`stock_price_history`, a C++ class for dealing with stock price histories.

Bernt Arne Ødegaard

April 2007

Contents

Chapter 1

Stock price history

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

1.1 Stock history

The purpose of the `stock_price_history` class is to hold a price history for a stock, with a number of special functions for that purpose.

The class is made by adding data and functions to the base class `security_price_history`. The reason the general security price history class is not enough, is the problem that dividends and other adjustments need to accounted for in returns calculations.

1.2 Header file

1.2.1 stock_price_history.h

Note that a lot of the functionality of the class is inherited from `security_price_history`

```
#ifndef _STOCK_PRICE_HISTORY_H_

#include "security_price_history.h"
#include "dated_obs.h"

class stock_price_history : public security_price_history { // most of the functionality is inherited from security price history
private:
    dated_observations dividends_;
    dated_observations adjustments_;
public:
    stock_price_history();
    stock_price_history(const stock_price_history&);
    stock_price_history operator = (const stock_price_history&);
    ~stock_price_history(){ clear(); };
    void clear();

    bool empty() const;
    //////////////////// dividends ////////////////////////
    void     add_dividend (const date&, const double&);
    void     remove_dividend (const date&);
    int      no_dividends() const;
    date     dividend_date(const int&) const;
    double   dividend(const int&) const;
    bool     dividends_between(const date&, const date&) const;
    int      no_dividends_between(const date&, const date&) const;
    double   total_dividends_between(const date&, const date&) const;
    bool     dividend_on(const date&) const;
    double    dividend(const date&) const;

    ///// adjustments //////////////////////////////
    void   add_adjustment      (const date&, const double&);
    void   remove_adjustment  (const date&);
    int    no_adjustments() const;
    date   adjustment_date(const int& i) const;
    double adjustment(const int& i) const;
    bool   adjustments_between(const date& d1, const date& d2) const;
    int    no_adjustments_between(const date& d1, const date& d2) const;
    double aggregated_adjustments_between(const date& d1, const date& d2) const;
    bool   adjustment_on(const date& d) const;
    double adjustment(const date& d) const;
};

dated_observations daily_prices    (const stock_price_history& st);
dated_observations daily_prices    (const stock_price_history& st, const date& from, const date& to);
dated_observations daily_trade_prices (const stock_price_history& st);
dated_observations daily_trade_prices (const stock_price_history& st, const date& from, const date& to);
dated_observations monthly_prices (const stock_price_history& st);
dated_observations annual_prices  (const stock_price_history& st);

dated_observations dividends(const stock_price_history&);
dated_observations dividends(const stock_price_history&, const date& d1, const date& d2);

dated_observations adjustments(const stock_price_history&);
dated_observations adjustments(const stock_price_history&, const date& d1, const date& d2);

#define _STOCK_PRICE_HISTORY_H_
#endif
```

Header file 1.1: Define all class elements

1.3 Implementation

```
#include "stock_price_history.h"

const double MISSING_OBS = −1; // Assumption: Prices are positive. Negative prices are missing observations.

stock_price_history::stock_price_history(){; };

stock_price_history::stock_price_history(const stock_price_history& sh): security_price_history(sh){
   dividends_.clear();
   for (unsigned i=0;i<sh.no_dividends();++i)
      add_dividend(sh.dividend_date(i), sh.dividend(i));
   adjustments_.clear();
   for (unsigned i=0;i<sh.no_adjustments();++i)
      add_adjustment(sh.adjustment_date(i), sh.adjustment(i));
};

stock_price_history stock_price_history:: operator= (const stock_price_history& sh) {
   clear();
   security_price_history::operator=(sh);
   for (unsigned i=0;i<sh.no_dividends();++i)
      add_dividend(sh.dividend_date(i), sh.dividend(i));
   for (unsigned i=0;i<sh.no_adjustments();++i)
      add_adjustment(sh.adjustment_date(i), sh.adjustment(i));
   return *this;
};

void stock_price_history::clear() {
    security_price_history::clear();
    dividends_.clear();
    adjustments_.clear();
};

bool stock_price_history::empty() const {
   if (no_dividends()>0) return false;
   if (no_adjustments()>0) return false;
   return security_price_history::empty();
};
```

C++ Code 1.1: Basic operations

```
#include "stock_price_history.h"
// querying
int stock_price_history::no_dividends() const { return dividends_.size(); };
date stock_price_history::dividend_date(const int& i) const { return dividends_.date_at(i); };
double stock_price_history::dividend(const int& i) const { return dividends_.element_at(i); };

bool stock_price_history::dividend_on(const date& d) const { return dividends_.contains(d); };

double stock_price_history::dividend(const date& d) const {
   double div = dividends_.element_at(d);
   if (div>0) return div;
   return 0.0;
};

void stock_price_history::add_dividend(const date& dividend_date, const double& dividend_amount) {
   dividends_.insert(dividend_date,dividend_amount);
};

void stock_price_history::remove_dividend(const date& dividend_date) { dividends_.remove(dividend_date);};

bool stock_price_history::dividends_between(const date& d1, const date& d2) const {
    date first, last;
    if (d1<d2) { first=d1; last=d2; } else { first=d2; last=d1; };
    if( dividends_.no_obs_between(first,last)>0) return true;
    return false;
};

int stock_price_history::no_dividends_between(const date& d1, const date& d2) const {
    date first, last;
    if (d1<d2) { first=d1; last=d2; } else { first=d2; last=d1; };
    return dividends_.no_obs_between(first,last);
};

double stock_price_history::total_dividends_between(const date& d1, const date& d2) const {
   if (!dividends_between(d1,d2)) return 0;
   date first, last;
   if (d1<d2) { first=d1; last=d2; } else { first=d2; last=d1; };
   double tot_dividend = 0.0;
   for (int i=0; i<dividends_.size(); ++i){
      if ( (dividends_.date_at(i)>first) && (dividends_.date_at(i)<=last) ) {
          tot_dividend += dividends_.element_at(i);
      };
   };
   return tot_dividend;
};

dated_observations dividends(const stock_price_history& sphist, const date& d1, const date& d2){
    dated_observations dividends;
    for (int i=0;i<sphist.no_dividends();++i){
         date d=sphist.dividend_date(i);
         if ( (d>=d1) && (d<=d2) && (sphist.dividend(i)>=0.0) ){
             dividends.insert(d,sphist.dividend(i));
         };
    };
    return dividends;
};

dated_observations dividends(const stock_price_history& sphist){
    return dividends(sphist,sphist.first_date(),sphist.last_date());
};
```

C++ Code 1.2: Dividends

```
#include "stock_price_history.h"

int stock_price_history::no_adjustments() const{ return adjustments_.size(); };
date stock_price_history::adjustment_date(const int& i) const{ return adjustments_.date_at(i); };
double stock_price_history::adjustment(const int& i) const { return adjustments_.element_at(i);};
bool stock_price_history::adjustment_on(const date& d) const {
   for (int i=0; i<adjustments_.size(); ++i) {
      if (adjustments_.date_at(i)==d) return true;
   };
   return false;
};
double stock_price_history::adjustment(const date& d) const {
    for (int i=0; i<adjustments_.size(); ++i){ if (adjustments_.date_at(i)==d) return adjustments_.element_at(i); };
    return 0.0;
};

void stock_price_history::add_adjustment ( const date& adjustment_date, const double& adj_factor) {
   if (adjustments_.contains(adjustment_date)) {
      adjustments_.insert(adjustment_date, adjustments_.element_at(adjustment_date)*adj_factor);
   }
   else { adjustments_.insert(adjustment_date,adj_factor); };
};

void stock_price_history::remove_adjustment(const date& adjustment_date){ adjustments_.remove(adjustment_date); };

bool stock_price_history::adjustments_between(const date& d1, const date& d2) const{
    date first, last;
    if (d1<d2) { first=d1; last=d2; } else { first=d2; last=d1; };
    if (adjustments_.no_obs_between(first,last)>0) return true;
    return false;
};

int stock_price_history::no_adjustments_between(const date& d1, const date& d2) const {
    date first, last;
    if (d1<d2) { first=d1; last=d2; } else { first=d2; last=d1; };
    return adjustments_.no_obs_between(first,last);
};

double stock_price_history::aggregated_adjustments_between(const date& d1, const date& d2) const {
    date first, last;
    if (d1<d2) { first=d1; last=d2; } else { first=d2; last=d1; };
    double tot_adjustments=1.0;
    for (int i=0; i<adjustments_.size(); ++i){
         if ( (adjustments_.date_at(i)>first) && (adjustments_.date_at(i)<=last) ) {
            tot_adjustments *= adjustments_.element_at(i);
         };
    };
    return tot_adjustments;
};

dated_observations adjustments(const stock_price_history& sphist, const date& d1, const date& d2){
    dated_observations adjustments;
    for (int i=0;i<sphist.no_adjustments();++i){
         date d=sphist.adjustment_date(i);
         if ( (d>=d1) && (d<=d2) && (sphist.adjustment(i)>0) ){
             adjustments.insert(d,sphist.adjustment(i));
         };
    };
    return adjustments;
};

dated_observations adjustments(const stock_price_history& sphist){
    return adjustments(sphist,sphist.first_date(),sphist.last_date());
};
```

C++ Code 1.3: Adjustments