SOC - Week 2

Radhika Agarwal

June 2025

MACD

```cpp
#include <vector>
#include <iostream>
#include <limits>
using namespace std;

vector<double> calculate_ema(const vector<double>& prices, int span) {
    vector<double> ema_values;

    if (prices.empty() || span <= 0)
        return ema_values;

    double alpha = 2.0 / (span + 1);

    double ema = prices[0];
    ema_values.push_back(ema);

    for (size_t i = 1; i < prices.size(); ++i) {
        ema = alpha * prices[i] + (1.0 - alpha) * ema;
        ema_values.push_back(ema);
    }
    return ema_values;
}

void calculate_MACD(const vector<double>& prices, vector<double>& MACD, vector<double>& Signal, vector<double>& Buy, vector<double>& Sell){

    vector<double> short_ema = calculate_ema(prices, 12);
    vector<double> long_ema = calculate_ema(prices, 26);

    MACD.clear();
    size_t min_size = min(short_ema.size(), long_ema.size());
    for (size_t i = 0; i < min_size; ++i) {
        MACD.push_back(short_ema[i] - long_ema[i]);
    }

    Signal = calculate_ema(MACD, 9);

    int flag = -1;
```

```
    for (size_t i = 0; i < min_size; ++i){
        if MACD[i] > Signal[i]){
            Sell.push_back(numeric_limits<double>::quiet_NaN());
            if (flag!=-1){
                Buy.push_back(prices[i]);
                flag = 1;
            }
            else{
                Buy.push_back(numeric_limits<double>::quiet_NaN());
            }
        }
        else if (MACD[i] < Signal[i]){
            Buy.push_back(numeric_limits<double>::quiet_NaN());
            if(flag!=0){
                Sell.push_back(prices[i]);
                flag = 0;
            }
            else{
                Sell.push_back(numeric_limits<double>::quiet_NaN())
                    ;
            }
        }
        else{
            Buy.push_back(numeric_limits<double>::quiet_NaN());
            Sell.push_back(numeric_limits<double>::quiet_NaN());
        }
    }
}
```

Bollinger Bands

```
#include <iostream>
#include <vector>
#include <cmath>
#include <numeric>
#include <limits>

using namespace std;

vector<double> sma(const vector<double>& prices, int window) {
    vector<double> sma(prices.size(), numeric_limits<double>::
        quiet_NaN());

    for (size_t i = window - 1; i < prices.size(); ++i) {
        double sum = accumulate(prices.begin() + i - window + 1,
            prices.begin() + i + 1, 0.0);
        sma[i] = sum / window;
    }

    return sma;
}

vector<double> std_dev(const vector<double>& prices, int window) {
    vector<double> std_dev(prices.size(), numeric_limits<double>::
        quiet_NaN());
```

```
    for (size_t i = window - 1; i < prices.size(); ++i) {
        double mean = accumulate(prices.begin() + i - window + 1, prices.begin() + i + 1, 0.0) / window;
        double sum_sq_diff = 0.0;

        for (size_t j = i - window + 1; j <= i; ++j) {
            sum_sq_diff += pow(prices[j] - mean, 2);
        }

        std_dev[i] = sqrt(sum_sq_diff / window);
    }

    return std_dev;
}

void bollinger_bands(const vector<double>& prices, int window, double num_std_dev,
                     vector<double>& sma, vector<double>& upper_band, vector<double>& lower_band, vector<string>& signal) {
    sma = sma(prices, window);
    vector<double> std_dev = std_dev(prices, window);

    upper_band.resize(prices.size(), numeric_limits<double>::quiet_NaN());
    lower_band.resize(prices.size(), numeric_limits<double>::quiet_NaN());
    signal.resize(prices.size(), "HOLD");

    for (size_t i = 0; i < prices.size(); ++i) {
        if (!isnan(sma[i]) && !isnan(std_dev[i])) {
            upper_band[i] = sma[i] + num_std_dev * std_dev[i];
            lower_band[i] = sma[i] - num_std_dev * std_dev[i];

            double margin = 0.5;
            if (abs(prices[i] - lower_band[i]) < margin)
                signal[i] = "BUY";
            else if (abs(prices[i] - upper_band[i]) < margin)
                signal[i] = "SELL";
        }
    }
}
```

Fibonacci Retrcement

```
    #include <vector>
    #include <algorithm>

    using namespace std;

    vector<double> fibonacci_levels(const vector<double>& prices) {
        if (prices.empty()) return {};

```

```
        double high = *max_element(prices.begin(), prices.end());
        double low  = *min_element(prices.begin(), prices.end());

        vector<double> levels = {0.0, 0.236, 0.382, 0.5, 0.618,
            0.786, 1.0};
        vector<double> retracements;

        for (double level : levels) {
            double value = high - (high - low) * level;
            retracements.push_back(value);
        }

        return retracements;
    }

    string get_signal(double price, const vector<double>& levels) {

        double level_618 = levels[4];
        double level_382 = levels[2];

        if (price > level_618) return "BUY";
        else if (price < level_382) return "SELL";
        else return "HOLD";
    }
```