1. Introduction.

ANALYSIS

Egg Database Analysis Tools

by John Walker
http://www.fourmilab.ch/

This program implements frequently-performed analyses of *generic_eggsummary* data, in the form of a *generic_egganalysis* class derived therefrom which adds various analytic tools and the storage to represent them and support their computation. (The "generic" stuff refers to the fact that all of these classes are defined as templates in which the data type used for the egg sample data is parametric. This permits analyses which wish to aggregate data or express it in other measures, for example, z scores, to avoid the round-off which forcing the egg sample to its default value of **unsigned char** would entail. We classify statistics computed as "all egg", where we produce time series which evaluate quantities at a moment of time across all eggs reporting, and "per egg", where we measure the behaviour of individual eggs across the entire time period in the data set.

To use this code, you'll need to be familiar with the `eggdata` program; you'll probably use the cache to obtain your data sets and prepare extracts of time intervals of interest, and you'll need to understand the *generic_eggsummary* class to access raw data. You may also wish to consult the `timedate` program to manipulate dates and times, and the `statlib` program for statistical tests on the data.

In general, you can think of the development of analysis software as a process of migration of tools from specific purpose-built analysis programs such as those presented in `examples`, into this program as they become more widely used, and finally into `eggdata` if they are fundamentally applicable to the original data set.

⟨ `analysis_test.c` 1 ⟩ ≡
```
#define REVDATE  "4th␣February␣2002"
```
See also section 23.

2. Program global context.

```
#include "config.h"     /* System-dependent configuration */
  ⟨ Preprocessor definitions ⟩
  ⟨ Application include files 4 ⟩
  ⟨ Class implementations 6 ⟩
```

3. We export the class definitions for this package in the external file `analysis.h` that programs which use this library may include.

```
⟨ analysis.h 3 ⟩ ≡
#ifndef ANALYSIS_HEADER_DEFINES
#define ANALYSIS_HEADER_DEFINES
#include <math.h>     /* Make sure math.h is available */
#include <iostream>
#include <iomanip>
#include <fstream>
#include <exception>
#include <stdexcept>
#include <string>
#include <vector>
#include <map>
  using namespace std;
#include <ctype.h>
#include <assert.h>
#include "eggdata.h"
#include "timedate.h"
#include "statlib.h"
  ⟨ Class definitions 8 ⟩
#endif
```

4. The following include files provide access to external components of the program not defined herein.

```
⟨ Application include files 4 ⟩ ≡
#include "analysis.h"     /* Class definitions for this package */
```

This code is used in section 2.

5. This is a dummy placeholder for code to be compiled into the binary part of the program. At the moment, we have only template classes, so we'd get an error on the reference absent this definition of the code portion. If we end up exclusively with templates, this definition and the reference below may be removed.

```
⟨ Egg analysis utilities 5 ⟩ ≡
```

This code is used in section 6.

6. The following classes are defined and their implementations provided.

```
⟨ Class implementations 6 ⟩ ≡
  ⟨ Egg analysis utilities 5 ⟩
```

This code is used in section 2.

7. Egg analysis utilities.

8. The Egg Analysis Class.

The *egganalysis* extends *eggsummary* to provide a variety of analyses of the raw sample data and output thereof. The intention is that *egganalysis* will implement the standard "workhorse" statistics, then be further extended to add exploratory measures. We "slice" the raw data two ways here: as a time series across all eggs reporting in a given time period and per-egg—looking at the behaviour of each egg over the entire time period in the data set.

As with the parent class, we define this as a template, *generic_egganalysis*, which can be instantiated with whatever data type is required for the individual egg samples. A default instantiation with **unsigned char** is provided via the **typedef** *egganalysis*.

```
⟨ Class definitions 8 ⟩ ≡
  template⟨class Sample⟩ class generic_egganalysis : public generic_eggsummary < Sample > {
public:
  double *mean, *variance, *stouffer_z;
  double *egg_mean, *egg_stdev;
  Sample *egg_min_sample, *egg_max_sample;
  int *egg_num_sample;
  int *nsamples; private:
  void free_all_egg_analysis_data(void){
#define ReleaseTable(x)
      if ((x) ≠ Λ) {
        delete (x);
        (x) = Λ;
      }
      ReleaseTable(mean);
      ReleaseTable(variance);
      ReleaseTable(stouffer_z);
      ReleaseTable(nsamples);
#undef ReleaseTable
      } void free_per_egg_analysis_data(void){
#define ReleaseTable(x)
        if ((x) ≠ Λ) {
          delete (x);
          (x) = Λ;
        }
        ReleaseTable(egg_mean);
        ReleaseTable(egg_stdev);
        ReleaseTable(egg_min_sample);
        ReleaseTable(egg_max_sample);
        ReleaseTable(egg_num_sample);
#undef ReleaseTable
        }
      public:
        generic_egganalysis( )
        {
          mean = variance = stouffer_z = Λ;
          nsamples = Λ;
          egg_mean = egg_stdev = Λ;
          egg_min_sample = egg_max_sample = Λ;
          egg_num_sample = Λ;
        }
        void write_egg_statistics_CSV(string fileName);
        void write_per_egg_statistics_CSV(string fileName);
```

```
void write_deviation_plot(string fileName, string chartTitle = "", int timeInterval = 1, int
    tickOffset = 0, string x_axis_label = "", double referenceProbability = 0.05, systemtime
    referenceProbabilityStartTime = systemtime(0));
~generic_egganalysis( )
{
  free_all_egg_analysis_data( );
  free_per_egg_analysis_data( );
}
void compute_all_egg_statistics(void);
void compute_per_egg_statistics(void); } ;
typedef generic_egganalysis < unsigned char > egganalysis;
```

See also sections 9, 10, 16, 17, and 18.

This code is used in section 3.

9. Here we compute a time series for each time interval in the database, aggregating data across all the eggs (assuming they are synchronised). We compute:

`nsamples`	Number of eggs reporting in this second
`mean`	Mean value across all eggs reporting in this second
`stouffer_z`	Stouffer Z for eggs reporting

Let v_e be the sample reported by egg e for a given second, n the number of eggs which reported values this second, μ and σ the mean and standard deviation, respectively, of the samples produced by the eggs. Then the values tabulated are as follows:

`nsamples`	n
`mean`	$\frac{\sum_{e=0}^{n} v_e}{n}$
`stouffer_z`	$\frac{\sum_{e=0}^{n} \text{mean}-\mu}{\sigma/\sqrt{n}}$

```
⟨ Class definitions 8 ⟩ +≡
  template⟨class Sample⟩ void generic_egganalysis⟨Sample⟩::compute_all_egg_statistics(void){
        assert ( generic_eggsummary < Sample > ::seconds_per_row ) ;
      /* If this pops, you probably forgot to load the data */
    int nitems = generic_eggsummary < Sample > ::seconds_of_data/generic_eggsummary <
        Sample > ::seconds_per_row;
    free_all_egg_analysis_data( );
    mean = new double[nitems];
    variance = new double[nitems];
    stouffer_z = new double[nitems];
    nsamples = new int[nitems];
    int i; for (i = 0; i < nitems; i++) { int j;
    double rsum = 0, zsum = 0;
    int n = 0; for (j = 0; j < generic_eggsummary < Sample > ::eggs_reporting; j++ ) { if
        ( isSampleValid ( generic_eggsummary < Sample > ::get_egg_index(i, j) ) ) { n++;
        rsum += generic_eggsummary < Sample > ::get_egg_index(i, j); } } nsamples[i] = n;
    mean[i] = rsum/n; stouffer_z[i] = ( mean[i] - ( generic_eggsummary < Sample > ::trial_size/2 ) )
        / ( sqrt ( generic_eggsummary < Sample > ::trial_size/4.0 ) / sqrt(static_cast⟨double⟩(n)) )
        ;
    if (isnan(stouffer_z[i]) ∧ (i > 0)) {
      stouffer_z[i] = stouffer_z[i − 1];
    }
    } }
```

10. Slicing the data set the other way, by egg rather than time, yields statistics on the behaviour of individual eggs over the time period in the database. These statistics are largely irrelevant to hypotheses being tested by the GCP, but facilitate studies of peculiarities of individual or classes of random event generators which may influence the composite result. For each egg, we compute for all n valid samples s_x:

`egg_num_sample`	n
`egg_mean`	$\mu = \frac{\sum_{i=0}^{n} s_i}{n}$
`egg_stdev`	$\sigma = \sqrt{\frac{\sum_{i=0}^{n} (s_i - \mu)^2}{n-1}}$
`egg_min_sample`	$\min_{1 \le i \le n} s_i$
`egg_max_sample`	$\max_{1 \le i \le n} s_i$

⟨ Class definitions 8 ⟩ +≡
```
template⟨class Sample⟩ void generic_egganalysis⟨Sample⟩::compute_per_egg_statistics(void)
{
  ⟨ Allocate and initialise per egg statistics tables 11 ⟩;
  ⟨ Compute per egg sum, maximum, and minimum values 12 ⟩;
  ⟨ Compute per egg mean and standard deviation 13 ⟩;
}
```

11. We begin computation of the per-egg statistics by allocating arrays for the results, one slot per egg and initialise them appropriately for the subsequent computations. If previously allocated tables are present, they are released.

⟨ Allocate and initialise per egg statistics tables 11 ⟩ ≡
```
free_per_egg_analysis_data( );  egg_mean = new double [ generic_eggsummary <
    Sample > ::eggs_reporting ] ;  egg_stdev = new double [ generic_eggsummary <
    Sample > ::eggs_reporting ] ;  egg_min_sample = new Sample [ generic_eggsummary <
    Sample > ::eggs_reporting ] ;  egg_max_sample = new Sample [ generic_eggsummary
    < Sample > ::eggs_reporting ] ;  egg_num_sample = new int [ generic_eggsummary <
    Sample > ::eggs_reporting ] ;      /* Initialise variables for per-egg statistics */
int i; int nitems = generic_eggsummary < Sample > ::seconds_of_data/generic_eggsummary <
    Sample > ::seconds_per_row; for (i = 0; i < generic_eggsummary < Sample > ::eggs_reporting;
    i++ ) { egg_mean[i] = egg_stdev[i] = 0; egg_min_sample[i] = generic_eggsummary <
    Sample > ::trial_size + 1;
egg_max_sample[i] = 0;
egg_num_sample[i] = 0; }
```
This code is used in section 10.

12. Next, we make a first pass over the data, accumulating the sum (to be used later to obtain the mean value), minimum, and maximum of valid samples from that egg. Along the way, we count the number of valid samples from each egg present in the data set.

```
⟨ Compute per egg sum, maximum, and minimum values 12 ⟩ ≡
  for (i = 0; i < generic_eggsummary < Sample > ::eggs_reporting; i++ ) { int j; for (j = 0; j < nitems;
      j++) { Sample es = generic_eggsummary < Sample > ::get_egg_index(j, i);
  if (isSampleValid(es)) {
    egg_num_sample[i]++;
    egg_mean[i] += es;
    if (es < egg_min_sample[i]) {
      egg_min_sample[i] = es;
    }
    if (es > egg_max_sample[i]) {
      egg_max_sample[i] = es;
    }
  }
  } }
```

This code is used in section 10.

13. Finally, we can compute the mean and standard deviation for each egg, the mean simply by dividing the sample sum computed in the first pass by the number of valid samples, and the standard deviation by making a second pass over the data now that the mean is known. Eggs which contributed no valid samples are ignored in this step and their mean and standard deviation set to zero. The standard deviation is also set to zero if an egg contributes only a single sample.

```
⟨ Compute per egg mean and standard deviation 13 ⟩ ≡
  for (i = 0; i < generic_eggsummary < Sample > ::eggs_reporting; i++ ) { int j; if
      (egg_num_sample[i] > 1) { egg_mean[i] /= egg_num_sample[i]; for (j = 0; j < nitems; j++) {
      Sample es = generic_eggsummary < Sample > ::get_egg_index(j, i);
  if (isSampleValid(es)) {
    egg_stdev[i] += (es − egg_mean[i]) ∗ (es − egg_mean[i]);
  }
  } egg_stdev[i] = sqrt(egg_stdev[i]/(egg_num_sample[i] − 1)); } }
```

This code is used in section 10.

14. As a convenience, output routines which depend on the all-egg statistics having been computed test whether the statistics have been computed and, if not, perform this task automatically.

```
⟨ Compute all egg statistics if not previously done 14 ⟩ ≡
  if (mean ≡ Λ) {
    compute_all_egg_statistics( );
  }
```

This code is used in sections 16 and 18.

15. And here's the same convenience for folks interested in the per-egg statistics.

```
⟨ Compute per egg statistics if not previously done 15 ⟩ ≡
  if (egg_mean ≡ Λ) {
    compute_per_egg_statistics( );
  }
```

This code is used in section 17.

16. This method writes a `.csv` file containing a time series of the all-egg statistics for the time period. **Need to include a table of fields in this file once it stops changing.**

```
⟨ Class definitions 8 ⟩ +≡
  template⟨class Sample⟩ void generic_egganalysis⟨Sample⟩::write_egg_statistics_CSV(string
      fileName){ FILE *of = fopen(fileName.c_str(), "w");
    double ichisum = 0, chisum = 0; int nitems = generic_eggsummary < Sample >
      ::seconds_of_data/generic_eggsummary < Sample >::seconds_per_row;
    int i, idof = 0;
    ⟨ Compute all egg statistics if not previously done 14 ⟩;
    for (i = 0; i < nitems; i++) { double zsquare = stouffer_z[i] * stouffer_z[i];
    double xe = chiSquareDistribution::x_from_p_k(0.99999, i + 1);
    chisum += zsquare;
    idof++; fprintf (of, "24,%lu,%d,%.6f,%.6f,%.6f,%d,%.6f,%.6f\n", generic_eggsummary <
      Sample >::start_time.get_time() + ( i * generic_eggsummary < Sample >::seconds_per_row
      ) , nsamples[i], mean[i], stouffer_z[i], chisum, idof, xe, (chisum − xe)/xe,
      chiSquareDistribution::p_from_k_x(chisum, i + 1) ) ; } fclose(of); }
```

17. The *write_per_egg_statistics_CSV* method creates a `.csv` file containing the per-egg statistics calculated by *compute_per_egg_statistics*. The statistics will be calculated automatically if they haven't already been prepared. The `.csv` file contains one record for each egg which contributed samples to the database, formatted as follows:

`25,`*Egg number*`,`*Number of samples*`,`*Mean*`,`*Max*`,`*Min*`,`*Stdev*

where `25` is the record type code, *Egg number* the egg number, *Number of samples* the number of valid samples from this egg present in the data set, *Mean*, *Max*, and *Min* the arithmetic mean value of all samples, and the maximum and minimum values present, and *Stdev* the standard deviation of the samples from the egg. Eggs whose data have been entirely discarded as invalid are not listed in the report.

```
⟨ Class definitions 8 ⟩ +≡
  template⟨class Sample⟩ void generic_egganalysis⟨Sample⟩::write_per_egg_statistics_CSV(string
      fileName){ ofstream of(fileName.c_str());
    ⟨ Compute per egg statistics if not previously done 15 ⟩;
    of ≪ setprecision(8); for (int i = 0; i < generic_eggsummary < Sample >::eggs_reporting;
      i++ ) { if (egg_num_sample[i] > 0) { of ≪ "25," ≪ generic_eggsummary < Sample >
      ::egg_number[i] ≪ "," ≪ egg_num_sample[i] ≪ "," ≪ egg_mean[i] ≪ "," ≪ ((double)
      egg_max_sample[i]) ≪ "," ≪ ((double) egg_min_sample[i]) ≪ "," ≪ egg_stdev[i] ≪ endl; } }
      of.close(); }
```

18. The *write_deviation_plot* method generates, with the able assistance of `GNUPLOT`, a chart of the cumulative deviation the time span in the object. A comparison line to a given *referenceProbability* (0.05 by default) can be plotted as well. If a *referenceProbability* of zero is specified, no reference line is plotted. You can offset the reference probability curve with respect to the data set by specifying a *referenceProbabilityStartTime* for the curve; the curve is plotted starting at the first time at or after the specified moment, with this Y axis origin at the value of the all-egg deviation curve at that moment. Specifying a negative value for the *referenceProbability* causes the curve to be plotted below the expectation value curve.

```
⟨ Class definitions 8 ⟩ +≡
  template⟨class Sample⟩ void generic_egganalysis⟨Sample⟩::write_deviation_plot(string
      fileName, string chartTitle, int timeInterval, int tickOffset, string x_axis_label, double
      referenceProbability, systemtime referenceProbabilityStartTime){ ofstream
      gp((fileName + ".gp").c_str( )), dat((fileName + ".dat").c_str( ));
    ⟨ Compute all egg statistics if not previously done 14 ⟩;
    ⟨ Write GNUPLOT data table for deviation plot 19 ⟩;
      /* Create GNUPLOT instructions to plot data */
    double final_p = chiSquareDistribution::p_from_k_x(idof, chisum);
    gp << "set term pbm small color" << endl;
    ⟨ Generate chart title for deviation plot 20 ⟩;
    gp << "set ylabel \"Cumulative Deviation" << endl;
    ⟨ Generate label for X (time) axis of deviation plot 21 ⟩;
    if (timeInterval ≠ 1) {
      ⟨ Generate X axis tick labels for deviation plot 22 ⟩;
    }
    gp << "plot \"" << fileName << ".dat\" using 1:2 title \"Composite: " <<
        generic_eggsummary < Sample > ::eggs_reporting << " eggs; p = " << setprecision(3) <<
        (1.0 − final_p) << "\" with lines,\\" << endl;
    if (referenceProbability ≠ 0) {
      gp << "     \"" << fileName << ".dat\" using 1:3 title \"p=" << referenceProbability <<
          "\" with lines,\\" << endl;
    }
    gp << "     0 title \"\"" << endl;
    string command("gnuplot ");
    command += fileName + ".gp | ppmtogif > " + fileName + ".gif";
#ifdef DEV_PLOT_DEBUG
    cout << command << endl;
#else
    command += " 2>/dev/null";
#endif
    gp.close( );
    dat.close( );
    system(command.c_str( ));
#ifndef DEV_PLOT_DEBUG      /* Delete the temporary files used to create the plot */
    remove((fileName + ".gp").c_str( ));
    remove((fileName + ".dat").c_str( ));
#endif
    }
```

19. The `GNUPLOT` data file for the deviation plot consists of three columns: the time interval index (used for the x axis), the cumulative deviation, and the reference probability curve (if *referenceProbability* $\neq 0$).

```
⟨ Write GNUPLOT data table for deviation plot 19 ⟩ ≡
  double ichisum = 0, chisum = 0; int nitems = generic_eggsummary < Sample >
      ::seconds_of_data/generic_eggsummary < Sample > ::seconds_per_row;
  int i, ref = 0, idof = 0;
  double cxebase;
  bool firstref = true; for (i = 0; i < nitems; i++) { double zsquare = stouffer_z[i] * stouffer_z[i];
  double xe = chiSquareDistribution::x_from_p_k(0.5, i + 1);
  chisum += zsquare;
  idof ++;
  dat ≪ i ≪ "␣" ≪ (chisum − xe); if ( ( generic_eggsummary < Sample > ::start_time.get_time( ) + (
      generic_eggsummary < Sample > ::seconds_per_row * i ) ) ≥ referenceProbabilityStartTime.get_time( )
      )
  {
    double xpRef = chiSquareDistribution::x_from_p_k(fabs(referenceProbability), ref + 1);
    if (firstref) {
      cxebase = chisum − xe;
      firstref = false;
    }
    dat ≪ "␣" ≪ (((referenceProbability < 0) ? −1 : 1) * ((chiSquareDistribution::x_from_p_k(0.5,
        ref + 1) − xpRef) + cxebase));
    ref ++;
  }
  dat ≪ endl; }
```

This code is used in section 18.

20. If the user doesn't supply a custom title for the chart, generate one from the start and end times of the data plotted. If the chart covers precisely one day, the default title is just the date. If the start and end times are in the same day but not the first and last seconds of it, the date is given only before the start time.

```
⟨ Generate chart title for deviation plot 20 ⟩ ≡
  if (chartTitle ≡ "") { chartTitle = generic_eggsummary < Sample > ::start_time.dateToString( );
      if ( ( generic_eggsummary < Sample > ::start_time.get_time( ) ≠ generic_eggsummary <
      Sample > ::start_time.midnight( ) ) ∨ ( ( generic_eggsummary < Sample > ::end_time.get_time( )+1
      ) ≠ ( generic_eggsummary < Sample > ::start_time.get_time( ) + systemtime::SecondsPerDay ) )
      ) { chartTitle += "␣" + generic_eggsummary < Sample > ::start_time.timeToString( );
  chartTitle += "␣--␣"; if ( generic_eggsummary < Sample >
      ::start_time.midnight( ) ≠ generic_eggsummary < Sample > ::end_time.midnight( ) )
      { chartTitle += generic_eggsummary < Sample > ::end_time.dateToString( ) + "␣";
      } chartTitle += generic_eggsummary < Sample > ::end_time.timeToString( ); } }
      gp ≪ "set␣title␣\"" ≪ chartTitle ≪ "\"" ≪ endl;
```

This code is used in section 18.

21. The caller can supply the label for the x (time) axis of the deviation plot explicitly by passing the desired label as the *x_axis_label* argument (or a single blank for no label at all). If the null string is supplied (the default if the argument is omitted), a label will be generated based on the *timeInterval* argument. If the *timeInterval* is not one of the intervals defined below, the axis will be labeled as "Time Periods"—you should provide a custom label along with a nonstandard time interval.

```
⟨ Generate label for X (time) axis of deviation plot 21 ⟩ ≡
  if (x_axis_label ≠ "␣") {
    if (x_axis_label ≡ "") {
      switch (timeInterval) {
      case 1: x_axis_label = "Seconds";
        break;
      case systemtime::SecondsPerMinute: x_axis_label = "Minutes";
        break;
      case systemtime::SecondsPerHour: x_axis_label = "Hours";
        break;
      case systemtime::SecondsPerDay: x_axis_label = "Days";
        break;
      default: x_axis_label = "Time␣Periods";
        break;
      }
      x_axis_label += "␣Elapsed";
    }
    gp ≪ "set␣xlabel␣\"" ≪ x_axis_label ≪ "\"" ≪ endl;
  }
```

This code is used in section 18.

22. If the *timeInterval* is one second (the default), GNUPLOT's default x axis labeling is adequate. Otherwise, we must generate a "`set␣xtics`" statement which supplies the labels and gives their locations along the axis. We place labels at each *timeInterval*, offset *tickOffset* seconds, which may be positive or negative.

```
⟨ Generate X axis tick labels for deviation plot 22 ⟩ ≡
  int ctime = generic_eggsummary < Sample > ::start_time.get_time( ) + tickOffset;
  bool first = true;
  gp ≪ "set␣xtics␣(␣\\" ≪ endl;
  for (i = 0; i < nitems; i++) {
    if ((ctime % timeInterval) ≡ 0) {
      if (¬first) {
        gp ≪ ",\\" ≪ endl;
      }
      first = false;
      gp ≪ "\"" ≪ ((i + tickOffset)/timeInterval) ≪ "\"␣" ≪ i;
    }
    ctime++;
  }
  gp ≪ "\\" ≪ endl ≪ "␣␣␣␣)" ≪ endl;
```

This code is used in section 18.

23. Test program.

```
⟨ analysis_test.c 1 ⟩ +≡
  ⟨ Test program include files 26 ⟩;
  ⟨ Show how to call test program 25 ⟩;
  int main(int argc, char *argv[]){ extern char *optarg;      /* Imported from getopt */
      extern int optind; try { int opt;
      ⟨ Process command-line options 24 ⟩;
#if 0     /* Eggsummary analysis and plotting tests */
      { int j;
      eggdatabases ed;
      ed.set_Fourmilab_defaults( );
#ifndef CACHE_TEST
#define DEFAULT_EGGANALYSIS
#ifdef DEFAULT_EGGANALYSIS
      generic_eggsummary_cache < egganalysis > ec(&ed, "rotten_egg.csv", 50, 150);
      egganalysis * esr;
      egganalysis ext;
#else
      typedef generic_egganalysis⟨double⟩ double_egganalysis; generic_eggsummary_cache <
          double_egganalysis > ec(&ed, "rotten_egg.csv", 50, 150);
      double_egganalysis *esr;
      double_egganalysis ext;
#endif
      esr = ec.get_by_date("2001-04-22");
      systemtime refstart;      /* refstart.fromString("2001-04-22 08:00:00"); */
        /* refstart.fromString("2001-04-22 00:00:00"); */
#ifndef EXTRACT_TEST
      systemtime exstart, exend;
      exstart.fromString("2001-04-22␣13:00:00");
      exend.fromString("2001-04-22␣22:30:00");
      esr→extract_time_range(&ext, exstart, exend);
      esr = &ext;
#endif
      esr→compute_per_egg_statistics( );
      esr→describe( );
      esr→write_per_egg_statistics_CSV("peregg.csv");
      esr→write_deviation_plot("devtest",      /* Name of plot file */
      "",      /* Custom title for chart */
      systemtime::SecondsPerHour,      /* Time interval for x axis labels */
      systemtime::SecondsPerHour * 8,      /* Offset in seconds for x axis labels */
      "",      /* Custom title for x axis */
      0.05,      /* Probability for reference curve */
      refstart      /* Time to begin reference curve plot */
      );
#else
      egganalysis esr;
      systemtime t;
      t.fromString("2001-04-22");      /* t.fromString("2001-11-29"); */
      esr.load_from_CSV(ed.database_file(t));
      esr.exclude_bad_data("rotten_egg.csv");
```

```
    esr.limit_to_range(50, 150, &cout);
    esr.compute_all_egg_statistics( );
#if 0
    for (j = 0; j < esr.eggs_reporting; j++) {
      cout << esr.egg_number[j] << ": " << (int) esr.get_egg_index(86399, j) << "\n";
    }
#endif
#if 0
    esr.write_egg_statistics_CSV("test_24.csv");
#endif
#ifndef SIMPLE
    esr.write_deviation_plot("devtest",      /* Name of plot file */
    "",      /* "Earth Day: 2001 April 22", // Custom title for chart */
    systemtime::SecondsPerHour,      /* Time interval for x axis labels */
    systemtime::SecondsPerHour * 8,      /* Offset in seconds for x axis labels */
    "",      /* Custom title for x axis */
    0.05      /* Probability for reference curve */
    );
#else
    esr.write_deviation_plot("devtest",      /* Name of plot file */
    "Earth Day: 2001 April 22",      /* Custom title for chart */
    systemtime::SecondsPerHour      /* Time interval for x axis labels */
    );
#endif
#endif
    }
#endif
#if 1      /* Multi-day extraction and analysis tests */
    {
      eggdatabases ed;
      ed.set_Fourmilab_defaults( );
      generic_eggsummary_cache < egganalysis > ec(&ed, "rotten_egg.csv", 50, 150);
      egganalysis ext;
      systemtime refstart;
      systemtime exstart, exend;
      exstart.fromString("2001-04-20 13:00:00");
      exend.fromString("2001-04-24 22:30:00");
      ec.extract_time_range(&ext, exstart, exend);
      ext.compute_per_egg_statistics( );
      ext.describe( );
      ext.write_per_egg_statistics_CSV("peregg.csv");
      ext.write_deviation_plot("devtest",      /* Name of plot file */
      "",      /* Custom title for chart */
      systemtime::SecondsPerHour,      /* Time interval for x axis labels */
      systemtime::SecondsPerHour * 8,      /* Offset in seconds for x axis labels */
      "",      /* Custom title for x axis */
      0.05,      /* Probability for reference curve */
      refstart      /* Time to begin reference curve plot */
      );
    }
#endif
    } catch(exception &e)
```

```
    {
      cout ≪ "Blooie!!!␣␣Exception␣popped:␣" ≪ e.what( ) ≪ endl;
#ifndef CORE_DUMP
#ifdef STACK_TRACE
      char s[160];
      sprintf(s,
          "/bin/echo␣'where\nq'␣>/tmp/gdbcmd␣;␣gdb␣␣-batch␣--command␣""/tmp/gdbcmd␣%s␣%d",
          argv[0], getpid( ));
      system(s);
      sleep(5);
#endif
      throw;      /* Re-throw exception to dump core */
#endif
    }
    return 0; }
```

24. We use *getopt* to process command line options. This permits aggregation of options without arguments and both `-d`*arg* and `-d` *arg* syntax.

```
⟨ Process command-line options 24 ⟩ ≡
  while ((opt = getopt(argc, argv, "nu-:")) ≠ −1) {
    switch (opt) {
    case 'u':     /* -u Print how-to-call information */
      case '?': usage( );
      return 0;
    case '-':     /* -- Extended options */
      switch (optarg[0]) {
      case 'c':     /* --copyright */
        cout ≪ "This␣program␣is␣in␣the␣public␣domain.\n";
        return 0;
      case 'h':     /* --help */
        usage( );
        return 0;
      case 'v':     /* --version */
        cout ≪ PRODUCT ≪ "␣" ≪ VERSION ≪ "\n";
        cout ≪ "Last␣revised:␣" ≪ REVDATE ≪ "\n";
        cout ≪ "The␣latest␣version␣is␣always␣available\n";
        cout ≪ "at␣http://www.fourmilab.ch/eggtools/eggshell\n";
        return 0;
      }
    }
  }
```

This code is used in section 23.

25. Procedure *usage* prints how-to-call information.

```
⟨ Show how to call test program 25 ⟩ ≡
  static void usage(void)
  {
    cout ≪ PRODUCT ≪ "␣␣--␣␣Analyse␣eggsummary␣files.␣␣Call:\n";
    cout ≪ "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣" ≪ PRODUCT ≪ "␣[options]␣[infile]␣[outfile]\n";
    cout ≪ "\n";
    cout ≪ "Options:\n";
    cout ≪ "␣␣␣␣␣␣␣␣␣␣␣--copyright␣␣␣␣␣␣␣Print␣copyright␣information\n";
    cout ≪ "␣␣␣␣␣␣␣␣␣␣␣-u,␣--help␣␣␣␣␣␣␣␣Print␣this␣message\n";
    cout ≪ "␣␣␣␣␣␣␣␣␣␣␣--version␣␣␣␣␣␣␣␣␣Print␣version␣number\n";
    cout ≪ "\n";
    cout ≪ "by␣John␣Walker\n";
    cout ≪ "http://www.fourmilab.ch/\n";
  }
```

This code is used in section 23.

26. We need the following definitions to compile the test program.

```
⟨ Test program include files 26 ⟩ ≡
#include "config.h"      /* Our configuration */      /* C++ include files */
#include <iostream>
#include <exception>
#include <stdexcept>
#include <string>
  using namespace std;
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#ifdef HAVE_GETOPT
#ifdef HAVE_UNISTD_H
#include <unistd.h>
#endif
#else
#include "getopt.h"      /* No system getopt—use our own */
#endif
#include "analysis.h"      /* Class definitions for this package */
```

This code is used in section 23.

27. Index. The following is a cross-reference table for `analysis`. Single-character identifiers are not indexed, nor are reserved words. Underlined entries indicate where an identifier was declared.

ANALYSIS