1. Introduction.

EXAMPLES

Examples of Egg Data Analysis

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

Almost everything I know of the craft of programming I learned *reading code*—I remember one Sunday morning when I was in college, with The Fifth Dimension record on the stereo and a listing of "The Collector"—the linker for UNIVAC EXEC-8 open on the desk in front of me, thinking "It doesn't get any better." Almost nobody reads code any more, and consequently those who write code rarely bother with the æsthetic niceties which render it readable, closing the vicious circle of "write-only software". One of the principal goals of Literate Programming is to encourage the reading of code by making code *more readable*: providing tools to assist those writing code in making it attractive and accessible to readers.

Rather than write a separate document for users of the analysis suite I've developed for the Global Consciousness Project data set, I've tried to make the programs themselves readable, with the following set of examples as the port of entry. Each of the following examples performs an analysis of genuine interest on the data set using the analysis toolkit. Every example produces a stand-alone executable program you can run and examine its output. The examples are all self-contained—you can read their code from start to end without reference to anything else other than simple application "glue" which is shared by all the programs. There is some duplication of code among examples, but I believe that's justified to make each readable in isolation.

While the examples show how a variety of tasks are accomplished with the toolkit, they only provide a glimpse of the underlying tools they employ. To explore further, you'll want to look at the the following programs, listed in more or less decreasing order of abstraction from the nitty-gritty.

analysis This program implements higher level analyses of egg data sets, either original one day summaries or arbitrary time spans extracted from the collection of daily summaries.

eggdata Provides tools for reading both egg data tables and auxiliary databases, such as the properties of individual eggs, known bad data which should be excluded from analyses, etc., with facilities for extracting and assembling data sets for analysis.

fourier Tools for computing Fourier and inverse Fourier transforms of data sets and determining power spectra from frequency domain information. **Note:** I am not an expert in this domain, and anybody using this code would be well-advised to look closely at its implementation and test it with known data before relying on its results.

statlib General purpose statistical library, which includes both tools for computing various probability distributions as well as descriptive statistics on data tables with a user-defined type.

timedate Facilities for working with times and dates, using UNIX **time_t** quantities as the underlying type. Conversion to and from string representations, Julian day numbers, and computation of astronomical quantities such as sidereal time, the phase of the Moon, and the position of the Sun and Moon are provided.

colour Tools for manipulating colours and colour spaces, both physical and perceptual. This program isn't currently used by any of the examples; it was implemented for eventual use in stand-alone graphics generation (eliminating the need for GNUPLOT post-processing, but may prove useful in "artistic" presentations of the data set.

Although the examples are independent of one another, it's best to read them in the order they're presented, since in-depth explanation of each facility is provided only at its first use.

If you develop innovative analyses using these tools, please consider making verions which conform to the conventions of this document and submitting them as additional examples. There's nothing liked "worked problems" to enable and encourage others to tackle problems so far unexplored.

```
#define REVDATE "22nd␣February␣2002"
```

2. Let's define the names of our auxiliary databases right up front so if they change we don't have to fix all the examples individually.

```
#define EGGS_CSV "eggs.csv"     /* Egg properties table */
#define ROTTEN_CSV "rotten_egg.csv"     /* Known bad data table */
```

3. Example 1: Earth Day 2001.

Let's start with a very simple example which shows how easy it is to perform routine analyses and prepare graphical presentations of them. We'll take advantage of many defaults to simplify the code; in subsequent examples we'll delve more deeply into the flexibility additional parameters can provide.

The task in this example is to produce a cumulative deviation plot for "Earth Day," April 22nd 2001. We'll plot the entire 24 hour period for all eggs which reported during that day, with egg samples aligned by the Universal Time at which they were taken. For each second's data, the samples for each egg are expressed as a z score and the Stouffer Z is computed for each one second period. This is then plotted, along with the $p = 0.05$ expectation value and a zero baseline. Samples from known bad eggs and samples which fail "sanity check" limits (below 50 or above 150) are excluded from the analysis.

Since this is the first example, the explanation of what's going on will be even more prolix than usual.

4. Every UNIX program requires a modicum of "glue" to deal with the operating system and C++ library. To avoid repeating this for every example, we've created a standard prologue and epilogue for all of the examples which are included by reference, wrapped around the code which actually does the work. A program containing just the prologue and epilogue is a well-formed executable program which does nothing at all. If you're interested in the details of this, please refer to the "Common Framework for Example Programs" near the end of this document.

⟨ `example-1.c` 4 ⟩ ≡
  ⟨ Prologue for example program 47 ⟩;

See also sections 5, 6, 7, 8, and 9.

5. First of all, we need to tell the analysis suite where to look for the `eggsummary` CSV files we're going to analyse. The directory paths for both the real and pseudorandom mirror data are supplied by the *eggdatabases* class defined in `eggdata`. You can supply the explicit path names when you create the object, but here we rely on a gimmick which sets the correct path names at the existing GCP data set mirror sites based on the host name determined when the program was configured; this allows compiling analysis programs for use on any of the sites without the need for any source code changes. The analysis code assumes that `eggsummary` and pseudorandom data files for all days are kept in two separate directories. The files may be compressed with `gzip`.

⟨ `example-1.c` 4 ⟩ +≡
  eggdatabases ed;
  ed.set_local_defaults( );

6. With the egg database directories identified, we can now create an *egg summary cache* to mediate our interaction with the data set. One needn't use the cache—you're free to call lower level functions and load files directly, but the cache keeps track of which days' data are in memory to avoid redundant loading of data from disc, and can automatically exclude bad data so you don't forget. The cache is implemented as a C++ template class *generic_eggsummary_cache* in the `eggdata` program, which we instantiate here with an *egganalysis* object defined in the `analysis` program. *egganalysis*, in turn, is an extension of the *eggsummary* class defined in `eggdata` which adds analytical functionality.

When creating the cache, we supply the database location via the *eggdatabases* object, the name of the file containing data known to be bad, and lower and upper limits outside which samples are automatically discarded. Any data returned by the cache will have bad samples removed.

⟨ `example-1.c` 4 ⟩ +≡
  generic_eggsummary_cache < *egganalysis* > *ec*(&*ed*, `ROTTEN_CSV`, 50, 150);

7. Since we're analysing an entire day's data, we can simply request it from the cache by date. By default, the cache returns actual data from the eggs; a simple change when it's declared will cause the pseudorandom data to be returned instead. The cache returns a pointer to the *egganalysis* object for the given day, which we can give in a variety of forms: here we'll specify it as a string.

```
⟨ example-1.c 4 ⟩ +≡
  egganalysis * esr = ec.get_by_date("2001-04-22");
```

8. The *egganalysis* object itself has all functionality we need to compute the cumulative deviation and plot it. The following call computes statistics for each second in the day, then writes a plot file named "example-1.gif" of the results with the time axis labeled in hours since midnight. We supply a custom title for the chart.

```
⟨ example-1.c 4 ⟩ +≡
  esr→write_deviation_plot("example-1",      /* Name of plot file */
  "Earth␣Day:␣2001␣April␣22",      /* Custom title for chart */
  systemtime::SecondsPerHour      /* Time interval for x axis labels */
  );
```

9. That's it, except for the other book-end: the epilogue common to all the examples.

```
⟨ example-1.c 4 ⟩ +≡
  ⟨ Epilogue for example program 48 ⟩;
```

10. Example 2: Earth Day 2001—pseudorandom control.

Now let's compare the results for Earth Day obtained from the actual egg network with pseudorandom data replacing the samples from the eggs. As each day's summary is prepared, a pseudorandom mirror is generated in which each sample provided by an egg is replaced by one generated by a high quality pseudorandom number generator with statistics equal to the chance expectation of the random event generators used by the egg sites. We start precisely as before.

```
⟨ example-2.c 10 ⟩ ≡
  ⟨ Prologue for example program 47 ⟩;
  eggdatabases ed;
  ed.set_local_defaults( );
```

See also sections 11 and 12.

11. This time, when creating the cache, we add an argument to specify that we're interested in the pseudorandom data. (In the example above, this argument was omitted, which causes it to default to `"gcp"`, the identifier for the genuine egg data. The same bad data limits are used.

```
⟨ example-2.c 10 ⟩ +≡
  generic_eggsummary_cache < egganalysis > ec(&ed, ROTTEN_CSV, 50, 150, "pseudo");
```

12. The rest is identical to the plot from the egg data, other than labeling this plot as based on the pseudorandom data.

```
⟨ example-2.c 10 ⟩ +≡
  egganalysis * esr = ec.get_by_date("2001-04-22");
  esr→write_deviation_plot("example-2",      /* Name of plot file */
  "Earth␣Day:␣2001␣April␣22␣(Pseudorandom␣Data)",      /* Custom title for chart */
  systemtime::SecondsPerHour      /* Time interval for x axis labels */
  );
  ⟨ Epilogue for example program 48 ⟩;
```

13. Example 3: Multiple day time span plot.

In the previous examples, the cache proved handy when we wanted to switch between live and pseudorandom data, but otherwise it didn't do much for us. The cache comes into its own when you want to study time spans which cross day boundaries, or random access days from the database.

Here's a simple example. Let's study the behaviour of the eggs in the six hours surrounding the year 2001–2002 transition, measured by Universal Time (it may make more sense to align the data by local time at the eggs, but we'll leave that for later). We start by defining the database and creating the cache, precisely as in Example 1.

```
⟨ example-3.c 13 ⟩ ≡
  ⟨ Prologue for example program 47 ⟩;
  eggdatabases ed;
  ed.set_local_defaults( );
  generic_eggsummary_cache < egganalysis > ec(&ed, ROTTEN_CSV, 50, 150);
```

See also sections 14, 15, and 16.

14. The cache provides an *extract_time_range* method which retrieves samples for the specified time interval from the database, loading individual daily summary files as required, and fills in an egg data object of the type it was instantiated with with samples for the requested interval. Here we initialise an *egganalysis* object with samples for the six hours surrounding midnight UTC at the end of A.D. 2001. The start and end of the interval are given as "constants" of the **systemtime** class defined in the `timedate` program. This class comes in very handy when dealing with various ways of specifying dates and times, and is worth a quick scan if you need to do arithmetic on dates and times.

```
⟨ example-3.c 13 ⟩ +≡
  egganalysis ext;
  ec.extract_time_range(&ext, systemtime("2001-09-07␣00:00:00"),
      systemtime("2001-09-16␣0:00:00"));
```

15. With the data for the interval of interest extracted, we use the *describe* method to print a summary of it on standard output. This can be written to any output stream; by omitting the argument, we select the default, *cout*.

```
⟨ example-3.c 13 ⟩ +≡
  ext.describe( );
```

16. Finally, we produce the deviation plot, here adding arguments to offset the time axis in the plot by −3 hours so midnight is labeled as zero, and specifying a custom label for the X axis. We also explicitly specify the probability for the reference curve, but we really didn't need to, as it is the default value. Then, it's just a matter of tacking on the standard epilogue and we're out of here.

```
⟨ example-3.c 13 ⟩ +≡
  ext.write_deviation_plot("example-3",      /* Name of plot file */
  "",      /* Custom title for chart */
  systemtime::SecondsPerDay,      /* Time interval for x axis labels */
  8 * systemtime::SecondsPerDay,      /* Offset in seconds for x axis labels */
  "September␣Days␣(UTC)",      /* Custom title for x axis */
  0.05      /* Probability for reference curve */
  );
  ⟨ Epilogue for example program 48 ⟩;
```

17. Example 4: Per-egg statistics for a week's data.

So far, we've "sliced" data along the time axis: considering the behaviour of eggs at individual seconds of Universal Time. It's also interesting to monitor the behaviour of the individual eggs over periods of time. The expectation value is well defined, and deviations from that may indicate problems with the random event generators or the computers they're connected to. Also, studies of egg behaviour may point out idiosyncrasies of individual random event generators or of a given kind.

Here we study the behaviour of the eggs which submitted data in the first week of June, 2001. Most of the heavy lifting is done by methods of the *egganalysis* class, which you may consult in the `analysis` program. We start by extracting a data set for the week in question, as in the previous example.

```
⟨ example-4.c 17 ⟩ ≡
  ⟨ Prologue for example program 47 ⟩;
  eggdatabase ed;
  ed.set_local_defaults( );
  generic_eggsummary_cache < egganalysis > ec(&ed, ROTTEN_CSV, 50, 150);
  egganalysis ext;
  ec.extract_time_range(&ext, systemtime("2001-06-01␣00:00:00"),
      systemtime("2001-06-07␣23:59:59"));
```

See also sections 18, 19, and 20.

18. With data for the week in hand, we use the *compute_per_egg_statistics* method to prepare the statistics for the individual eggs.

```
⟨ example-4.c 17 ⟩ +≡
  ext.compute_per_egg_statistics( );
```

19. Finally, we write a primate-readable report of the statistics. If you require a computer-readable CSV file instead, the *write_per_egg_statistics_CSV* method of the *egganalysis* class will be happy to oblige. Before delving into the per-egg statistics, we print column headings.

```
⟨ example-4.c 17 ⟩ +≡
  cout ≪ "␣Egg␣Samples␣␣␣␣␣␣Mean␣␣␣␣␣␣␣␣Max␣␣␣Min␣␣␣␣␣␣Std.␣Dev." ≪ endl;
  cout.setf(ios::fixed, ios::floatfield);
  cout.precision(8);
  for (int i = 0; i < ext.eggs_reporting; i++) {
    if (ext.egg_num_sample[i] > 0) {
      cout ≪ setw(4) ≪ ext.egg_number[i] ≪ "␣␣" ≪ setw(6) ≪ ext.egg_num_sample[i] ≪
          "␣␣" ≪ setw(12) ≪ ext.egg_mean[i] ≪ "␣␣" ≪ setw(4) ≪ ((int)
          ext.egg_max_sample[i]) ≪ "␣␣" ≪ setw(4) ≪ ((int)
          ext.egg_min_sample[i]) ≪ "␣␣" ≪ setw(12) ≪ ext.egg_stdev[i] ≪ endl;
    }
  }
```

20. Okay, but what are the *expectation values* for an egg behaving entirely randomly? What an excellent opportunity to dig into the `statlib` program, which defines most of the common statistical metrics and provides facilities for evaluating their parameters! The random event generators at the egg sites are modeled by a binomal distribution with a probability $r = 0.5$ and $n = 200$ trials.

We create a *binomialDistribution* with the requisite parameters, and it serves up the mean (μ) and standard deviation (σ) for the distribution of our eggs' data. The expectation for the number of samples is simply the number of seconds in the interval.

⟨ `example-4.c` 17 ⟩ +≡

```
binomialDistribution pdist(200, 0.5);
cout ≪ endl ≪ setw(4) ≪ "Exp." ≪ "␣␣" ≪ setw(6) ≪ ext.seconds_of_data ≪ "␣␣" ≪ setw(12) ≪
    pdist.mean( ) ≪ "␣␣" ≪ setw(4) ≪ "␣" ≪ "␣␣" ≪ setw(4) ≪ "␣" ≪ "␣␣" ≪ setw(12) ≪
    pdist.stdev( ) ≪ endl;
⟨ Epilogue for example program 48 ⟩;
```

21. Example 5: The egg properties table.

Finally. . .an example that's meant to be *run* as much as read! The *egg_properties_database* class in `eggdata` provides access to a manually compiled table of characteristics of individual egg sites including location and the kind of random event generator used by the site. This database permits analyses to make geographical distinctions among eggs, for example aligning samples by local solar time at the site rather than Universal Time, or to study differences in the behaviour of different random event generators.

In this example we'll simply load the egg properties database from the CSV file in which it's maintained and print it out in tabular form using a method provided by the class. Refer to the resulting table in conjunction with the definition of the class in `eggdata` for details of the information it provides.

⟨ `example-5.c` 21 ⟩ ≡
```
⟨ Prologue for example program 47 ⟩;
egg_properties_database ed;
ed.loadFromCSV(EGGS_CSV);
```
See also section 22.

22. The egg properties table can be written to any **ostream**. Here we omit the argument to the *tabulate* method, which causes the table to be written to the default of *cout*. When you look at the output of this program, note that sites for which the altitude (metres above mean sea level) is unknown (the majority) show −9999 in the "Alt" column. North latitudes are positive, South negative; longitudes West of the Greenwich meridian are positive, East longitudes negative.

⟨ `example-5.c` 21 ⟩ +≡
```
ed.tabulate( );
⟨ Epilogue for example program 48 ⟩;
```

23. Example 6: New year transition in local time.

In Example 3 above we studied the six hour period surrounding the year 2001 − 2002 transition in the Gregorian calendar, looking at the behaviour of all the eggs at midnight Universal time. But we might ask what the data look like if we consider the local time at each egg—perhaps the local definition of midnight is more significant than Universal Time. (In this example, we ignore considerations about eggs located in areas where the Gregorian calendar is not used.)

What we'd like to do then, is time-shift the samples according to the prevailing time zone at each egg so that midnight in the resulting data set represents midnight *at each egg*. What we really need to accomplish this is a table giving the UNIX time zone declaration for each egg site, but we haven't compiled that information for the egg properties database (but we ought to!). So, unable to precisely correct for time zones, we opt for a rather tacky alternative of dividing the world into rigid 15° time zones and determining an egg's notional time zone by its latitude, ignoring politically-drawn zone boundaries and summer time. (The latter isn't as bad as you might think in this case, since the majority of the eggs are in the northern hemisphere as of this date, and aren't subject to summer time. Further, many southern hemisphere countries, which tend to be closer to the equator than those in the northern hemisphere, don't indulge in the silliness of summer time.)

We start out by declaring a cache and the egg properties database and loading the latter from the CSV file.

```
⟨ example-6.c 23 ⟩ ≡
  ⟨ Prologue for example program 47 ⟩;
  eggdatabases ed;
  ed.set_local_defaults( );
  generic_eggsummary_cache < egganalysis > ec(&ed, ROTTEN_CSV, 50, 150);
  egg_properties_database epd;
  epd.loadFromCSV(EGGS_CSV);
```

See also sections 24, 25, 26, 27, and 28.

24. Now we bring in three days' worth of data surrounding the New Year in Universal Time. Why three days? Because the process of shifting the eggs to align them to their time zone will shift the data in the table and we need to be sure we have sufficient data to cover the period of interest after shifting.

```
⟨ example-6.c 23 ⟩ +≡
  egganalysis ext;
  ec.extract_time_range(&ext, systemtime("2001-12-30␣00:00:00"),
      systemtime("2002-01-01␣23:59:59"));
```

25. Now we'll build a table of time zones for eggs contributing to the data set, looking up each in the egg properties database and estimating the time zone from its longitude. We build a **map** from egg number to the number of seconds of offset between its local clock and Universal Time.

```
⟨ example-6.c 23 ⟩ +≡
  int i;
  map⟨unsigned int, int⟩ offset;
  for (i = 0; i < ext.eggs_reporting; i++) {
    egg_properties * ep = epd.find(ext.egg_number[i]);
    assert(ep ≠ Λ);
    double zlon = ep→longitude − ((360/24)/2.0);
    double alon = fabs(zlon);
    int atz = (int)(alon/(360/24));
    int tz = (zlon < 0) ? (−atz) : atz;
    offset.insert(make_pair(ep→eggNumber, tz * systemtime::SecondsPerHour));
#if 0
    cout ≪ ep→eggNumber ≪ "␣␣" ≪ ep→longitude ≪ "␣␣" ≪ tz ≪ "␣␣" ≪
        offset.find(ep→eggNumber)→second ≪ endl;
#endif
  }
```

26. Now we walk through the eggs and time shift the data of each to align them to the time zones we've sorted them into. The *time_shift* method shifts samples for the given egg (specified by index rather than number, which is why we need to map the egg number to index with *egg_number_to_index*) by the specified number of seconds. Samples shifted out of the data set are replaced with a code which causes them to be ignored, but that's of no concern to us since in the next section we'll extract a sub-interval which is guaranteed to contain valid data for all eggs.

Note that we complement the sign of the time shift we computed above. Think about it: when it's midnight in London, it's 1 A.M. in Berne, so for a time zone expressed as −1 we need to shift the samples one hour *later* to align them with those already in Universal time. Using positive numbers for time zones West of Greenwich and negative for East is pretty much an artefact of provincialism on the part of the creators of UNIX, but there's nothing we can do about it now.

```
⟨ example-6.c 23 ⟩ +≡
  for (i = 0; i < ext.eggs_reporting; i++) {
    if (offset.find(ext.egg_number[i])→second ≠ 0) {
#if 0
      cout ≪ "Shifting␣egg␣" ≪ ext.egg_number[i] ≪ "␣by␣" ≪
          (−offset.find(ext.egg_number[i])→second) ≪ "␣seconds." ≪ endl;
#endif
      ext.time_shift(ext.egg_number_to_index(ext.egg_number[i]), −(offset.find(ext.egg_number[i])→second));
    }
  }
```

27. Having time-aligned the data for each egg to its local time zone in the data set spanning three days (in the process, potentially shifting out up to 12 hours on either end), we now use the *extract_time_range* method of the *egganalysis* class (inherited from its parent, *generic_eggsummary* in `eggdata` to prepare an extract of the six hours surrounding local midnight on December 31st, 2001. Note the difference between the *extract_time_range* methods of the egg summary object and the cache: the former extracts a subset of the data within a data set while the latter assembles a data set from the historical database.

```
⟨ example-6.c 23 ⟩ +≡
  egganalysis exa;
  ext.extract_time_range(&exa, systemtime("2001-12-31␣21:00:00"),
      systemtime("2002-01-01␣03:00:00"));
```

28. The deviation plot is made from the extracted data set as before.

```
⟨ example-6.c 23 ⟩ +≡
  exa.write_deviation_plot("example-6",      /* Name of plot file */
  "",      /* Custom title for chart */
  systemtime::SecondsPerHour,      /* Time interval for x axis labels */
  −3 * systemtime::SecondsPerHour,      /* Offset in seconds for x axis labels */
  "Hours␣from␣Midnight,␣Local␣Time",      /* Custom title for x axis */
  0.05      /* Probability for reference curve */
  );
  ⟨ Epilogue for example program 48 ⟩;
```

29. Example 7: SuperBowl 2001.

The SuperBowl of American Football is probably for most fervently celebrated secular sacrament in the United States. Let's look at how we might proceed to compare the behaviour of eggs located in or near the U.S. with other eggs around the world for SuperBowl XXXV in 2001. First of all, we bring in the data for the day in question. No, the date isn't wrong...recall that our databases are kept in Universal Time, and the SuperBowl, which is played in the afternoon in the U.S., begins after midnight in Universal Time.

```
⟨ example-7.c 29 ⟩ ≡
  ⟨ Prologue for example program 47 ⟩;
  eggdatabases ed;
  ed.set_local_defaults( );
  generic_eggsummary_cache < egganalysis > ec(&ed, ROTTEN_CSV, 50, 150);
  egganalysis * esr = ec.get_by_date("2001-01-29");
```

See also sections 30, 31, and 32.

30. Now we'll load the egg property database and walk though it, making a list of eggs in "North America" and elsewhere. We arbitrarily define "North America" as the portion of the globe between $23° - 40°$ North latitude and $60° - 135°$ West latitude. We build two vectors of egg numbers, *na_eggs* for those within the geographical bounds and *other_eggs* for those outside.

```
⟨ example-7.c 29 ⟩ +≡
  egg_properties_database epd;
  epd.loadFromCSV(EGGS_CSV);
  vector⟨unsigned int⟩ na_eggs, other_eggs;
  for (int i = 0; i < epd.size( ); i++) {
    egg_properties * ep = epd[i];
    if ((ep→latitude ≥ 25) ∧ (ep→latitude ≤ 50) ∧ (ep→longitude ≥ 60) ∧ (ep→longitude ≤ 135)) {
      na_eggs.push_back(ep→eggNumber);
    }
    else {
      other_eggs.push_back(ep→eggNumber);
    }
  }
```

31. Now we declare two separate *egganalysis* objects and use the *extract_eggs* method to extract data for eggs near North America in one and the balance of eggs in the other.

```
⟨ example-7.c 29 ⟩ +≡
  egganalysis exg, exo;
  esr→extract_eggs(&exg, na_eggs);
  esr→extract_eggs(&exo, other_eggs);
```

32. Now it's just a matter of making deviation plots for the complete set of eggs and the two geographical subsets. This time we add arguments to the *write_deviation_plot* calls to offset the labels on the time axis and the probability reference curve to 02:00, which I'll take as the time of the kickoff (actually, I have no idea when it actually was). Since most of the action in these charts was mostly to the downside, we use a negative number for the reference probability curve, which causes it to be plotted below the expectation value.

```
⟨ example-7.c 29 ⟩ +≡
  systemtime kickoff("2001-01-29␣02:00:00");
  esr→write_deviation_plot("example-7a",      /* Name of plot file */
  "SuperBowl␣2001-01-29:␣Worldwide␣Eggs",      /* Custom title for chart */
  systemtime::SecondsPerHour,      /* Time interval for x axis labels */
  systemtime::SecondsPerHour * -2,      /* Offset in seconds for x axis labels */
  "Hours␣from␣Kickoff",      /* Custom title for x axis */
  -0.05,      /* Probability for reference curve */
  kickoff      /* Time to begin reference curve plot */
  );
  exg.write_deviation_plot("example-7b",      /* Name of plot file */
  "SuperBowl␣2001-01-29:␣North␣American␣Eggs",      /* Custom title for chart */
  systemtime::SecondsPerHour,      /* Time interval for x axis labels */
  systemtime::SecondsPerHour * -2,      /* Offset in seconds for x axis labels */
  "Hours␣from␣Kickoff",      /* Custom title for x axis */
  -0.05,      /* Probability for reference curve */
  kickoff      /* Time to begin reference curve plot */
  );
  exo.write_deviation_plot("example-7c",      /* Name of plot file */
  "SuperBowl␣2001-01-29:␣Eggs␣Outside␣North␣America",      /* Custom title for chart */
  systemtime::SecondsPerHour,      /* Time interval for x axis labels */
  systemtime::SecondsPerHour * -2,      /* Offset in seconds for x axis labels */
  "Hours␣from␣Kickoff",      /* Custom title for x axis */
  -0.05,      /* Probability for reference curve */
  kickoff      /* Time to begin reference curve plot */
  );
  ⟨ Epilogue for example program 48 ⟩;
```

33. Example 8: Fourier transform analysis of a day's data.

```
⟨ example-8.c 33 ⟩ ≡
  ⟨ Prologue for example program 47 ⟩;
  eggdatabases ed;
  ed.set_local_defaults( );
  generic_eggsummary_cache < egganalysis > ec(&ed, ROTTEN_CSV, 50, 150);
  egganalysis ext;
  ec.extract_time_range(&ext, systemtime("2001-06-01␣00:00:00"),
      systemtime("2001-06-01␣23:59:59"));
```

See also sections 34, 35, 36, 37, and 38.

34. With data for the week in hand, we use the *compute_per_egg_statistics* method to prepare the statistics for the individual eggs.

```
⟨ example-8.c 33 ⟩ +≡
  ext.compute_all_egg_statistics( );
```

35. The Stouffer Z scores for each second of data are now copied into a *complex* array (with the imaginary part of each number set to zero) and Fourier transformed, then the power spectrum is computed from the transform. The *fourierTransform* class requires the arrays it operates on to be a power of two, so we create vectors of 2^{17} elements, the next power of two larger than the number of seconds per day.

```
⟨ example-8.c 33 ⟩ +≡
  const int ftsize = 131072;
  fourierTransform::complex vec[ftsize];
  double ps[ftsize];
  unsigned int nrows = ext.seconds_of_data / ext.seconds_per_row;
  memset(vec, 0, sizeof vec);
  for (int n = 0; n < nrows; n++) {
    vec[n].r = ext.stouffer_z[n];
  }
  fourierTransform ft(ftsize);
  ft.transform(vec);
  ft.power_spectrum(vec, ps);
```

36. At this point, the first **systemtime**::*SecondsPerDay* elements of the *ps* array starting with element 1 contain the power spectrum of the Stouffer Z scores in units of seconds. Now we aggregate the power into longer time units, summing power in each bin, to determine whether there is a peak at some binned time.

```
⟨ example-8.c 33 ⟩ +≡
  int aggregationBin = systemtime::SecondsPerHour;      /* Bin size in seconds */
  int nbins = systemtime::SecondsPerDay / aggregationBin;
  double *powerBin = new double[nbins];
  int psv = 1;      /* Power spectrum starts in cell 1: 0 is DC bias */
  for (int b = 0; b < nbins; b++) {
    powerBin[b] = 0;
    for (int k = 0; k < aggregationBin; k++) {
      powerBin[b] += ps[psv++];
    }
  }
```

37. Now it's just a matter of writing out the binned power spectrum and firing up `GNUPLOT` to create the chart. The resulting chart isn't particularly interesting, but this code may serve as a point of departure for other analyses of this nature.

```
⟨ example-8.c 33 ⟩ +≡
  string fileName = "example-8";
  ofstream gp((fileName + ".gp").c_str( )), dat((fileName + ".dat").c_str( ));
  for (int j = 0; j < nbins; j++) {
    dat ≪ j ≪ "␣" ≪ powerBin[j] ≪ endl;
  }
  delete powerBin;
  gp ≪ "set␣term␣pbm␣small␣color" ≪ endl;
  gp ≪ "plot␣\"" ≪ fileName ≪ ".dat\"␣using␣1:2␣with␣boxes" ≪ endl;
  string command("gnuplot␣");
  command += fileName + ".gp␣|␣ppmtogif␣>" + fileName + ".gif";
#undef DEV_PLOT_DEBUG
#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
```

38. Finally, we write a primate-readable report of the statistics. If you require a computer-readable CSV file instead, the *write_per_egg_statistics_CSV* method of the *egganalysis* class will be happy to oblige. Before delving into the per-egg statistics, we print column headings.

```
⟨ example-8.c 33 ⟩ +≡
  ⟨ Epilogue for example program 48 ⟩;
```

39. Example 9: Deviation analysis for a week's data.

This example illustrates *data mining*—searching the database for occurrences of a given kind of event or analysing the frequency or time distribution of an event of a certain kind. Here, we'll scan a week's worth of data from the eggs and tabulate the maximum deviation, both positive and negative, which occurred during each day of the week.

This task will also provide the opportunity to look "under the hood" of the *egganalysis* object we've been using in many of the previous examples. As before, we start by defining the database paths and create a cache.

```
⟨ example-9.c 39 ⟩ ≡
  ⟨ Prologue for example program 47 ⟩;
  eggdatabases ed;
  ed.set_local_defaults( );
  generic_eggsummary_cache < egganalysis > ec(&ed, ROTTEN_CSV, 50, 150);
```

See also sections 40, 41, 42, 43, and 44.

40. Next, we define the storage which will be used to hold the results we're tabulating. We'll analyse the first full week of 2002, starting with Sunday the 6th of January. We initialise a **systemtime** variable to the start date of the period of analysis and define the number of days we wish to process. Then, we emit the column headings for the deviation table.

```
⟨ example-9.c 39 ⟩ +≡
  systemtime thisdate("2002-01-06");      /* Current date */
  const int numDays = 7;      /* How many days to analyse */
  cout ≪ "␣␣␣Date␣␣␣␣␣Day␣␣␣␣Min.␣Dev.␣␣␣Max␣Dev.␣␣␣␣Delta␣Dev." ≪ endl;
```

41. Now we loop over the days in the interval. For each, we obtain an *egganalysis* object for the day in question from the cache and apply the *compute_all_egg_statistics* method to it in order to prepare the time series analysis across all the eggs. Among the statistics prepared are the Stouffer Z for all the eggs reporting for each second. Please see the definition of *compute_all_egg_statistics* in the `analysis` program for details of the values it computes.

```
⟨ example-9.c 39 ⟩ +≡
  for (int i = 0; i < numDays; i++) {
  egganalysis * esr = ec.get_by_date(thisdate);
  esr→compute_all_egg_statistics( );
```

42. With the time series statistics in hand, all that remains is to loop over the seconds in the day and compute the deviation by summing the squares of the Stouffer Z values (which obey χ^2 statistics), normed by the number of degrees of freedom, which is simply the incrementing number of seconds in the day. We use the *chiSquareDistribution* class in the `statlib` program for the latter calculation. As the deviation is computed over the day, we keep track of the extremal values.

```
⟨ example-9.c 39 ⟩ +≡
  double chisum = 0, mindev = 1 · 10^100, maxdev = −1 · 10^100;
  int nitems = esr→seconds_of_data / esr→seconds_per_row;
  double cxebase;
  for (int j = 0; j < nitems; j++) {
    double zsquare = esr→stouffer_z[j] * esr→stouffer_z[j];
    double xe = chiSquareDistribution::x_from_p_k(0.5, j + 1);
    chisum += zsquare;
    double deviation = chisum − xe;
    if (deviation < mindev) {
      mindev = deviation;
    }
    if (deviation > maxdev) {
      maxdev = deviation;
    }
  }
```

43. Now it's just a simple matter of writing out the minimum and maximum deviation and the difference between them in a nice tabular form.

```
⟨ example-9.c 39 ⟩ +≡
  static const char *dayOfWeek[7] = {"Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"};
  cout.setf(ios::fixed, ios::floatfield);
  cout.precision(3);
  cout << thisdate.dateToString( ) << "␣␣" << dayOfWeek[thisdate.weekday( )] << "␣␣" << setw(10) <<
      mindev << "␣␣" << setw(10) << maxdev << "␣␣" << setw(10) << (maxdev − mindev) << endl;
```

44. Tomorrow is another day! We use the *nextDay* method of the **systemtime** class to advance to the next day. Please see the definition of this class in the `timedate` program for other tools for manipulating dates and times. Since we're working on days in a linear fashion and assured we'll never revisit a day's data, we use the *purge* method of the cache to explicitly discard the data for the day we've requested. This reclaims the memory used by the day we've just completed and reduces the total memory required to analyse the week. In the case of a week-long analysis this doesn't make much difference but if you were, for example, scanning a year's data, failing to purge might cause your program to run out of memory and crash. The cache will eventually automatically purge data as required, but I haven't gotten around to that yet. As long as you know you're through with data in the cache, explicitly purging it can't hurt.

```
⟨ example-9.c 39 ⟩ +≡
  thisdate.nextDay( );      /* Advance to next day */
  ec.purge( );      /* Discard day just completed from cache */
  }
  ⟨ Epilogue for example program 48 ⟩;
```

45. Template for new examples.

⟨ `examples_test.c` 45 ⟩ ≡
⟨ Prologue for example program 47 ⟩;
cout ≪ `"Here␣is␣where␣you␣put␣the␣code␣for␣the␣example."` ≪ *endl*;
⟨ Epilogue for example program 48 ⟩;

46. Common Framework for Example Programs.

47. All the examples use a common prologue. It defines the *main* function which provides the entry point of the program, processes command line options, and wraps the experiment-specific code in a **try**–**catch** block which guarantees any unprocessed exceptions will result in an error message as opposed to a silent memory dump.

```
⟨ Prologue for example program 47 ⟩ ≡
  ⟨ Example program include files 51 ⟩;
  ⟨ Show how to call test program 50 ⟩;
  int main(int argc, char *argv[]){ extern char *optarg;      /* Imported from getopt */
      extern int optind; try { ⟨ Process command-line options 49 ⟩;
```

This code is used in sections 4, 10, 13, 17, 21, 23, 29, 33, 39, and 45.

48. A standard epilogue for example programs completes the exception handling block and terminates the program with normal status.

```
⟨ Epilogue for example program 48 ⟩ ≡
  } 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; }
```

This code is used in sections 9, 12, 16, 20, 22, 28, 32, 38, 44, and 45.

49. 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 49 ⟩ ≡
  int opt;
  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 47.

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

```
⟨ Show how to call test program 50 ⟩ ≡
  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 47.

51. We need definitions in the following header files to compile the example programs. Note that `analysis.h` takes the initiative to pull in the rest of the header files of the analysis suite as well as the usual ratty collection of C++ standard header files.

```
⟨ Example program include files 51 ⟩ ≡
#include "config.h"      /* Our configuration */
  ⟨ Preprocessor definitions ⟩
#include "analysis.h"      /* analysis.h pulls in everything else we need */
#include "fourier.h"      /* ...except fourier.h, which is rather more specialised */
#include <stdio.h>
#include <stdlib.h>
#ifdef HAVE_GETOPT
#ifdef HAVE_UNISTD_H
#include <unistd.h>
#endif
#else
#include "getopt.h"      /* No system getopt—use our own */
#endif
```

This code is used in section 47.

52. We generate an empty C program to keep `make` happy; it isn't used for anything.

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

⟨ Epilogue for example program 48 ⟩ Used in sections 9, 12, 16, 20, 22, 28, 32, 38, 44, and 45.
⟨ Example program include files 51 ⟩ Used in section 47.
⟨ Process command-line options 49 ⟩ Used in section 47.
⟨ Prologue for example program 47 ⟩ Used in sections 4, 10, 13, 17, 21, 23, 29, 33, 39, and 45.
⟨ Show how to call test program 50 ⟩ Used in section 47.
⟨ `example-1.c` 4, 5, 6, 7, 8, 9 ⟩
⟨ `example-2.c` 10, 11, 12 ⟩
⟨ `example-3.c` 13, 14, 15, 16 ⟩
⟨ `example-4.c` 17, 18, 19, 20 ⟩
⟨ `example-5.c` 21, 22 ⟩
⟨ `example-6.c` 23, 24, 25, 26, 27, 28 ⟩
⟨ `example-7.c` 29, 30, 31, 32 ⟩
⟨ `example-8.c` 33, 34, 35, 36, 37, 38 ⟩
⟨ `example-9.c` 39, 40, 41, 42, 43, 44 ⟩
⟨ `examples_test.c` 45 ⟩

EXAMPLES