October, 1998

**PHY307F/407F - Computational Physics**
**Background Material for Expt. 5 - Visualisation of Ozone Data**

David Harrison

*"The words or the language, as they are written or spoken, do not seem to play any role in my mechanism of thought. The psychical entities which seem to serve as elements in thought are certain signs and more or less clear images which can be 'voluntarily' reproduced and combined... this combinatory play seems to be the essential feature in productive thought before there is any connection with logical construction in words or other kinds of signs which can be communicated to others."*

*- Albert Einstein in a letter to Jacques Hadamard*

## INTRODUCTION

The topic of Scientific Visualisation is yet another example of the use of computer technology to do something that could in principle be done by hand given sufficient time. But in common with many other topics of this course, the technology makes it feel to us that we are doing something qualitatively different from what is possible without computers. The 'leading edge' of scientific visualisation is very close to the technology of virtual reality; this experiment is far from that edge.

Recall Anscombe's quartet of data that you examined in Experiment 1: four data sets, each consisting of 11 **{x,y}** pairs. The means of the **x**'s and **y**'s were virtually identical. Each data set fit to virtually the same straight line, with the same intercept, slope, error in the intercept, error in the slope, and sum of the squares. Only when we examined the graphs of the data did we realise what was going on. So this is one crucial example of scientific visualisation.

In this experiment we will be examining another case where visualisation is vital: when the quantity of data is so large that it is difficult to detect patterns in the data using any other technique. Our primary dataset will be for ozone levels in the upper atmosphere: each set contains 51,840 numbers. We will also be using a smaller dataset of groundwater levels for testing which has only 30 numbers.

Commercial visualisation software costs in the thousands of dollars. One of the largest markets for the software is in the medical industry, and this implies that huge attempts are made by the developers to provide powerful calculational tools to people with little understanding of the underlying mathematics. The same

software is also of great use to people who do understand the mathematics, however, and is used by some physicists in this Department. We will be using the rich but comparatively "low level" graphical environment of *Mathematica* for our investigation of visualisation; this is probably the right choice for a teaching application even if some vendor decided to donate commercial visualisation software to *UPSCALE*. Nonetheless as a dedicated single-purpose tool, commercial visualisation software often requires less computing resources than the general-purpose graphics of Mathematica.

The human visual system can simultaneously distinguish between approximately 15 different shades of gray. Thus in a visualisation of a dataset using a black and white graphic, only about 15 different values can be perceived, regardless of how many different shades are actually displayed in the graphic. The visual system can distinguish about an order of magnitude more colors than "gray scales"; thus a color graphic can provide much more information than a black and white one. This implies that visualisation systems require extensive computational and graphics capability. The design of "color maps" assigning particular numeric values to particular hues is an art using information about the type of data being used and what should be emphasised, factors in our human visual and cognitive systems, and more.

This document is organised as follows:

I. The Ozone Data: experimental details of how the data were taken.

II. Filling Missing Data: comparing methods of filling in missing data points.

III. Cubic Spline Interpolation: yet another method of filling in missing data.

IV. References.

V. Code Listing: a listing of the package you will be using in the experiment.

## I. THE OZONE DATA

The ozone data were taken by the *Total Ozone Measurment Spectrometer (TOMS)* aboard the Nimbus 7 satellite. The remainder of this section contains extracts from the document by Guimaraes and McPeters that accompanies the CD-ROMs containing the data.

*Nimbus 7 is in a south-north sun synchronous polar orbit such that it is always close to local noon/midnight beneath the satellite. Thus, ozone over the entire world is measured every 24 hours. TOMS measures ozone by measuring the ultraviolet sunlight scattered from the Earth's atmosphere. Total column ozone is inferred from the differential absorption of scattered sunlight in the ultraviolet. Ozone is calculated from the ratio of two wavelengths, 312 nm and 331 nm for*

*instance, where one wavelength is strongly absorbed by ozone while the other is only weakly absorbed. Ozone is measured in a 50 km square field of view of the instrument.*

*TOMS collects 35 measurements every 8 seconds as it scans from right to left, giving about 200,000 ozone measurements every day. These individual measurements have been averaged and converted to byte images which are scaled between 100 and 650 Dobson units (DU). A dobson unit is a measurement of the thickness of the ozone layer as an equivalent layer of pure ozone gas at NTP conditions (normal temperature and pressure at sea level), so that 300 Dobson units would equal 3 mm (about a tenth of an inch) thickness of pure ozone gas at NTP conditions.*

*Because TOMS measures ozone using scattered sunlight, it is not possible to measure ozone in the polar winter region where the sun never shines. Consequently, maps of the Antarctic ozone hole for August and September, for instance, will always have an area of missing data due to the polar night. [Some] days occasionally have large areas of missing data because of missing orbits.*

*A ... type of edge effect will be noticed in the polar plots consisting of apparent discontinuities in the ozone field. Since TOMS takes 24 hours to map the entire earth, at the day boundary the ozone measurements have been taken almost 24 hours apart. If the ozone is changing with time, this will produce the observed discontinuity.*

## II. FILLING MISSING DATA

In the early 1990's, Spyglass Inc. marketted a version of the now-defunct *Mosaic* world wide web browser, and also sold software for visualisation of data. In this section, we summarise some techniques on filling in missing data points as discussed in a handbook by Brand Fortner from Spyglass.[1]

The handbook uses a sample dataset of groundwater levels, extracted from a much larger matrix of data collected by Dr. Wolfram Herth. The data are:

1. Chapter 11.

| Groundwater Levels | | | | |
|---|---|---|---|---|
| -15.91 | 0 | -16.30 | 0 | -16.68 |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 |
| -14.15 | -14.37 | 0 | 0 | 0 |
| -13.36 | -13.57 | 0 | 0 | -13.96 |
| 0 | -12.56 | -12.75 | -12.89 | -12.98 |

In the above, zeroes indicate missing data points.

A very simple method of filling in the missing data is to use the value of the nearest neighbor. Fortner displays the following as the dataset after using the nearest neighbor method:

| After Nearest Neighbor Fill | | | | |
|---|---|---|---|---|
| -15.91 | -15.91 | -16.30 | -16.30 | -16.68 |
| -15.91 | -15.91 | -16.30 | -16.30 | -16.68 |
| -14.15 | -14.37 | -14.37 | -13.96 | -13.96 |
| -14.15 | -14.37 | -14.37 | -13.96 | -13.96 |
| -13.36 | -13.57 | -12.75 | -12.89 | -13.96 |
| -13.36 | -12.56 | -12.75 | -12.89 | -12.98 |

Note that in the above, data points which were missing and did not have a non-missing nearest neighbor have been filled in. You should be able to understand how this happened.

Next, Fortner uses a technique he calls **Linear Interpolation** where the missing data values are interpolated from the closest data values in the same column, the same row, or both. For example:

| After Linear Interpolation along Rows | | | | |
|---|---|---|---|---|
| -15.91 | -16.10 | -16.30 | -16.49 | -16.68 |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 |
| -14.15 | -14.37 | -14.59 | -14.80 | -15.02 |
| -13.36 | -13.57 | -13.70 | -13.83 | -13.96 |
| -12.37 | -12.56 | -12.75 | -12.89 | -12.98 |

Note that because there were no data points in rows two or three, these rows

remain empty.

A way of filling in rows two and three is to interpolate first along columns, and then along rows:

| After Interpolating Columns and then Rows | | | | |
|---|---|---|---|---|
| -15.91 | -16.44 | -16.30 | -14.69 | -16.68 |
| -15.33 | -15.75 | -15.66 | -14.41 | -16.07 |
| -14.74 | -15.06 | -15.02 | -14.13 | -15.45 |
| -14.15 | -14.37 | -14.38 | -13.85 | -14.83 |
| -13.36 | -13.57 | -13.58 | -13.36 | -13.96 |
| -12.47 | -12.56 | -12.75 | -12.89 | -12.98 |

Another method Fortner calls **Smooth Fill**. Here, each missing data point is replaced by an average of all of its non-missing immediate neighbors:

| After a Single Smooth Fill | | | | |
|---|---|---|---|---|
| -15.91 | -16.11 | -16.30 | -16.49 | -16.68 |
| -15.91 | -16.11 | -16.30 | -16.49 | -16.68 |
| -14.26 | -14.26 | -14.37 | 0 | 0 |
| -13.86 | -13.86 | -13.97 | -13.96 | -13.96 |
| -13.60 | -13.46 | -13.22 | -13.14 | -13.28 |
| -13.16 | -13.06 | -12.94 | -13.14 | -13.28 |

Note that there are still two missing values in the above, because in the original data they had no non-missing neighbors. Also note that the routine has modifed known data values.

A second Smooth Fill pass will eliminate the two missing values in the above data. Since the method is replacing known values with averages, it will also smooth out the variations in the dataset. In fact, if one does enough passes using this method all the data points will end up having the same value!

An extension of the idea behind Smooth Fills comes from the realization that the closer a data point is to the missing value to be filled, the more weight we should give it in the average. Thus, in a **Weighted Fill** we include data points further away from the missing value than just the nearest neighbor, but give them a reduced weight in the calculation.

Various functions can be used in calculating the weighting factors, such as a Gaussian, or $1/r$, or $1/r^2$, etc.

Note that all of the above functions only approach zero asymptotically. So that the computer does not spend an inordinate amount of time using data points very far away from the point to be filled, commonly a *cutoff radius* is defined, and points further away from the missing value than that radius are not used in the weighted fill. Sometimes the selection of a value of the cutoff radius can be made by understanding the data itself. Otherwise, one may reason as follows: If there are no missing values, then the number $n$ of data points inside a radius $r$ is:

$$n = \pi r^2$$

If a fraction $f$ of the data is not missing, then the number of non-missing data points inside the radius $r$ is:

$$n = f\pi r^2$$

Thus, a reasonable choice for the cutoff radius is:

$$r_{cutoff} = \left( \frac{n}{f\pi} \right)^{0.5}$$

**III. CUBIC SPLINE INTERPOLATION**

You may recall that in the background material for the experiment on *Fitting Techniques* we briefly mentioned but did not investigate fits in which a model is not available or appropriate: such fits are called *nonparametric*. In this section we discuss a type of nonparametric fit that can be used for filling in missing data.

Of course, if a model for the data is available, one may simply fit the data, say by rows, to the model and then use the fit results to interpolate and possibly extrapolate to fill in missing data points.

Consider using a French curve to draw a smooth curve through a graph of some data. You break the data into a series of segments and try to find a part of the French curve to smoothly connect the points in each segment. At the 'knots' between the segments you adjust so the two curves meet and the slopes are continuous. A *cubic spline* is essentially a computerised French curve, where the data in each segment is fit to a cubic polynomial. This is clearly a nonparametric fit: there is probably no theoretical reason why a cubic polynomial should fit the data, but it usually provides a sufficiently smooth way of "connecting the dots." *Mathematica*'s **`Interpolation`** procedure does cubic spline interpolations where each data point is one of the knots, and the fits are constrained to go through each data point exactly.

In the experiment you will be investigating a routine that uses the **Interpolation** procedure to fill in missing data points.

Another type of cubic spline interpolation allows the user to choose the position of the knots. In this case, the interpolation tends to smooth out noise in the data. Such an interpolation is used in, for example, the Mass Spectrometer experiment in the III & IV Year Physics Laboratory.

A further issue with many filling methods, including this one, is what is the largest block of missing data which should be filled. The answer to this question is subtle, and usually depends on the largest size of a block of non-missing data that does not show significant structure inside it.

## IV. REFERENCES


- Brand Fortner, **The Data Handbook** (Spyglass Inc., 1992).
- P. Guimaraes and R. McPeters, Eds., "TOMS Ozone Image Data 1978-1991", (NASA Goddard Space Flight Center).
  This is the document that accompanies the CD-ROMS containing the ozone data.


## V. CODE LISTING

```
(*
 * Support package for Expt 5: Visualisation of Ozone Data.
 * All routines written by David Harrison.
 *
 * Contents visible from outside the package:
 *  LoadDu[] - load a day's data.
 *  TableDu[] - form a table of parts of multiple day data.
 *    FillData[] - fill in missing data.
 *    DoNearestNeighbor[] - fill using nearest neighbor.  Default
 *          method used by FillData[]
 *    SurroundIt[] - actually an internal routine used by DoNearestNeighbor[]
 *          but made accessible from outside the package for this experiment.
 *    GroundWaterData
 *
 * Internal routines:
 *    columnInterpolate[] - optionally used by FillData[]
 *    rowInterpolate[] - optionally used by FillData[]
 *
 * @(#)Expt5.m    1.10 U of T Physics 01/12/96
 *
```

```
 * LoadDu, TableDu are Copyright (c) 1993, 1994, 1995 by David Harrison.
 *
 * FillData, DoNearestNeighbor, SurroundIt, columnInterpolate,
 * rowInterpolate[] and smoothFill are Copyright (c) 1995
 * by Wolfram Research Inc.
 *)

(* Graphics‘ParametricPlot3D‘ for SphericalPlot3D[] *)

BeginPackage["PHY307F‘Expt5‘", "Graphics‘ParametricPlot3D‘" ]

FillData::usage =
      "FillData[data] fills in missing values in the matrix of values ‘data‘.
      The default Method is NearestNeighbor, and the routine determines which
      values are to be filled by the definition of MissingIs, which by default
      is 0."

DoNearestNeighbor::usage =
      "DoNearestNeighbor[data,missing] fills in missing values in the matrix
      of values ‘data’ by replacing elements whose value is equal to ‘missing’
      with the value of its nearest neighbor."

SurroundIt::usage =
      "SurroundIt[data, missing, n] takes a matrix ‘data’ and inserts and
      appends n rows and columns of value equal to ‘missing’.  Thus, the
      original matrix contains ‘rows’ rows and ‘cols’ columns, it returns
      a matrix of ‘rows + 2*n’ rows and ‘cols + 2*n’ columns."

Options[FillData] = {
      MaxIterations -> 1,
      Method -> NearestNeighbor,
      MissingIs -> 0,
      StructureSize -> Infinity
}

NearestNeighbor::usage =
      "NearestNeighbor is the default Method for FillData.  This Method fills
      in missing values with duplicates of an adjancent non-missing neighbor.
      Other Methods are ColumnInterpolation and RowInterpolation."

ColumnInterpolation::usage =
```

```
      "ColumnInterpolation is an optional Method for FillData.  This Method
      fills in missing values with interpolations from non-missing values in
      the same column."

RowInterpolation::usage =
      "RowInterpolation is an optional Method for FillData.  This Method
      fills in missing values with interpolations from non-missing values in
      the same row."

MaxIterations::usage =
      "MaxIterations is an option to various routines that use
      iterative techniques, and specifies the maximum number
      of iterations to perform before quitting."

MissingIs::usage =
      "MissingIs defines contents of cells in a matrix of data that
      FillData assumes are missing data."

StructureSize::usage =
      "StructureSize is an Option for FillData when the Method is
      RowInterpolation or ColumnInterpolation that specifies the size of
      structures in the data.  If a block of missing values is greater than
      the value of StructureSize, that block is not interpolated."

FillData::badopt = "Option ‘1‘ is not known."
FillData::nomethod =
      "Method ‘1‘ is not valid for FillData."

GroundwaterData::usage =
      "GroundwaterData is made up data on groundwater from Brand
      Fortner, \"The Data Handbook\" (Spyglass Inc., 1992), pg. 155.
      Missing data are represented by a value of zero."

LoadDu::usage =
      "LoadDu[day] loads the TOMS data for a day during December 1988.
      The argument day is a number between 01 and 31 and determines the
      day of the month that is loaded."

TableDu::usage =
      "TableDu[ {day1, day2, daycounter}, {latbin1,latbin2,
      latcounter}, {longbin1, longbin2,longcounter}] uses LoadDu[]
```

```
      to load the data for day1 through day2 in increments of daycounter.
      It extracts the latitude bins latbin1 through latbin2 in increments
      of latcounter, and extracts longitude bins longbin1 through longbin2
      in increments of longcounter. If invoked as:\n
      \tthetable = TableDu[ {10,15,1}, {10,20,1}, {30,40,1}];\n
      then thetable[[a]] contains the du data for the a'th day in the
      table, thetable[[a, b]] contains the longitude data for
      the a'th day and the b'th latitude bin of the table, and
      thetable[[a,b,c]] contains the single du value for the a'th day,
      b'th latitude bin and c'th longitude bin of the table.
      CAUTION:  the routine removes all variables that begin with the
      letters 'dec' from your session."

LoadDu::noday = "day must be an integer between 1 and 31."

Begin["`Private`"]

FillData[data_?MatrixQ, opts___?OptionQ] := Module[
      {
            i,                              (* dummy variable *)
            iter, workdata,

            optMaxIterations,
            optMethod,
            optMissingIs,
            optStructureSize
      },

      i = Complement[ First /@ {opts},
                        First /@ Options[FillData] ];
      If[Length[i] != 0,
            Message[FillData::badopt, #]& /@ i;
            Return[$Failed]
      ];


      optMaxIterations = MaxIterations /. {opts} /. Options[FillData];
      optMethod = Method /. {opts} /. Options[FillData];
      optMissingIs = MissingIs /. {opts} /. Options[FillData];
      optStructureSize = StructureSize /. {opts} /. Options[FillData];
```

```
      workdata = data;

      For[iter = 1, iter <= optMaxIterations, iter++,
            If[Count[Flatten[workdata], optMissingIs] == 0,
                  Break[]
            ];
            Switch[ optMethod,

                  ColumnInterpolation,
                        workdata = columnInterpolate[workdata, optMissingIs,
                              optStructureSize ],

                  NearestNeighbor,
                        workdata = DoNearestNeighbor[workdata, optMissingIs ],

                  RowInterpolation,
                        workdata = rowInterpolate[workdata, optMissingIs,
                              optStructureSize ],

                  _,    Message[FillData::nomethod, optMethod];
                        Return[$Failed];

            ];
      ];

      workdata

]

columnInterpolate[data_, missing_, size_ ] := Transpose[
      rowInterpolate[ Transpose[data], missing, size ]]

(*
 * Since this is intended to be an "internal" routine called from
 * FillData[], we don't do argument checking.
 *)
DoNearestNeighbor[data_, missing_] := Module[
      {
            i,j,k,      (* dummy variables *)
            posns,
            row, col,
```

```
      numrows, numcols,
      parts, added, workdata
},

numrows = Length[data];
numcols = Length[First[data]];

(* We will put the filled in terms in added.  This keeps a filled
 * point from propagating down through the rows as it would if we
 * simply changed values in workdata.
 *)
added = Table[0, {numrows + 2}, {numcols + 2}];

workdata = SurroundIt[data, missing, 1];

For[row = 1, row <= numrows, row++,

      (* Find positions of missing data.  Since we have surrounded the
       * data with missing values, posns will be {1,numcols + 2} if
       * there is no missing data in the row.
       *)
      posns = Flatten[Position[workdata[[row + 1]], missing]];
      If[ Length[posns] == 2,
            Continue[]
      ];
      posns = Take[ posns, {2,-2}];
      (* Now we form the 3x3 matrices with the missing values in
       * the center and flatten them.
       *)
      parts = Take[workdata, {row, row + 2} ];
      parts = Transpose[parts];
      parts = Take[ parts, {# - 1, # + 1}]& /@ posns;
      parts = Flatten /@ parts;

      (* Drop the missing values *)
      parts = DeleteCases[#, i_ /; i == missing]& /@ parts;
      If[ Union[parts] === {{}},
            Continue[];
      ];

      (* Remove any empty parts from both parts and posns *)
```

```
            {parts,posns} = Transpose[DeleteCases[Transpose[{parts,posns}],
                  {i_, j_} /; Length[i] == 0] ];

            (* The first value left in parts will replace the missing one. *)
            i = First /@ parts;
            j = Length[i];
            For[k = 1, k <= j, k++,
                  added = ReplacePart[added, i[[k]],
                        {row + 1, posns[[k]] }  ];
            ];

      ];

      If[missing != 0,
            posns = Position[workdata, missing];
            workdata = ReplacePart[workdata, 0, posns];
      ];

      workdata = workdata + added;
      workdata = Take[workdata, {2,-2}];
      workdata = Transpose[Take[Transpose[workdata],{2,-2}]];

      workdata//N

]

rowInterpolate[data_, missing_, size_ ] := Module[
      {
            i,j,k,

            intOrder,
            first, last,
            numtofill,
            posnsMissing, posnsFilled,
            row, numrows, numcols,
            numToInterpolate, skipFills,
            tofill,
            workdata, workrow
      },

      numrows = Length[data];
```

```
numcols = Length[First[data]];

workdata = data;

For[row = 1, row <= numrows, row++,
      workrow = workdata[[row]];

      posnsMissing = Flatten[Position[workrow, missing]];
      If[Length[posnsMissing] == 0 || Length[posnsMissing] == numcols,
            Continue[]
      ];

      posnsFilled = Complement[ Table[i, {i,numcols}], posnsMissing];
      tofill = Partition[posnsFilled, 2, 1];

      (* We only fill when there is a missing value *)
      tofill = DeleteCases[tofill, {i_,j_} /; j - i == 1 ];
      If[Length[tofill] == 0,
            Continue[]
      ];

      (* Now we assemble all the ranges in tofill which have
       * too many missing values.  We will delete these from
       * the list of posnsMissing, so they will not be filled
       * in by the Interpolation.  We assume that interpolating
       * across these ranges is still reasonable since Interpolation
       * is forced to go through every data point.
       *)
      skipFills = Cases[ tofill, {i_, j_} /; j - i > size ];
      j = Length[skipFills];
      For[ i = 1, i <= j, i++,
            posnsMissing = DeleteCases[posnsMissing,
                  k_ /; k > First[ skipFills[[i]] ] &&
                              k < Last[ skipFills[[i]] ] ];
      ];

      (* Simplest case - interpolate everything and then go on to
       * the next row.
       *)
      first = First[posnsFilled];
      last = Last[posnsFilled];
```

```
            i = Part[workrow, posnsFilled];
            i = Transpose[ {posnsFilled, i} ];

            (* If not enough data points for default interpolation,
             * we will have to reduce the order.
             *)
            intOrder = 3;
            If[Length[i] <= 3,
                  intOrder = Length[i] - 1;
            ];
            j = Interpolation[i, InterpolationOrder -> intOrder ];

            (*
             * Drop missing positions outside the range of the
             * interpolation.
                  *)
            posnsMissing = DeleteCases[ posnsMissing,
                  i_ /; i < first || i > last ];
            k = Length[posnsMissing];

            (* Construct the interpolated values and put in workrow *)
            For[i = 1, i <= k, i++,
                  workrow[[ posnsMissing[[i]] ]] = j[ posnsMissing[[i]] ];
            ];
            workdata[[row]] = workrow;

      ];

      workdata//N
]


(*
 * Surround the data by n levels of missing values.  The reason for
 * setting it up this way and including an input argument ‘n’ is that
 * I anticipate adding a WeightedAverage Method to FillData in a future
 * release.
 *
 * Since this is intended to be an "internal" routine called from
 * DoNearestNeighbor[], we don’t do argument checking.
 *)
```

```
SurroundIt[data_, missing_, n_] := Module[
      {
            i,j,
            numrows, numcols,
            workdata = data
      },

      numrows = Length[data];
      numcols = Length[First[data]];

      j = Table[missing, {numcols}];
      For[i = 1, i <= n, i++,
            workdata = Insert[workdata,j,{ {1}, {-1} }];
      ];
      j = Table[missing, {numrows + 2*n}];
      workdata = Transpose[workdata];
      For[i = 1, i <= n, i++,
            workdata = Insert[workdata, j, {{1},{-1}}];
      ];
      workdata = Transpose[workdata];
      workdata


]

GroundwaterData =
      {     {-15.91, 0, -16.30, 0, -16.68},
            {0, 0, 0, 0, 0},
            {0, 0, 0, 0, 0},
            {-14.15, -14.37, 0, 0, 0},
            {-13.36, -13.57, 0, 0, -13.96},
            {0, -12.56, -12.75, -12.89, -12.98}};

LoadDu[day_?IntegerQ] /; (day >= 1 && day <= 31) := Switch[ day,
            01,   Print["Loading dec01"]; << PHY307F/TOMS/01dec88;,
            02,   Print["Loading dec02"]; << PHY307F/TOMS/02dec88;,
            03,   Print["Loading dec03"]; << PHY307F/TOMS/03dec88;,
            04,   Print["Loading dec04"]; << PHY307F/TOMS/04dec88;,
            05,   Print["Loading dec05"]; << PHY307F/TOMS/05dec88;,
            06,   Print["Loading dec06"]; << PHY307F/TOMS/06dec88;,
            07,   Print["Loading dec07"]; << PHY307F/TOMS/07dec88;,
```

```
            08,   Print["Loading dec08"]; << PHY307F/TOMS/08dec88;,
            09,   Print["Loading dec09"]; << PHY307F/TOMS/09dec88;,
            10,   Print["Loading dec10"]; << PHY307F/TOMS/10dec88;,
            11,   Print["Loading dec11"]; << PHY307F/TOMS/11dec88;,
            12,   Print["Loading dec12"]; << PHY307F/TOMS/12dec88;,
            13,   Print["Loading dec13"]; << PHY307F/TOMS/13dec88;,
            14,   Print["Loading dec14"]; << PHY307F/TOMS/14dec88;,
            15,   Print["Loading dec15"]; << PHY307F/TOMS/15dec88;,
            16,   Print["Loading dec16"]; << PHY307F/TOMS/16dec88;,
            17,   Print["Loading dec17"]; << PHY307F/TOMS/17dec88;,
            18,   Print["Loading dec18"]; << PHY307F/TOMS/18dec88;,
            19,   Print["Loading dec19"]; << PHY307F/TOMS/19dec88;,
            20,   Print["Loading dec20"]; << PHY307F/TOMS/20dec88;,
            21,   Print["Loading dec21"]; << PHY307F/TOMS/21dec88;,
            22,   Print["Loading dec22"]; << PHY307F/TOMS/22dec88;,
            23,   Print["Loading dec23"]; << PHY307F/TOMS/23dec88;,
            24,   Print["Loading dec24"]; << PHY307F/TOMS/24dec88;,
            25,   Print["Loading dec25"]; << PHY307F/TOMS/25dec88;,
            26,   Print["Loading dec26"]; << PHY307F/TOMS/26dec88;,
            27,   Print["Loading dec27"]; << PHY307F/TOMS/27dec88;,
            28,   Print["Loading dec28"]; << PHY307F/TOMS/28dec88;,
            29,   Print["Loading dec29"]; << PHY307F/TOMS/29dec88;,
            30,   Print["Loading dec30"]; << PHY307F/TOMS/30dec88;,
            31,   Print["Loading dec31"]; << PHY307F/TOMS/31dec88;
]

LoadDu[___] := Crikey /; Message[LoadDu::noday]

TableDu[day_List, lat_List, long_List] := Module[
      {
            i,          (* dummy *)
            duname,           (* name of the current loaded du data *)
            latc,       (* counter for latitudes *)
            longc,            (* counter for longitudes *)
            daytmp,           (* construct the result here *)
            lattmp,           (* place to hold the current latitude data *)
            longtmp,    (* place to hold the current longitude data *)
            s           (* for holding input strings *)
      },

      (*
```

```
 * Check that the input arguments make sense, warn if
 * defined names will be blown away.
 *)
If[ Length[day] != 3 || day[[1]] > day[[2]],
      Print["The day argument is not well formed."];
      Return[$Failed]
];
If[ Length[lat] != 3 || lat[[1]] > lat[[2]],
      Print["The latitude argument is not well formed."];
      Return[$Failed]
];
If[ Length[long] != 3 || long[[1]] > long[[2]],
      Print["The longitude argument is not well formed."];
      Return[$Failed]
];
If[ Length[Names[ "dec*"]] >= 1,
      Print["CAUTION: this procedure will remove the following"];
      Print["names from your session: ", Names[ "dec*"]];
];

daytmp = {};
For[ i = day[[1]], i <= day[[2]], i += day[[3]],

      (*
       * Construct the name of the the du data to be loaded.
       * If the day is less than 10, must explicitly put the
       * zero '0' in the string.
       *)
      If[ i > 9,
            duname = ToExpression[
                  StringJoin[ Characters["dec"], ToString[i] ]],
      (* else *)
            duname = ToExpression[
                  StringJoin[ Characters["dec0"], ToString[i] ]]
      ];

      (* Load data for a day, checking result *)
      Check[ LoadDu[i],
            ClearAll[ "dec* "];
            Return[$Failed]
      ];
```

```
            (* sanity check *)
            If[ MatrixQ[ duname ] == False ||
                  Length[ duname] < lat[[1]] ||
                  Length[ duname] < lat[[2]] ||
                  Length[ duname[[1]] ] < long[[1]] ||
                  Length[ duname[[1]] ] < long[[2]],
                        Print["Cannot extract parts."];
                        ClearAll[ "dec* "];
                        Return[$Failed]
            ];

            (*
             * Cycle through the latitudes and longitudes, storing
             * the desired data in 'longtmp'.
             *)
            longtmp = {};
            For[ latc = lat[[1]], latc <= lat[[2]], latc += lat[[3]],
                  lattmp = {};
                  For[ longc = long[[1]],
                       longc <= long[[2]], longc += long[[3]],
                        lattmp = AppendTo[lattmp, duname[[latc, longc]] ];
                  ];
                  longtmp = AppendTo[longtmp, lattmp];
            ];

            ClearAll[ "dec*" ]; (* clear the name *)

            daytmp = AppendTo[ daytmp, longtmp ];
      ];

      (* Finished - return the result *)
      daytmp
]

End[]
EndPackage[]
```

---