path, (1024, 768)).into_drawing_area();
        root.fill(&WHITE)?;

        let mut chart = ChartBuilder::on(&root)
            .caption(name, ("Hack", 44, FontStyle::Bold).into_font())
            .set_label_area_size(LabelAreaPosition::Left, 60)
            .set_label_area_size(LabelAreaPosition::Bottom, 60)
            .margin(20)
            .build_cartesian_2d(
                self.universe[0]..self.universe[self.universe.len() - 1],
                0f64..1f64,
            )?;

        chart
            .configure_mesh()
            .disable_x_mesh()
            .disable_y_mesh()
            .draw()?;

        let color = Palette99::pick(0);
        chart
            .draw_series(LineSeries::new(
                self.universe
```

```
                .iter()
                .zip(self.membership.iter())
                .map(|(x, y)| (*x, *y)),
            color.stroke_width(2),
        ))?
        .label(self.name.clone())
        .legend(move |(x, y)| PathElement::new([(x, y), (x + 20, y)], color.filled()));

    chart
        .configure_series_labels()
        .label_font(("Hack", 14).into_font())
        .background_style(&WHITE)
        .border_style(&BLACK)
        .draw()?;

    root.present()?;

    Ok(())
}

pub fn degree_of(&self, input: f64) -> f64 {
    // edge case
    if input < self.universe[0] {
        return self.membership[0];
    } else if input > self.universe[self.universe.len() - 1] {
        return self.membership[self.membership.len() - 1];
    }
    let mut min_x = f64::MAX;
    let mut j: usize = 0;
    for (i, x) in self.universe.iter().enumerate() {
        let diff = (x - input).abs();
        if diff < min_x {
            j = i;
            min_x = diff;
        }
    }
    self.membership[j]
}

pub fn centroid_defuzz(&self) -> f64 {
    let top_sum = self
        .universe
        .iter()
        .enumerate()
        .fold(0.0, |s, (x, y)| s + (self.membership[x] * y));
    let bot_sum = self.membership.iter().fold(0.0, |s, v| s + v);
    if bot_sum == 0.0 {
        return 0.0;
    }
    top_sum / bot_sum
}

pub fn min(&self, input: f64, name: String) -> FuzzySet {
    let mut membership: Vec<f64> = vec![];
    for i in 0..self.membership.len() {
        membership.push(self.membership[i].min(input));
    }
    FuzzySet {
        name: name.to_string(),
        universe: self.universe.clone(),
        membership,
    }
}

pub fn std_union(&self, set: &FuzzySet, name: String) -> FuzzySet {
    // check if domain is equal or not?
    if self.universe != set.universe {
        panic!("domain needs to be equal");
    }

    // if equal
    let mut membership: Vec<f64> = vec![];
    for i in 0..self.membership.len() {
        membership.push(self.membership[i].max(set.membership[i]));
    }
    FuzzySet {
```

```
            name: name.to_string(),
            universe: self.universe.clone(),
            membership,
        }
    }

    pub fn std_intersect(&self, set: &FuzzySet, name: String) -> FuzzySet {
        // check if domain is equal or not?
        if self.universe != set.universe {
            panic!("domain needs to be equal");
        }

        // if equal
        let mut membership: Vec<f64> = vec![];
        for i in 0..self.membership.len() {
            membership.push(self.membership[i].min(set.membership[i]));
        }
        FuzzySet {
            name: name.to_string(),
            universe: self.universe.clone(),
            membership,
        }
    }
}

#[cfg(test)]
mod tests {
    use super::*;

    #[test]
    fn test_arange() {
        assert_eq!(arange(0f64, 5f64, 1f64), vec![0.0, 1.0, 2.0, 3.0, 4.0, 5.0]);
        assert_eq!(
            arange(0f64, 0.5f64, 0.1f64),
            vec![0.0, 0.1, 0.2, 0.3, 0.4, 0.5]
        );
    }

    #[test]
    fn test_degree() {
        let s1 = FuzzySet::new(
            &arange(0.0, 10.0, 0.01),
            &triangular(5f64, 0.8f64, 3f64),
            "f1".into(),
        );

        assert_eq!(s1.degree_of(11.0f64), 0.0);
        assert_eq!(s1.degree_of(5.0f64), 0.8);
        assert_eq!(s1.degree_of(3.5f64), 0.4);
        assert_eq!(s1.degree_of(0.0f64), 0.0);
        assert_eq!(s1.degree_of(-1.0f64), 0.0);
    }

    #[test]
    fn linguistic() {
        let var1 = LinguisticVar::new(
            vec![
                (&triangular(5f64, 0.8, 3f64), "normal"),
                (&triangular(3f64, 0.8, 1.5f64), "weak"),
            ],
            arange(0f64, 10f64, 0.01),
        );

        assert_eq!(var1.term("normal").degree_of(5.0), 0.8);
        assert_eq!(var1.term("weak").degree_of(3.0), 0.8);

        var1.plot("var1".into(), "img/t.svg".into()).unwrap();
    }
}
```

shape.rs

```
pub trait Shape {
    fn function(&self, x: f64) -> f64;
}
```

```rust
pub struct Triangular {
    a: f64,
    b: f64,
    s: f64,
}

impl Shape for Triangular {
    fn function(&self, x: f64) -> f64 {
        if (self.a - self.s) <= x && x <= (self.a + self.s) {
            return self.b * (1.0 - (x - self.a).abs() / self.s);
        }
        0.0
    }
}

pub fn triangular(a: f64, b: f64, s: f64) -> Triangular {
    Triangular { a, b, s }
}

pub struct Trapezoidal {
    a: f64,
    b: f64,
    c: f64,
    d: f64,
    e: f64,
}

impl Shape for Trapezoidal {
    fn function(&self, x: f64) -> f64 {
        if x >= self.a && x < self.b {
            return ((x - self.a) * self.e) / (self.b - self.a);
        } else if x >= self.b && x <= self.c {
            return self.e;
        } else if x > self.c && x <= self.d {
            return self.e * (1.0 - (x - self.c).abs() / (self.d - self.c));
        }
        0.0
    }
}

pub fn trapezoidal(a: f64, b: f64, c: f64, d: f64, e: f64) -> Trapezoidal {
    Trapezoidal { a, b, c, d, e }
}
```

data.rs

```rust
use serde::Deserialize;
use std::fs;

#[derive(Deserialize, Debug)]
pub struct Record {
    pub snapped_at: String,
    pub price: f64,
}

/// return (gain, loss)
fn gain_loss(p0: f64, p1: f64) -> (f64, f64) {
    let mut res = (0.0, 0.0);
    if p1 > p0 {
        res.0 = p1 - p0
    } else {
        res.1 = -(p1 - p0)
    }
    res
}

pub fn rsi(data: &Vec<Record>, n: usize) -> Vec<f64> {
    let mut rsi: Vec<f64> = vec![];

    let mut gains: Vec<f64> = vec![];
    let mut losses: Vec<f64> = vec![];

    let mut avg_g: Vec<f64> = vec![];
    let mut avg_l: Vec<f64> = vec![];
```

```
    for (i, item) in data.iter().enumerate() {
        if i <= n + 1 && i > 0 {
            let gl = gain_loss(data[i - 1].price, item.price);
            gains.push(gl.0);
            losses.push(gl.1);
        }

        if i <= n {
            rsi.push(-1.0);
            avg_g.push(0.0);
            avg_l.push(0.0);
        } else if i == n + 1 {
            let avg_gain = gains.iter().sum::<f64>() / (n as f64);
            let avg_loss = losses.iter().sum::<f64>() / (n as f64);
            rsi.push(100f64 - 100f64 / (1.0 + (avg_gain / avg_loss)));
            avg_g.push(avg_gain);
            avg_l.push(avg_loss);
        } else {
            let gl = gain_loss(data[i - 1].price, item.price);
            let avg_gain = (avg_g[i - 1] * (n - 1) as f64 + gl.0) / (n as f64);
            let avg_loss = (avg_l[i - 1] * (n - 1) as f64 + gl.1) / (n as f64);
            rsi.push(100f64 - 100f64 / (1.0 + (avg_gain / avg_loss)));
            avg_g.push(avg_gain);
            avg_l.push(avg_loss);
        }
    }
    rsi
}

/// return vector of (ma, std)
pub fn bb(data: &Vec<Record>, n: usize) -> Vec<(f64, f64)> {
    let mut bb: Vec<(f64, f64)> = vec![];
    for (i, _) in data.iter().enumerate() {
        if i < n {
            bb.push((-1.0, -1.0));
            continue;
        }

        let mut sum = 0.0;
        for j in (i - n)..i {
            sum += data[j].price;
        }
        let ma = sum / (n as f64);

        let mut std_sum = 0.0;
        for j in (i - n)..i {
            std_sum += (data[j].price - ma).powi(2);
        }
        let std = (std_sum / n as f64).sqrt();
        bb.push((ma, std));
    }
    bb
}

/// read coingecko csv data
pub fn read_csv(path: &str) -> Vec<Record> {
    let contents = fs::read_to_string(path).unwrap();

    let mut rdr = csv::Reader::from_reader(contents.as_bytes());
    let mut data: Vec<Record> = vec![];
    for result in rdr.deserialize() {
        let record: Record = result.unwrap();
        data.push(record);
    }
    data
}
```

backtest.rs

```
// get long, short signal as an input
// iterate over all data with inital capital 1000£
// entry when signal is >= 40
// entry size is 20% of capital
// stop-loss when price go down 10%
```

```rust
// take profit when price go up 20%

struct Position {
    at_price: f64,
    at_time: usize,
    amount: f64,
    realize: bool,
}

impl Position {
    pub fn new(price: f64, money: f64, time: usize) -> Position {
        Position {
            at_price: price,
            at_time: time,
            amount: money / price,
            realize: false,
        }
    }
}

fn realizing_pos(capital: f64, price: &Vec<f64>, pos_list: &mut Vec<Position>, pos_type: bool) {
    let mut profit: Vec<f64> = vec![];
    let mut losses: Vec<f64> = vec![];
    for (i, p) in price.iter().enumerate() {
        for j in 0..pos_list.len() {
            if i <= pos_list[j].at_time || pos_list[j].realize == true {
                continue;
            }
            let diff = (p - pos_list[j].at_price) / pos_list[j].at_price * 100.0;
            if !pos_type {
                if diff > 20.0 {
                    profit.push((p - pos_list[j].at_price) * pos_list[j].amount);
                    pos_list[j].realize = true;
                } else if diff < -10.0 {
                    losses.push((p - pos_list[j].at_price) * pos_list[j].amount);
                    pos_list[j].realize = true;
                }
            } else {
                if diff < -20.0 {
                    profit.push(-1.0 * (p - pos_list[j].at_price) * pos_list[j].amount);
                    pos_list[j].realize = true;
                } else if diff > 10.0 {
                    losses.push(-1.0 * (p - pos_list[j].at_price) * pos_list[j].amount);
                    pos_list[j].realize = true;
                }
            }
        }
    }

    let total_profit = profit.iter().fold(0.0, |s, x| s + x);
    let total_losses = losses.iter().fold(0.0, |s, x| s + x);
    println!("total trade: {:.3}", pos_list.len());
    println!("net profit: {:.3}", total_profit + total_losses);
    println!("count: {}, profits: {:.3}", profit.len(), total_profit);
    println!("count: {}, losses: {:.3}", losses.len(), total_losses);
    println!("---------------");
    /*
    println!(
        "total capital(+net_profit): {}",
        capital + (1000.0 - capital) + total_profit + total_losses
    );
    */
}

/// Fuzzy BackTest
/// pos_type - false for long, true for short
pub fn f_backtest(price: &Vec<f64>, signal: &Vec<f64>, pos_type: bool) {
    let mut capital = 1000.0;
    let mut pos_list: Vec<Position> = vec![];

    for (i, p) in price.iter().enumerate() {
        if signal[i] >= 40.0 {
            if capital > 0.0 {
                let pos = Position::new(*p, 100.0, i);
                capital -= 100.0;
                pos_list.push(pos);
```

```
            }
        }
    }

    realizing_pos(capital, price, &mut pos_list, pos_type)
}

pub fn c_backtest(price: &Vec<f64>, rsi: &Vec<f64>, bb: &Vec<(f64, f64)>, pos_type: bool) {
    let mut capital = 1000.0;
    let mut pos_list: Vec<Position> = vec![];

    for (i, p) in price.iter().enumerate() {
        let beta = (p - bb[i].0) / (2.0 * bb[i].1);
        if !pos_type {
            if rsi[i] < 30.0 && beta < -0.9 {
                if capital > 0.0 {
                    let pos = Position::new(*p, 100.0, i);
                    capital -= 100.0;
                    pos_list.push(pos);
                } else if rsi[i] < 30.0 && beta >= -0.9 && beta < 0.0 {
                    let pos = Position::new(*p, 100.0, i);
                    capital -= 100.0;
                    pos_list.push(pos);
                }
            }
        } else {
            if rsi[i] > 70.0 && beta < 0.9 {
                if capital > 0.0 {
                    let pos = Position::new(*p, 100.0, i);
                    capital -= 100.0;
                    pos_list.push(pos);
                } else if rsi[i] > 70.0 && beta <= 0.9 && beta > 0.0 {
                    let pos = Position::new(*p, 100.0, i);
                    capital -= 100.0;
                    pos_list.push(pos);
                }
            }
        }
    }

    realizing_pos(capital, price, &mut pos_list, pos_type)
}
```