# `security_price_history`, a C++ class for dealing with security histories.

Bernt Arne Ødegaard

April 2007

# Contents

# Chapter 1

# Security price history

The following class is used when we have a lot of historical data about securities. Given price observations, do various calculations and pulling of data, as well as printing various reports.

### 1.0.1 Security price history

When doing empirical work in finance, we always end up with the same basic problem. From a time series of price observations of a financial security, we want to calculate any number of things: Returns at different frequencies, average returns, max, min returns, volatility,...

We always return to the basic issue of storing data for the underlying prices. The class implemented in the following is an attempt to solve that problem once and for all. The purpose of the `security_price_history` class is to hold a *price history* for a security. The emphasis is on efficiently storing this data. The general idea is to store the data in the following form:

Security Name

| Date | bid price | trade price | ask price |
|---|---|---|---|
| 2 jan 1990 | 99 | 100 | 101 |
| 3 jan 1990 | 98 | 99 | 100 |
| . | . | . | . |
| . | . | . | . |

One assumption made here is that the data is not on a higher frequency than daily. If so need to change the `date` class to a `date_time` class that also stores time of day.

Calculating returns etc will then be simply provided as functions that work on this data structure.

This class is also useful as a base class. For example, a stock will need added functions for dividends, adjustments etc.

## 1.1 Header file

The header file defines the class interface.

```
#ifndef _SECURITY_PRICE_HISTORY_H_
#define _SECURITY_PRICE_HISTORY_H_

#include "dated_obs.h"

class security_price_history {
private:
    string          security_name_;
    vector<date> dates_;
    vector<double> bids_;
    vector<double> trades_;
    vector<double> asks_;
public:
    security_price_history();
    security_price_history(const string&);
    security_price_history(const security_price_history&);
    security_price_history operator=(const security_price_history&);
    ~security_price_history();
    void clear();
    void set_security_name(const string&);
    void add_prices(const date& d, const double& bid, const double& trade, const double& ask);
    // the most important function, all dates is added through calls to add_prices

    void set_bid (const int& i, const double& bid); // to change current data.
    void set_trade (const int& i, const double& trade);
    void set_ask (const int& i, const double& ask);

    void erase(const date&); // delete on given dates
    void erase_before(const date&);
    void erase_between(const date&, const date&);
    void erase_after(const date&);
    bool contains(const date&) const; // check whether this date is present
private:
    int index_of_date(const date&) const;// where in vector is this date?
    int index_of_last_date_before(const date&) const ;
    int index_of_first_date_after(const date&) const ;
public:
    bool empty() const;
    int size() const;// { return dates_.size(); };
    string security_name() const;
    vector<date> dates() const;

    int no_dates() const;
    int no_prices() const;
    int no_bids() const;
    int no_trades() const;
    int no_asks() const;
    int no_prices_between(const date&, const date&) const;
    date date_at(const int&) const;
    double bid(const int&) const;
    double bid(const date&) const;
    double trade(const int&) const;
    double trade(const date&) const;
    double ask(const int&) const;
    double ask(const date&) const;
    date first_date() const;
    date last_date() const;
    double price(const date&) const; // price at given data
    double current_price(const date&) const; // price at or before given data
    double price(const int&) const;
    double buy_price(const int&) const;
    double sell_price(const int&) const;
};

dated_observations prices(const security_price_history&);
vector<date> dates(const security_price_history&);

#endif
```

**Header file 1.1:** security price history.h

## 1.2 Implementation

```
#include "security_price_history.h"

security_price_history:: security_price_history(){ clear(); }; // make sure empty

security_price_history:: security_price_history(const string& name){
   clear();
   security_name_ = name;
};
security_price_history:: ~security_price_history(){ clear(); };

// copy construction
security_price_history:: security_price_history(const security_price_history& sh) {
    clear();
    dates_ = vector<date>(sh.size());
    bids_  = vector<double>(sh.size());
    trades_ = vector<double>(sh.size());
    asks_  = vector<double>(sh.size());
    for (unsigned i=0;i<sh.no_dates();++i){
         dates_[i]=sh.date_at(i);
         bids_[i]=sh.bid(i);
         trades_[i]=sh.trade(i);
         asks_[i]=sh.ask(i);
    };
    set_security_name(sh.security_name());
};

security_price_history security_price_history:: operator= (const security_price_history& sh) {
    clear();
    dates_ = vector<date>(sh.size());
    bids_ = vector<double>(sh.size());
    trades_= vector<double>(sh.size());
    asks_ = vector<double>(sh.size());
    for (unsigned i=0;i<sh.no_dates();++i){
         dates_[i]=sh.date_at(i);
         bids_[i]=sh.bid(i);
         trades_[i]=sh.trade(i);
         asks_[i]=sh.ask(i);
    };
    set_security_name(sh.security_name());
    return (*this);
};

void security_price_history::clear() {
    dates_.erase(dates_.begin(),dates_.end());
    bids_.erase(bids_.begin(),bids_.end());
    trades_.erase(trades_.begin(),trades_.end());
    asks_.erase(asks_.begin(),asks_.end());
    security_name_=string();
};
string security_price_history::security_name() const { return security_name_; };
void security_price_history::set_security_name(const string& s)     { security_name_ = s; };
date  security_price_history::date_at(const int& t) const { return dates_[t]; };

bool security_price_history::empty() const{
    if ( (no_dates()<1) && (security_name().length()<1) ) return true;
    return false;
};

void security_price_history::set_bid (const int& i, const double& bid) { if (i<no_dates()) bids_[i]=bid; };
void security_price_history::set_trade (const int& i, const double& trade) { if (i<no_dates()) trades_[i]=trade; };
void security_price_history::set_ask (const int& i, const double& ask) { if (i<no_dates()) asks_[i]=ask; };

double security_price_history::bid (const int& t) const { return bids_[t]; };
double security_price_history::trade(const int& t) const { return trades_[t]; };
double security_price_history::ask (const int& t) const { return asks_[t]; };
```

**C++ Code 1.1:** Implementing the class

```
#include "security_price_history.h"

int security_price_history::no_dates () const {
    return int(dates_.size());
};

int security_price_history::no_prices() const {
    int n_prices=0;
    for (int i=0;i<no_dates();++i){
        if ( (bid(i)>0) || (trade(i)>0) || (ask(i)>0) ) { ++n_prices; };
    };
    return n_prices;
};

int security_price_history::no_prices_between(const date& first, const date& last) const {
    int n_pric_between=0;
    for (int i=0;i<no_dates();++i) {
        if  ((date_at(i)>=first) && (date_at(i)<=last)) {
            if((bid(i)>0)||(trade(i)>0)||(ask(i)>0)) {++n_pric_between;};
        };
    };
    return n_pric_between;
};

int security_price_history::no_bids() const {
    int n_prices=0;
    for (int i=0;i<no_dates();++i){
        if (bid(i)>0) { ++n_prices; };
    };
    return n_prices;
};

int security_price_history::no_trades() const {
    int n_prices=0;
    for (int i=0;i<no_dates();++i){
        if (trade(i)>0) { ++n_prices; };
    };
    return n_prices;
};

int security_price_history::no_asks() const {
    int n_prices=0;
    for (int i=0;i<no_dates();++i){
        if (ask(i)>0) { ++n_prices; };
    };
    return n_prices;
};
```

**C++ Code 1.2:** Sizes

```
#include "security_price_history.h"
const double MISSING_OBS=−1;
double security_price_history::price(const int& i) const{
    if (i>=no_dates()) { return MISSING_OBS; };
    if (trade(i)>0.0) { return trade(i); };
    if ( (bid(i)>0.0) && (ask(i)>0.0)) { return 0.5*(bid(i)+ask(i)); }; // return bid ask average
    if (bid(i)>0.0)  { return bid(i); }; // to avoid big jumps,
    if (ask(i)>0.0)  { return ask(i); }; // skip these
    return MISSING_OBS;
};

double security_price_history::bid(const date& d) const {
  int i = index_of_date(d); // return price on date, only if obs on that date, else return missing
  if (i>=0) return bid(i);
  return MISSING_OBS;
};

double security_price_history::trade(const date& d) const {
  int i = index_of_date(d);
  if (i>=0) return trade(i);
  return MISSING_OBS;
};

double security_price_history::ask(const date& d) const {
  int i = index_of_date(d);
  if (i>=0) return ask(i);
  return MISSING_OBS;
};

double security_price_history::price(const date& d) const {
  int i = index_of_date(d);
  if (i>=0) return price(i);
  return MISSING_OBS;
};

double security_price_history::current_price(const date& d) const { // return price on given date.
    if (empty())         { return MISSING_OBS; };
    if (d<first_date()) { return MISSING_OBS; }; // if before first or after last, return missing
    if (d>last_date()) { return MISSING_OBS; };
    int i = index_of_date(d); // If don't have that price, return the last one observed before the wanted date
    if (i>=0) return price(i);
    i=index_of_last_date_before(d);   // otherwise use last previously observed price.
    if (i>=0) return price(i);
    return MISSING_OBS;
};

double security_price_history::buy_price(const int& i) const{
    if (i>=no_dates()) { return MISSING_OBS; };
    if (ask(i)>0.0)  { return ask(i); };
    if (trade(i)>0.0) { return trade(i); };
    return MISSING_OBS;
};

double security_price_history::sell_price(const int& i) const{
    if (i>=no_dates()) { return MISSING_OBS; };
    if (bid(i)>0.0)  { return bid(i); };
    if (trade(i)>0.0) { return trade(i); };
    return MISSING_OBS;
};

dated_observations prices(const security_price_history& sh ){
    dated_observations pr;
    for (int i=0;i<sh.no_prices();++i) {
        date d=sh.date_at(i);
        double price = sh.price(i);
        if (price>0) pr.insert(d,price);
    };
    return pr;
};
```

**C++ Code 1.3:** Prices

```
#include "security_price_history.h"


void security_price_history::add_prices(const date& d,
                                        const double& bid,
                                        const double& trade,
                                        const double& ask) {
    if (!d.valid()) return; // don't add
    if ( (bid<0) && (trade<0) && (ask<0) ) return; // don't bother about empty dates
    if ( (no_dates()<1) || (d>last_date() ) ) {
        dates_ .push_back(d);
        trades_.push_back(trade);
        bids_  .push_back(bid);
        asks_  .push_back(ask);
        return;
    };
    if (d<first_date()) {
        dates_.insert(dates_.begin(),d);
        bids_.insert(bids_.begin(),bid);
        trades_.insert(trades_.begin(),trade);
        asks_.insert(asks_.begin(),ask);
        return;
    };
    int i = index_of_date(d);
    if (i>=0) { // found, replace
        trades_[i]=trade;
        bids_[i]=bid;
        asks_[i]=ask;
        return;
    }
    // evidently should be inserted somewhere in between other observations.
    for (i=0;i<no_dates();++i){
        if (date_at(i)>d) {
            dates_.insert(dates_.begin()+i,d);
            bids_.insert(bids_.begin()+i,bid);
            trades_.insert(trades_.begin()+i,trade);
            asks_.insert(asks_.begin()+i,ask);
            return;
        };
    };
};


int security_price_history::size() const { return int(dates_.size()); };

date security_price_history::first_date() const {
    if (dates_.size()>0) { return dates_.front();};
    return date(); // else
};

date security_price_history::last_date() const {
    if (dates_.size()>0) { return dates_.back();};
    return date();
};

vector<date> security_price_history::dates() const {
    return dates_;
};
```

**C++ Code 1.4:** Utilities

```
#include "security_price_history.h"

void security_price_history::erase(const date& d) {
   int i = index_of_date(d);
   //   cout << "erasing" << endl;
   if (i>=0) { // returns -1 if not there
       //       cout << "erasing" << endl;
      dates_.erase(dates_.begin()+i);
      trades_.erase(trades_.begin()+i);
      bids_.erase(bids_.begin()+i);
      asks_.erase(asks_.begin()+i);
   };
};

void security_price_history::erase_between(const date& first,
                                           const date& last) {
   int i1 = index_of_first_date_after(first);
   int i2 = index_of_last_date_before(last);
   if ( (i1>=0) && (i2>=0) ) { // returns -1 if not there
      i2++; // the erase does not delete the last one.
      dates_.erase(dates_.begin()+i1,dates_.begin()+i2);
      trades_.erase(trades_.begin()+i1,trades_.begin()+i2);
      bids_.erase(bids_.begin()+i1,bids_.begin()+i2);
      asks_.erase(asks_.begin()+i1,asks_.begin()+i2);
   };
};

void security_price_history::erase_before(const date& d) {
   int i = index_of_last_date_before(d);
   if (i>=0) { // returns -1 if not there
      i++;// the erase does not delete the last one, so add one.
      dates_.erase(dates_.begin(),dates_.begin()+i);
      trades_.erase(trades_.begin(),trades_.begin()+i);
      bids_.erase(bids_.begin(),bids_.begin()+i);
      asks_.erase(asks_.begin(),asks_.begin()+i);
   };
};

void security_price_history::erase_after(const date& d) {
   int i = index_of_first_date_after(d);
   if (i>=0) { // returns -1 if not there
      dates_.erase(dates_.begin()+i,dates_.end());
      trades_.erase(trades_.begin()+i,trades_.end());
      bids_.erase(bids_.begin()+i,bids_.end());
      asks_.erase(asks_.begin()+i,asks_.end());
   };
};
```

**C++ Code 1.5:** Erasing

```
#include "security_price_history.h"

bool security_price_history::contains(const date& d) const {
    return binary_search(dates_.begin(),dates_.end(),d);
};

int security_price_history::index_of_date(const date& d) const {
   // this routine returns the index at which date d is, or -1 if not found.
   if (!contains(d)) return −1;
   for (int i=0;i<dates_.size();++i){
       if (dates_[i]==d) return i;
   };
   return −1;
};

int security_price_history::index_of_first_date_after(const date& d) const {
    if (!d.valid()) return −1;
    if (d>=last_date()) return −1;
    if (d<first_date()) return 0;
    for (int i=0;i<dates_.size();++i){
         if (dates_[i]>d) return i;
    };
    return −1;
};

int security_price_history::index_of_last_date_before(const date& d) const {
    if (!d.valid()) return −1;
    if (d<=first_date()) return −1;
    if (d>last_date()) return index_of_date(last_date());
    for (int i=0;i<dates_.size();++i){
         if (dates_[i]>=d) return i−1;
    };
    return −1;
};
```

**C++ Code 1.6:** Searching