`dated_observations`, C++ class for operating on dated observatons.

Bernt A Ødegaard

April 2007

Chapter 1

Dated observations

A specialization of the general `dated<T>` class to be used for time series of numbers.

```
#ifndef _DATED_OBS_H_
#define _DATED_OBS_H_

#include "dated.h" // templated dated<> class
#include <string> // ANSI string class

class dated_observations : public dated<double>{
  private:
    string          title_;
  public:
    ~dated_observations() { clear(); };
    void clear();

    void set_title(string s); // title
    string title() const;

    int no_obs() const;
    int no_obs_between(const date& d1, const date& d2) const;
    int no_obs_before(const date& d) const;
    int no_obs_after(const date& d) const;
};

///// io
ostream& operator << (ostream&, const dated_observations& );
void print_dated_observations(ostream& of, const dated_observations& d, int precision=3);

//// mischellaneous utilities
double max_obs(dated_observations& dobs);
double min_obs(dated_observations& dobs);
bool dates_match(const dated_observations& obs1, const dated_observations& obs2);

///// picking subsets.
dated_observations observations_between(const dated_observations& obs,const date& first,const date& last);
dated_observations observations_after(const dated_observations& obs, const date& first);
dated_observations observations_before(const dated_observations& obs, const date& last);
dated_observations observations_matching_dates(const dated_observations& obs, const vector<date>& dates);

///// picking periodic elements
dated_observations beginning_of_month_observations(const dated_observations&);
dated_observations end_of_month_observations(const dated_observations&);
dated_observations end_of_year_observations(const dated_observations&);
dated_observations end_of_year_current_observations(const dated_observations&);

#endif
```

Header file 1.1: dated obs h

```
#include "dated_obs.h"
#include <algorithm>

void dated_observations::set_title(string s) { title_ = s; };

void dated_observations::clear() {
   title_ = string();
   dated<double>::clear();
};

string dated_observations::title() const { return title_;};

int dated_observations::no_obs_between(const date& dat1, const date& dat2) const {
// count number of observations between given dates.
   if (!dat1.valid()) return -1;
   if (!dat2.valid()) return -1;
   int noobs=0;
   int T = size();
   for (int t=0;t<T;++t) {
      if ( (dat1<=date_at(t) ) && (date_at(t)<=dat2) ) ++noobs;
   };
   return noobs;
};

int dated_observations::no_obs_before(const date& d) const {
   return no_obs_between(first_date(),d);
};

int dated_observations::no_obs_after(const date& d) const {
   return no_obs_between(d,last_date());
};
```

C++ Code 1.1: dated obs cc

```
#include "dated_obs.h"

bool dates_match(const dated_observations& d1, const dated_observations& d2){
   if (d1.size()!=d2.size()) { return false; }
   for (unsigned int t=0;t<d1.size();++t){
      // slow, careful check that the time series match exactly
      if (d1.date_at(t) != d2.date_at(t)) return false;
   };
   return true;
};

double max_obs(dated_observations& dobs){
   vector<double> obs = dobs.elements();
   return *max_element(obs.begin(),obs.end());
};


double min_obs(dated_observations& dobs){
   vector<double> obs = dobs.elements();
   return *min_element(obs.begin(),obs.end());
};
```

C++ Code 1.2: calc

```
#include "dated_obs.h"
#include <iomanip>

void print_dated_observations(ostream& of,
                              const dated_observations& obs,
                              int precision) {
   if (obs.title().length()>0) of << obs.title() << endl;
   for (unsigned int t=0;t<obs.size();t++) {
       of << obs.date_at(t) << " "
          << setprecision(precision) << fixed << obs.element_at(t)
          << endl;
   };
};

ostream& operator << (ostream& os, const dated_observations& obs) {
    print_dated_observations(os,obs,4);
    return os;
};
```

C++ Code 1.3: io

```
#include "dated_obs.h"

dated_observations end_of_year_observations(const dated_observations& dobs) {
   if (dobs.size()<1) return dated_observations();
    dated_observations eoy_obs;
    eoy_obs.set_title(dobs.title());
    if (dobs.first_date().month()==1) { // include beginning of first year
         eoy_obs.insert(dobs.date_at(0),dobs.element_at(0));
    };
    for (int t=0;t<dobs.size()-1;++t) {
         if (dobs.date_at(t).year()!=dobs.date_at(t+1).year()) {
             eoy_obs.insert(dobs.date_at(t),dobs.element_at(t));
         };
    };
    if (eoy_obs.last_date().year() != dobs.last_date().year()) {
         eoy_obs.insert(dobs.last_date(),dobs.element_at(dobs.size()-1));
    }
    return eoy_obs;
};

dated_observations end_of_year_current_observations(const dated_observations& dobs) {
   if (dobs.size()<1) return dated_observations();
    dated_observations eoy_obs;
    eoy_obs.set_title(dobs.title());
    if (dobs.first_date().month()==1) { eoy_obs.insert(dobs.date_at(0),dobs.element_at(0)); };
    for (int year=dobs.first_date().year(); year<=dobs.last_date().year(); ++year){
         eoy_obs.insert(date(31,12,year),dobs.current_element_at(date(31,12,year)));
    };
    return eoy_obs;
}

dated_observations beginning_of_month_observations(const dated_observations& dobs) {
    if (dobs.size()<1) return dated_observations();
    dated_observations eom_obs;
    eom_obs.set_title(dobs.title());
    if (dobs.first_date().day()<5) { eom_obs.insert(dobs.date_at(0),dobs.element_at(0)); };
    for (int t=1;t<dobs.size();++t) {
         if ( (dobs.date_at(t).month()!=dobs.date_at(t-1).month())
              || (dobs.date_at(t).year()!=dobs.date_at(t-1).year()) ) {
             eom_obs.insert(dobs.date_at(t),dobs.element_at(t));
         };
    };
    return eom_obs;
}

dated_observations end_of_month_observations(const dated_observations& dobs) {
    if (dobs.size()<1) return dated_observations();
    dated_observations eom_obs;
    eom_obs.set_title(dobs.title());
    if (dobs.first_date().day()<10) { eom_obs.insert(dobs.date_at(0),dobs.element_at(0)); };
    for (int t=0;t<dobs.size()-1;++t) {
         if ( (dobs.date_at(t).month()!=dobs.date_at(t+1).month())
              || (dobs.date_at(t).year()!=dobs.date_at(t+1).year()) ) {
             eom_obs.insert(dobs.date_at(t),dobs.element_at(t));
         };
    };
    if ( (eom_obs.last_date().month() != dobs.last_date().month())
          || (eom_obs.last_date().year() != dobs.last_date().year()) ) {
         eom_obs.insert(dobs.last_date(),dobs.element_at(dobs.size()-1));
    }
    return eom_obs;
};
```

C++ Code 1.4: Periodic

```
#include "dated_obs.h"

dated_observations observations_between( const dated_observations& obs,
                                          const date& first,
                                          const date& last) {
// assume that the first and last date should be included.
    dated_observations picked = obs; // just copy and then remove. Fast enough
    picked.remove_after(last);
    picked.remove_before(first);
    return picked;
};

dated_observations observations_after( const dated_observations& obs,
                                        const date& first) {
// assume that the first date is to be included in the result
// should maybe be observations_on_and_after....
   dated_observations dobs = obs; // just copy and then remove. Fast enough
   dobs.remove_before(first);
   return dobs;
};

dated_observations observations_before( const dated_observations& obs,
                                         const date& last) {
// assume that the last date is to be included in the result
   dated_observations dobs = obs;
   dobs.remove_after(last);
   return dobs;
};
```

C++ Code 1.5: Subsets