TECHNICAL NOTE

No. TN-00/2

CIRCAD: AUTOMATED ANALYSIS OF CIRCADIAN CORE TEMPERATURE DATA

By

T.J. Doherty, M.D. Coyne, C.M. Kesick, and L.A. Stephenson

January 2000

U.S. Army Research Institute of Environmental Medicine
Natick, MA 01760-5007

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	JAN 2000	Technical Note for period 1 Jul 99 to 27 Jul 99

4. TITLE AND SUBTITLE

CIRCAD: Automated Analysis of Circadian Core Temperature Data

5. FUNDING NUMBERS

6. AUTHOR(S)

Tammy J. Doherty, Mary D. Coyne, Christina M. Kesick and Lou A. Stephenson

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

U.S. Army Research Institute of Environmental Medicine
Natick, MA 01760-5007

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

U.S. Army Medical Research and Materiel Command
Fort Detrick, MD 21702-5012

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for Public Release; Distribution is Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

No accepted methodologies exist to study day-to-day changes in the characteristics of the circadian rhythm. Testing different methodologies using data processing and curve-fitting functions within Microsoft Excel and Jandel Sigmaplot is prohibitively expensive in terms of both time and personnel. Use of the CIRCAD program, described in this report, dramatically reduces the amount of time required for circadian data analyses and provides the capability to quickly implement and test new analytical methods. This Technical Note includes a brief description of the software, complete program listings in the appendices, and sample input and output including both worksheet samplers and graphs.

14. SUBJECT TERMS

Circadian Rhythmn, Body Core Temperature, Computer Software, Analytical Methods, Cosinor, Curve Fitting

15. NUMBER OF PAGES
49

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFED	

DISCLAIMER

The views, opinions, and/or findings contained in this report are those of the author and should not be construed as an official Department of the Army position, policy or decision unless so designated by other official documentation.

Citations of commercial organizations and trade names in this report do not constitute an official Department of the Army endorsement or approval of the products or services of these organizations.

CONTENTS

FIGURES

EXECUTIVE SUMMARY

No accepted methodologies exist to study day to day changes in the characteristics of the circadian rhythm. Testing different methodologies using data processing and curve-fitting functions within Microsoft Excel and Jandel Sigmaplot is prohibitively expensive in terms of both time and personnel. Use of the CIRCAD program, described in this report, dramatically reduces the amount of time required for circadian data analyses and provides the capability to quickly implement and test new analytical methods. This Technical Note includes a brief description of the software, complete program listings in the appendices, and sample input and output including both Excel worksheet samplers and graphs.

INTRODUCTION

PURPOSE

The purpose of CIRCAD is to automate the process of analyzing circadian core temperature data, while providing the capability to readily implement and test new analytical methods. The analytical methods demonstrated in the current version of CIRCAD are variations of the cosinor-fit method and are used to explore day to day changes in circadian rhythm characteristics.

PROBLEM STATEMENT

Various methodologies including spectral analysis (e.g., Fast Fourier Transform), autoregressive modeling (e.g., ARMA and MESA), and curve-fitting approaches such as cosinor and Lomb are available for obtaining the circadian period using core temperature data windows of several days in width. These methods all assume "stationary" data. For circadian data, stationary implies that the period, phase, amplitude, mesor, zenith, nadir, and acrophase (Figure 1) are constant from one day to the next. In reality, circadian data are clearly not stationary (Figure 2). In fact, it is the goal of this project to develop software to aid in the study of this day to day variability.

Prior to the development of the CIRCAD software, implementation and testing of analytical methods to obtain circadian parameters on a day to day basis were severely impeded by the time and resources required. All data manipulation and curve fitting were performed using Microsoft Excel and Jandel Sigmaplot. Core temperature and actigraphy data, including time and date marks, were copied to an Excel spreadsheet. After copying the data to Excel, technicians plotted the data to look for outliers and valid data start and stop times. Outliers and any data collected outside the data start and stop times were removed manually. Core temperature and actigraphy data were then meshed to a single date/time reference. Sleep/wake data were entered manually, using the same time/date reference as the core temperature and actigraphy data. Values of 0 were entered for all time points in which the subjects were awake, and values of 38.2 were entered for all time points in which the subjects were asleep. Data were then copied to Sigmaplot. Cosine curves, with a set period length of 24 hours, were fit to each 24-hour day (midnight to midnight) within the data set. Plots were then generated showing the predicted versus the raw data. Scatter plots of mesor, period, amplitude and acrophase versus day, relative to an event marker, were also generated. Manipulations and curve-fitting, using Excel and SigmaPlot, were tediously slow due to the large number of data records (frequently exceeding 10,000). Even after technicians became familiar with the routine, analysis of each data set averaged several man-days. With several subjects and several data sets per subject, as well as a desire to investigate other analytical approaches besides cosinor with constant period length, it was clear that a more expedient process for preparing and analyzing data was required.

OVERVIEW

The CIRCAD software first and foremost eliminates and speeds up much of the tedium associated with circadian data analysis. Core temperature and actigraphy data must be copied to the Excel file as before. The technician must enter into the Excel file the time/date of all sleep/wake events, the date of the reference event marker, and the start and stop times for valid data. The technician then loads Matlab and runs the CIRCAD program. It takes several minutes for CIRCAD to perform the data manipulation tasks of removing outliers and meshing the core temperature, actigraphy, and sleep/wake data into a single date/time reference that were previously assigned to the technician.

Once this data manipulation has been accomplished, CIRCAD analyzes the data using 48 different variations on the cosinor method. Other methods will be added in future versions of the software. The three types of variations of the cosinor method include (1) cosine fits to filtered rather than raw core temperature data, (2) alternate methodologies for obtaining circadian period, and (3) alterations in the width and starting point (in time) of the data window used in performing the cosine curve fitting. All results are written to the Excel file in a way that facilitates graphing.

METHODS

EXPERIMENTAL DATA AND PREPARATION OF THE EXCEL DATA FILE

Three types of data were collected. Core temperature was obtained using a core temperature telemetry pill (Human Technologies Inc., St. Petersburg, FL) and a BCTM2 or BCTM3 receiver/recorder (Personal Electronic Devices Inc., Wellesley, MA). These data were recorded approximately once per minute. Data were also recorded using an actigraphy/skin temperature monitor (Mini-Mitter Company, Sunriver, OR). Actigraphy data consist of the counts of body movements in a 1-minute period. In a few cases, skin temperature data at 2-3 locations, were also recorded. Finally, sleep/wake data were recorded manually, on a log sheet, kept by the test subjects. Each type of data was associated with its own time and date marks. Data were typically recorded for a 2-10 day period from a single subject for each data set.

Core temperature and actigraphy data were first downloaded from the recording device to an Excel worksheet on a personal computer. Separate date/time references were maintained for each recording device (i.e., BCTM and MiniMitter). Sleep/wake data were manually entered into the worksheet. Sleep or wake onset date and time were entered as separate columns; a third0 column was used to contain the sleep/wake code ("S" for sleep onset, or "W" for awakening). The technician followed strict rules in naming variables within the worksheet and ensured that there were no blank values in any of the time or date columns. A file naming system was developed that includes codes to identify the subject, the data set, and the Julian date and year at the start of data collection. The worksheet containing the raw BCTM and MiniMitter data was given the same name. This was done because the functions for dynamic data exchange (DDE) in Matlab make connections to worksheets of a specified name; these DDE

functions ignore file name specifications. Thus, if the same worksheet name were to be used in 2 different files, Matlab would read data from the worksheet of whichever file was active at the time. The only way to ensure that the correct worksheet is read by CIRCAD is to use worksheet names that are identical to the file names.

Once the raw data worksheet was generated and properly named, a new worksheet named **GenInfo** was created. The reference event date and the data start and stop times are then entered into the **GenInfo** worksheet. Two other worksheets, **MATLAB Summary Results** and **MATLAB Results for Graphs**, were also created to which results can be written. A sample portion of an Excel worksheet is shown in Figure 3.

DEVELOPMENT/DEPLOYMENT ENVIRONMENT

All programs are implemented in Matlab script language (Mathworks, Inc., Natick, MA) and run within the Matlab environment. For the purposes of modifying and enhancing the software in the future, functions were encapsulated to the greatest practical extent. In Matlab, external functions are required to be saved in a Matlab script file (m-file) of the same name. A list of all the m-files that make up the CIRCAD software is provided in Appendix A.

THE MAIN FUNCTION (CIRCAD)

The "main" function, used to call all other functions, is called *circad*. *circad* is located within the Matlab m-file of the same name (see Appendix B). *circad* calls the function *ReadCircadData,* which reads the core temperature, actigraphy, and sleep/wake data from the Excel file. Meshing the data into a single date/time reference with a constant sampling interval is accomplished using the function *MeshData.* Sleep/Wake data are transformed to a time-based vector using the same date/time reference as the core temperature and actigraphy data in *TransformSleepWake.* "S" values are recoded as 38.2, and "W" values as 0 in *TransformSleepWakeData* as well. Removal of outliers is accomplished in *RemoveOutliers.* Filtering of data is performed in the function *FFT_Filtering,* which uses Matlab's built-in fast Fourier transform function. The function *RunAnalyses* sets parameter values to be used in the model fitting procedures, calls the *DoCosinor* function, and prints and saves the results. *DoCosinor* is responsible for fitting a cosine curve to specified data intervals. All of these functions are listed in the appendices along with any supporting functions and are described in more detail below.

READING CIRCADIAN DATA (*READCIRCADDATA*)

ReadCircadData (Appendix C) is responsible for reading data from the Excel file. This is accomplished using Matlab's built-in functions for dynamic data exchange (DDE). At the start of the project, data existed in Excel worksheets in many different formats. To minimize the amount of reformatting required and to take advantage of data manipulation that had already been performed manually,

ReadCircadData was written to provide a certain amount of flexibility. Although it was necessary to insist on a standard set of variable names, it was not necessary to insist on standard locations for each variable within the data file. It was also possible to accommodate the three types of "date" formats that had been used; "date" could be entered either as a day of the month (Day), a date string (Date) or the number of elapsed days from the start of data collection (ElDays). Recognized variable names include the following: BCTM-Time, BCTM-Date, BCTM-Day, BCTM-ElDays, Pill Temp, Mini-Time, Mini-Date, Mini-Day, Mini-ElDays, Twrist, Tother, Activity, Sleep-Time, Sleep-Date, Sleep-ElDays, and Sleep/Wake. Variable names may occur anywhere within the first 30 columns and 15 rows of the worksheet. It is assumed that whatever falls below the variable name (in the same column) is the data for that variable. The only mandatory columns are BCTM-Time, and one of the BCTM "date" options (i.e., BCTM-Date, BCTM-Day, or BCTM-ElDays). If actigraphy or sleep/wake data occur without corresponding date and time columns, they are assumed to occur at the same time/date points as the BCTM data on the same row.

ReadCircadData initially reads data from an Excel file. However, because reading from Excel files is a slow process, the data are saved to a Matlab binary file (mat-file) for subsequent sessions (mat-files can be read much faster than the corresponding Excel files. *ReadCircadData* will read the mat file if it exists. Unless otherwise specified, all of the discussions that follow pertain to reading data from Excel files.

ReadCircadData performs the following tasks: (1) get the Excel file name using *GetFileName*, (2) check to see if a mat-file (Matlab binary file) exists using the function *DecideBetweenExcelAndMatFile*, (3) read the Excel or mat-file data (*ReadExcelData* or *ReadMatData*), and (4) if the data came from an Excel file, save the raw data as a mat file using the function *Save2MatFile*.

At the conclusion of ReadCircadData, a structure X exists that contains all the data read from the Excel or mat file. X currently includes the following fields:

PathName	Data path name (string)
FileName	File name (string)
FileHandle	File name without the extension (string)
BCTM_DateNum	Time and Date from BCTM as a date number (numeric array)
BCTM_Hour	Time and Date from BCTM as the elapsed hours since midnight of the first day (numeric array)
PillTemp	Pill temperature data (numeric array)
Mini_DateNum	Time and Date from Mini-Mitter as a date number (numeric array)
Mini_Hour	Time and Date from MiniMitter as the elapsed hours since midnight of the first day (numeric array)
Twrist	Wrist temperature data (numeric array)
Tother	Auxillary skin temperature data from Mini-

	Mitter (numeric array)
Activity	Actigraphy data (numeric array)
Sleep_DateNum	Time and Date from the sleep/wake log as a date number (numeric array)
Sleep_Hour	Time and Date from the sleep/wake log as the elapsed hours since midnight of the first day (numeric array)
SleepWake	Sleep/Wake onset from sleep/wake log (numeric or string array)
StartDateNum	Start Date for data collection as a date number
EventDateNum	Date of the reference event converted to a date number
DataStartDateNum	Start of "good" data as a date number
DataStopDateNum	End of "good" data as a date number

MESHING DATA INTO A SINGLE TIME/DATE REFERENCE (MESHDATA)

The process of meshing the BCTM, MiniMitter and Sleep/Wake data into a single Date/Time reference is straightforward, but requires a large amount of time if performed manually. The function *MeshData* (Appendix D) is found in a Matlab m-file of the same name. The first step in *MeshData* is to copy all of the non-time-based fields in X (i.e., PathName, FileName, FileHandle, StartDateNum, EventDateNum, DataStartDateNum, and DataStopDateNum) to a new structure, Y. This is accomplished in the function *CopyNonMeshedFields2Y*. The next step is to compute the appropriate start and stop times for valid data using the function *GetDataStartStopTimes*. The start time is either the first valid date/time in any of the device fields, **or** the data start time as entered into the **GenInfo** worksheet, whichever is later. The stop time is either the last valid date/time in any of the device fields, **or** the data stop time as entered into the **GenInfo** worksheet, whichever is earlier. Next, a new array field is created in structure Y, named Y.Time, to hold time (in hours). The first value in Y.Time is the start time, the last value is the stop time, and the interval between data points (Y.dTime) is one (1) minute or 1/60 hours. All of the time/date fields are rounded to the nearest minute. Finally, all of the time-based arrays are mapped into the new time reference using the function *DoTheMapping*.

The function *DoTheMapping* and the function it calls, *Map*, both take advantage of a special Matlab feature that allows array assignments to be made using arrays of index values. For example, if x=[11 22 33 44 55] and i=[2 5], then the statement y=x(i) will result in y=[22 55]. It is also possible to use the statement y(i)=x(4:5), which would result in y=[NaN 44 NaN NaN 55], where NaN is an empty placeholder in Matlab. NaN stands for "not a number".

For any of the devices (i.e., BCTM, MiniMitter or Sleep/Wake log), the difference in minutes between the *device*_Hour value and the start time will produce the position of the device's data values with respect to the new time reference. Adding 1 to these values gives the appropriate array index. These

array indices are named BCTM.i, Mini.i, and Sleep.i. *DoTheMapping* calls the function *Map* for each of the time-based data arrays (i.e., PillTemp, Twrist, Tother, Activity, and SleepWake). When *Map* is called, the data array is passed into the variable Data, and the array indices are passed into the variable iData. N is the size of the Y.Time array, and NumericYN tells *Map* whether the data array contains numeric data (1 for numeric, 0 for string).

The first thing *Map* does is check to see if Data is empty. If it is, then the mapped variable must also be empty. If Data is not empty, then the mapping process is started. First, Data and iData are trimmed to remove elements corresponding to data collected before the start time or after the stop time. This is done by identifying values in iData which are greater than or equal to 1 and less than or equal to N. This valid interval for Data, called iGoodData, is used to trim the arrays: Data=Data(iGoodData) and iData=ix(iGoodData). The array that Data is to be mapped into (DataMapped) is first filled with empty placeholders (numeric arrays only). The empty placeholder for numeric arrays in Matlab is the value NaN. The mapping process itself is accomplished using the command DataMapped=Data(iData). When Matlab assigns array values using a variable index, the size of the new array is set to the largest value in the array index. In this case, the size of DataMapped is set to the largest value in iData. The largest value in iData is not necessarily equal to N, resulting in DataMapped being smaller than the time reference array (Y.Time). To get around this, a series of statements in *Map* beginning with "if length(DataMapped)<N" assigns a value to the Nth value of DataMapped (NaN for numeric arrays and the last "good" string value for string arrays) which forces the size of the mapped array to be equal to N. The final meshed data are named Y.Time, Y.Traw, Y.Twrist, Y.Tother, Y.Activity, and Y.SleepWake. All of these arrays except Y.SleepWake are written to the Excel worksheet named **MATLAB Results for Graphs**. Upon returning to the calling function, *circad*, X is reassigned the values in Y.

TRANSFORM SLEEP/WAKE DATA (*TRANSFORMSLEEPWAKE*)

The function *TransformSleepWake* (Appendix E) takes sleep/wake event data (X.SleepWake), where the event "sleep" is coded either with the numerical value of 38.2 or the letter "S", and "wake" is coded either with the numerical value of 0 or the letter W. For string data (i.e., letter codes), *TransformSleepWake* converts these data to a numerical array in which S and W are represented by their corresponding ASCII codes (83 and 87, respectively) and missing values are coded zero (0). If the first value in the array is missing (likely because the data have already been mapped to the common date/time reference), it is coded with the value 87 (assuming that the subject is awake at the start of data collection). Other missing values are filled in by dragging the previous value along. After all missing values have been filled in, all values of 83 are converted to 38.2 (a convenient value for representing sleep periods on core temperature plots), and values of 87 are converted to the value 0. Transformed SleepWake data are written to the Excel worksheet named **MATLAB Results for Graphs** prior to returning to the calling function.

REMOVING OUTLIERS (*REMOVEOUTLIERS*)

The function *RemoveOutliers* (Appendix E) takes the array X.Traw and copies it to X.Tclean. Core temperature values in X.Tclean that are strings, or that are less than 35 or greater than 40 are set equal to NaN. Cleaned-up core temperature data (X.Tclean) are written to the Excel worksheet named **MATLAB Results for Graphs** prior to returning to the calling function.

FILTERING USING FFT (FFT_FILTERING)

Filtering is performed using the Fast Fourier Transform (FFT). The function *FFT_Filtering* (Appendix F) uses Matlab's built-in FFT (*fft*) and Inverse FFT (*ifft*) functions. *FFT_Filtering* starts with the cleaned-up core temperature vector, X.Tclean. To use *fft* and *ifft*, it is necessary for data to be equally spaced in time. Because the time array (X.Time) is already equally-spaced at 1-minute intervals, this requires that all missing temperature values (i.e., values of NaN in X.Tclean) be assigned some kind of estimated temperature value.

Filling in missing temperature values is accomplished in the function *FillTemperatureBlanks*. For FFT analysis, missing value estimation may consist of either setting the missing data points to the average value of the entire data series, or of setting any missing data point equal to the previous, good data point[1]. The latter approach is used in *FillTemperatureBlanks*. It is possible that the first data point(s) in the data series may also contain missing data. In this case, starting with the first "good" data point and moving backwards in the series, any missing data point is set equal to the next good data point in the series.

Once a valid core temperature vector is obtained (named X.T), the FFT is performed, producing the vector FFTofT. FFT frequency (freq), period, and magnitude vectors are also computed.

Filtering is accomplished by selecting a frequency cut-off point, and setting all elements of FFTofT that correspond to frequencies greater than the cut-off value to zero. The cut-off frequencies used in this version of CIRCAD are 0.25, 0.50, and 1.00 hours^{-1}, corresponding to cut-off periods of 4, 2, and 1 hours, respectively. Taking the inverse of the filtered FFTofT vector produces the "filtered" vectors of core temperature with respect to time (i.e., X.Tfilt1, X.Tfilt2, X.Tfilt4, which correspond to cut-off periods of 1, 2, and 4 hours, respectively).

The final step in the *FFT_Filtering* function is to plot the results and send the filtered arrays (X.Tfilt1, X.Tfilt2, and X.Tfilt3) to the Excel worksheet **MATLAB Results for Graphs**.

RUN ANALYSES

All analyses are performed by calling the *RunAnalyses* function (Appendix B) from within *circad*. *RunAnalyses* first selects one of the 48 different conditions to run. Each condition includes:

1. The variable (Var) to fit (i.e., X.T, X.Tfilt1, X.Tfilt2, or X.Tfilt4);
2. The method (PeriodStr) for obtaining the circadian period (i.e, "free" in which the cosine-fit procedure estimates the period, "24-hour", in which a constant period of 24 hours is used, or "Lomb" in which the function *LombPeriod* (Appendix G) is used to compute the circadian period);

3. The data interval (Interval) to use (i.e., whole curve, individual 24-hour (midnight to midnight) days, individual 24-hour (noon to noon) days, or 36-hour (6pm to 6am) segments).

RunAnalyses then calls the appropriate analytical function, passing the appropriate time and temperature vectors, as well as a numeric code representing the period, in the function call. In the current version of CIRCAD, there is only one analytical function named *DoCosinor*, which performs a cosine fit to the data.

The result of calling the analytical function (e.g., *DoCosinor*) is to return a data structure containing the results to the variable IntervalResults. IntervalResults may describe either a fit to the entire data set, or to a 24- or 36-hour segment of the data set. For the case in which the interval is the entire data set, IntervalResults is copied directly into a structure named Results. In the case where the intervals are 24- or 36-hours, IntervalResults structures are appended together into the Results structure to cover the entire period of data collection. For example, if there are 5 days of data, then *DoCosinor* is run once for each of the 5 days. The resulting IntervalResults structures are appended together to form the structure named Results. Although the Results structure contains the same fields as IntervalResults, the fields are generally arrays rather than scalars (one array element for each day). Also, the Tpred and t arrays would contain data from all 5 days. Note that in the case of 36-hour data windows, values of Tpred and t from midnight to midnight are appended together; the 6-hour data segments preceding and succeeding the 24-hour period are not saved.

Once the results are appended together, predicted values are plotted along with whatever data were used to obtain the predicted values (i.e., "cleaned-up" or filtered). Predicted values are also plotted against the unfiltered data. These plots are automatically saved as .jpg files. Predicted values are also written to the Excel worksheet named **MATLAB Results for Graphs** for later plotting. Values of summary variables (e.g., mesor, amplitude, period, etc.) are written to the Excel worksheet named **MATLAB Summary Results**.

COSINOR ANALYSES (DOCOSINOR)

DoCosinor is called from RunAnalyses to perform a cosine fit to the data passed to it. *DoCosinor* is listed in Appendix H and is responsible for computing the following fields of the structure named cosinor:

Tpred	Predicted temperature using cosinor model and fit parameters
t	The time interval used
mesor	The mesor (y-offset, estimated mean temperature) as computed by the cosinor regression.
amplitude	The amplitude as computed by the cosinor regression
period	The period as computed by the cosinor regression (if applicable), 24, or the Lomb

	period.
phase	The phase as computed by the cosinor regression (for example, -8.75)
gphase	The phase relative to the sampling interval (for example, 39.35)
clock_phase	The phase relative to clock time (for example, 15.25).
phase_clockstring	The phase as a text string (for example, 15:15).
R	Correlation coefficient (R-value) for the regression
N	Number of data points used in the regression
MaxData	Maximum of the data values
TimeOfMaxData	Time (clock hours) corresponding to the maximum of the data values
MinData	Minimum of the data values
TimeOfMinData	Time (clock hours) corresponding to the minimum of the data values
MaxPred	Maximum of the predicted values
TimeOfMaxPred	Time (clock hours) corresponding to the maximum of the predicted values
MinPred	Minimum of the predicted values
TimeOfMinPred	Time (clock hours) corresponding to the minimum of the predicted values

Mesor, amplitude, period, and the various acrophase parameters (phase, gphase, clock_phase, and phase_clockstring) are obtained by fitting a cosine function to the time and Temperature data that are passed to the DoCosinor function. To obtain values for these fields, *DoCosinor* first assigns initial values to the 3 or 4 parameters (p) that will be fit. The number of parameters depends on whether the period is to be computed during the cosine fit procedure (Period==0), or whether period is set at 24-hours (Period==24) or computed using the Lomb analysis method. The first three initial parameter values correspond to the mesor (°C), amplitude (°C) and acrophase (radians), and the intial values are p(1)=40, p(2)=0.5, and p(3)=1, respectively. If the period is to be computed using the cosine fit procedure, then a fourth parameter is used which corresponds to the frequency (radians/hour) with an initial value of p(4)=0.25. These initial values were selected arbitrarily and have been found to produce satisfactory fits for all data analyzed to date.

Once initial values have been specified, *DoCosinor* calls either sumsqerror or sumsqerrorFP, depending on whether the period is to be computed or has already been set, respectively. The sumsqerror and sumsqerrorFP functions minimize the prediction error of the function *CosinorTpred*, which uses the cosine function to predict core temperature:

Tpred=p(1)+p(2)*cos(p(4)*t+p(3))

where 2*pi/period is substituted for p(4) when the period is not to be computed using the cosinor fit procedure.

The values of the cosinor fields are computed after the completion of the parameter optimization step:

- Tpred is computed by calling the *CosinorTpred* function using the optimized parameter values
- t is copied from the time vector that is passed to *DoCosinor*
- N is set to the length of the t vector
- R, the correlation coefficient between T and Tpred, is computed using Matlab's built-in function named *corrcoef*
- mesor and amplitude correspond directly to the first and second optimized parameter values, p(1) and p(2), respectively
- period is given by p(4), the frequency parameter, by taking the reciprocal of p(4), then converting from radians to cycles: period = $2\pi/p(4)$.
- phase ("acrophase", in hours) is computed using p(3), the phase in radians, and the period (in hours/cycle, computed in the step above):
  phase = -[p(3)/(2*π)]*period

For ease in viewing the location of the phase on the composite graphs (with time in hours on the x-axis), "gphase" is computed by repeatedly adding the period length to phase (gphase = phase + period + period +...) until gphase is greater than the first time value t(1) in the interval. "clock_phase" (i.e., acrophase in hours after the previous midnight) and "phase_clockstring" (i.e., clock_phase as a time string) are computed by repeatedly subtracting 24 hours from phase (clock_phase = phase –24 –24 -...) until a valid clock time (i.e., between 0 and 24 hours) is obtained.

Matlab's built in *max* and *min* functions are applied both to the data values (T) and the predicted values (Tpred) to obtain both the max and min values, and also the array indices corresponding to the max and min values. From the indices, it is possible to obtain the corresponding time of the max and min values. The final time values are computed by repeatedly subtracting 24 hours from the time value (e.g., TimeOfMax = TimeOfMax –24 –24 -...) until a valid clock time (i.e., between 0 and 24 hours) is obtained.

MISCELLANEOUS FUNCTIONS

Utility functions that are called by more than one function or that are utilized by other applications are contained in their own m-files. These m-files are listed in Appendix I.

RESULTS

SAMPLE INPUT/OUTPUT

Sample output from CIRCAD are shown in Figures 4 through 8. Three types of figures are produced in MATLAB during CIRCAD program execution. The first shows the results of the FFT_Filtering process (Figure 4). The topmost plot is of the raw data. The next plot shows the FFT-generated periodogram. The final three plots show the filtered data (frequency cut offs set to 1, 0.5, and 0.25 hours^{-1}, respectively). The next type of figure (Figure 5) shows the results of one of the 48 different variations of the cosinor method. In the example shown, a

36-hour data window was used, the cosinor was used to fit the circadian period, and filtered data ($f>0.5\ h^{-1}$) was used. For those variations in which the Lomb method was used to obtain the dominant (i.e., circadian) period, a Lomb periodogram is also displayed. The example Lomb periodogram in Figure 6 corresponds to a variation using a 36-hour data window using cleaned-up (not filtered) data.

CIRCAD also writes results back to the Excel data file so that publication-quality graphics can be generated using SigmaPlot. The time-based data, including the raw, cleaned up, and filtered core temperature data as well as all of the cosinor fits are written to the worksheet named **MATLAB Results for Graphs**. An example of the part of the worksheet containing the raw, cleaned up, and filtered core temperature data is shown in Figure 7a. Figure 7b shows the results of one of the 48 different variations of the cosinor method in which a 36-hour data window was used, the cosinor was used to fit the circadian period, and cleaned-up data (not filtered) was used for the fit.

In addition to the time-based results, CIRCAD also writes summary data, containing the parameter values for each of the cosinor fits, to the worksheet named **MATLAB Summary Results**. An example from a **MATLAB Summary Results** worksheet showing the results of a single variation of the cosinor fit method is provided in Figure 8.

TIME SAVINGS

Manually performing the data manipulation, curve fitting and plotting using Excel and Sigmaplot required approximate 3½ days for one cosinor fit (i.e., a cosinor using a constant period of 24 hours and midnight to midnight data windows on raw data). Using the CIRCAD software, this process was shortened considerably. It now requires approximately 30 minutes to read and process the data, and to perform 48 variations of the cosinor method (i.e., variations in circadian period, data windows, and data filtering). It requires an additional 20 minutes to plot the results, which is performed using Sigmaplot templates.

DISCUSSION: PROBLEMS/LIMITATIONS AND FUTURE DIRECTIONS

The current version of CIRCAD utilizes only the cosinor method for analyzing core temperature data. We intend to explore other signal processing and curve-fit methods for obtaining circadian parameters in future versions of the software. Also, we would like to apply these methods to activity and skin temperature data as well as body core temperature data.

Although the CIRCAD software significantly reduces the amount of manual labor involved in analyzing body core temperature, there is still a large amount of tedious labor (approximately 30 minutes per data set). We would like to reduce this as far as is practical. This may include obtaining sleep/wake data directly from the actigraphy monitor, modifying the way data are read and written

to the Excel worksheets, and creating a method for performing sequential analyses on multiple data sets (i.e., enabling a "batch" mode).

Certain limitations are also imposed by current "bugs" and limitations in the DDE functions in the Matlab programming environment. Currently, it is not possible to automatically start up Excel and open the desired input data file; the user must perform these operations manually. It is not possible to read from more than one data file, or to read from one file and place results in another (although it is possible to work with multiple worksheets within a single Excel file). Because the file is normally re-saved after results are written to it from CIRCAD, the number of instances in which it is possible to save time by reading from a mat-file is severely reduced. Finally, as was mentioned previously, the DDE functions do not pay attention to workbook (i.e., file) names, so input worksheet names must be unique to prevent mistakes in reading the data. The next version of the CIRCAD software will likely utilize ActiveX (as opposed to DDE) methods to communicate with Excel. Matlab's ActiveX functions have so far shown to overcome many of the limitations associated with DDE in terms of reading data from Excel. Also, ActiveX is a newer technology that is rapidly replacing DDE as the standard.

REFERENCES

1. Press, W.H., B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling. Numerical Recipes in C: The Art of Scientific Computing. Cambridge University Press, New York, 1992.
2. Coyne, M.D. and L.A. Stephenson. Personal Communication, 1999.

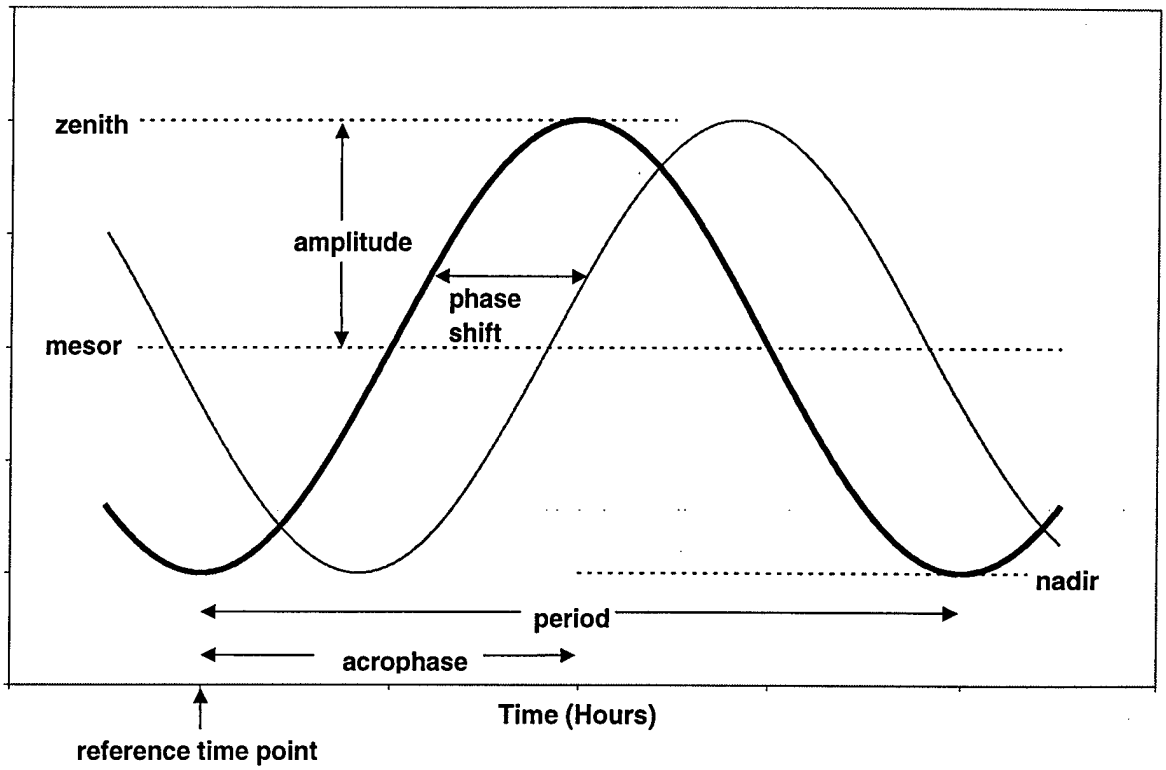

Figure 1. Parameters of a Circadian (approximately 24-hour) Rhythm.

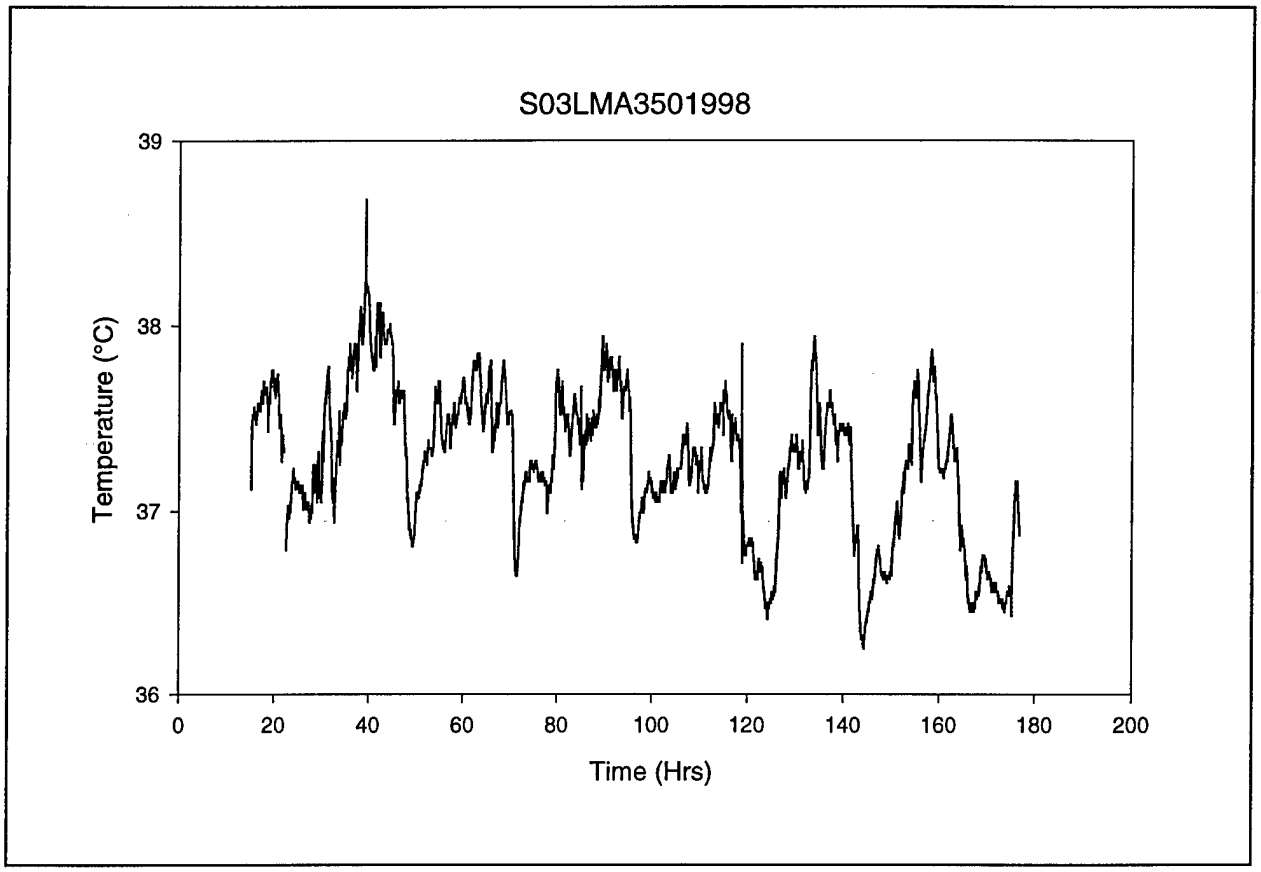

Figure 2. Plot demonstrating day to day variability in the circadian core temperature rhythm (from M. Coyne and L. Stephenson, unpublished[2]).

BCTM-Day	BCTM-Time	Pill Temp	Sleep-ElDays	Sleep-Time	Sleep/Wake	Mini-Date	Mini-Time	Twrist	Activity
30	13:55:23	36.4	0	18:00:00	S	5/1/99	8:01:00	24.71	8
30	13:56:23	36.58	0	23:00:05	W	5/1/99	8:02:00	24.71	52
30	13:57:23	36.69	1	1:30:00	S	5/1/99	8:03:00	24.71	60
30	13:58:23	36.75	1	6:30:00	W	5/1/99	8:04:00	24.71	124
30	13:59:23	36.78	1	22:00:00	S	5/1/99	8:05:00	24.71	125
30	14:00:23	36.82	2	7:00:00	W	5/1/99	8:06:00	24.71	165
30	14:01:23	36.86				5/1/99	8:07:00	24.71	133
30	14:02:23	36.89				5/1/99	8:08:00	24.71	159
30	14:03:23	36.93				5/1/99	8:09:00	24.71	121
30	14:04:23	36.93				5/1/99	8:10:00	24.71	20
30	14:05:23	36.95				5/1/99	8:11:00	24.71	60
30	14:06:23	36.97				5/1/99	8:12:00	24.71	152
30	14:07:23	36.97				5/1/99	8:13:00	24.71	71
30	14:08:23	37.02				5/1/99	8:14:00	24.71	112
30	14:09:23	37.02				5/1/99	8:15:00	24.71	22
30	14:10:23	37.02				5/1/99	8:16:00	24.71	0
30	14:11:23	37.02				5/1/99	8:17:00	29.03	77
30	14:12:23	37.04				5/1/99	8:18:00	29.5	16
30	14:13:23	37.06				5/1/99	8:19:00	29.8	23
30	14:14:23	37.06				5/1/99	8:20:00	30.06	33
30	14:15:23	37.08				5/1/99	8:21:00	30.37	0
30	14:16:23	37.11				5/1/99	8:22:00	30.57	4
30	14:17:23	37.13				5/1/99	8:23:00	30.88	3
30	14:18:23	37.15				5/1/99	8:24:00	31.04	4

Figure 3. A sample from an Excel data worksheet.

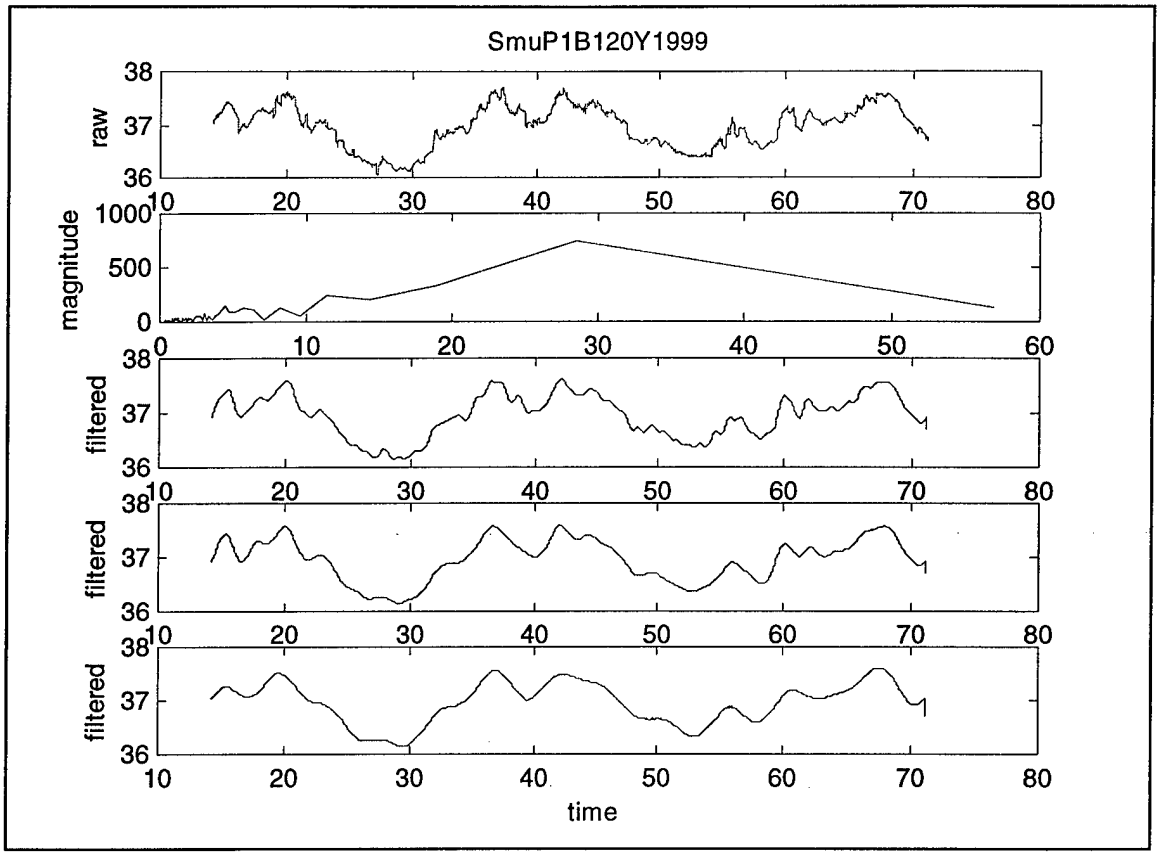

Figure 4. Example plot produced by the FFT_Filtering function. The first graph is of core temperature data (X.T) versus time. The second graph is the FFT periodogram. The last three graphs show the effects of filtering the core temperature data using cutoff periods of 1-hour, 2-hours, and 4-hours, respectively.

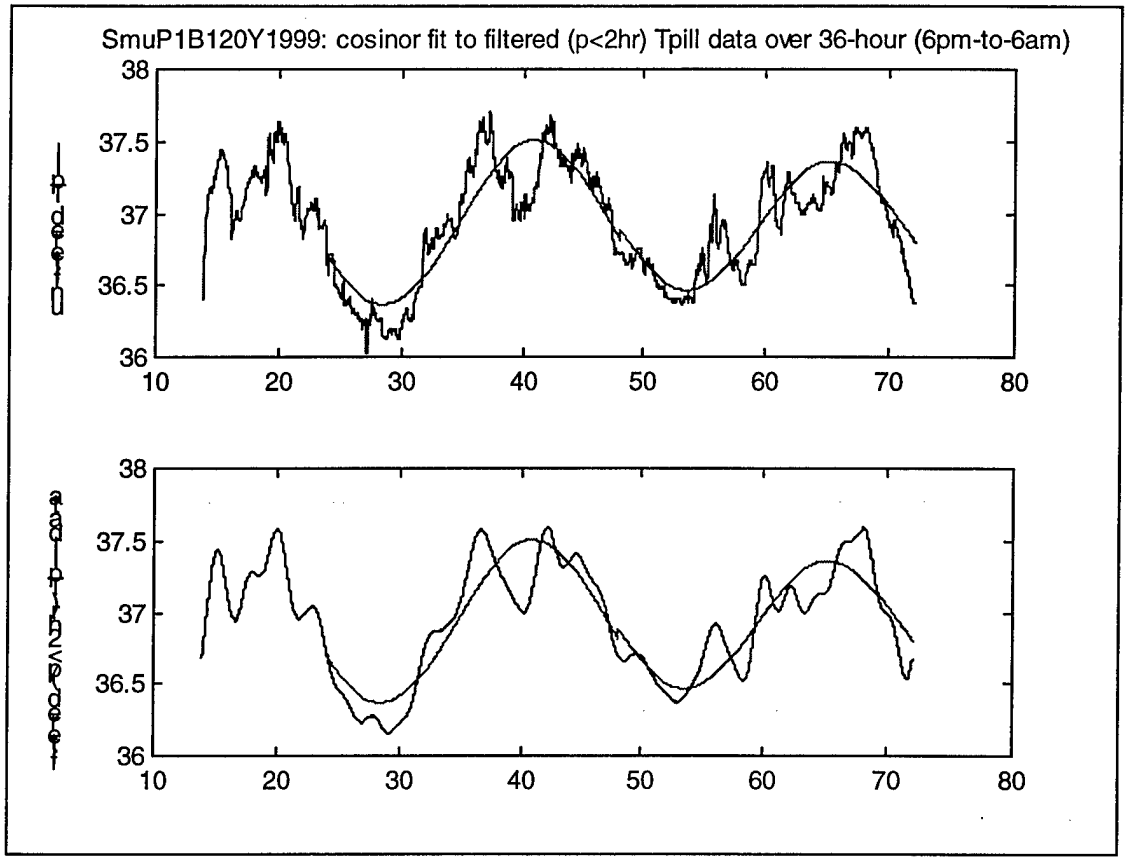

Figure 5. Example MATLAB-generated plots showing the cosinor fit against the cleaned-up raw data (top) and also against the variable that was fit, in this case filtered (cutoff frequency = 0.5 hours^{-1}) core temperature data.

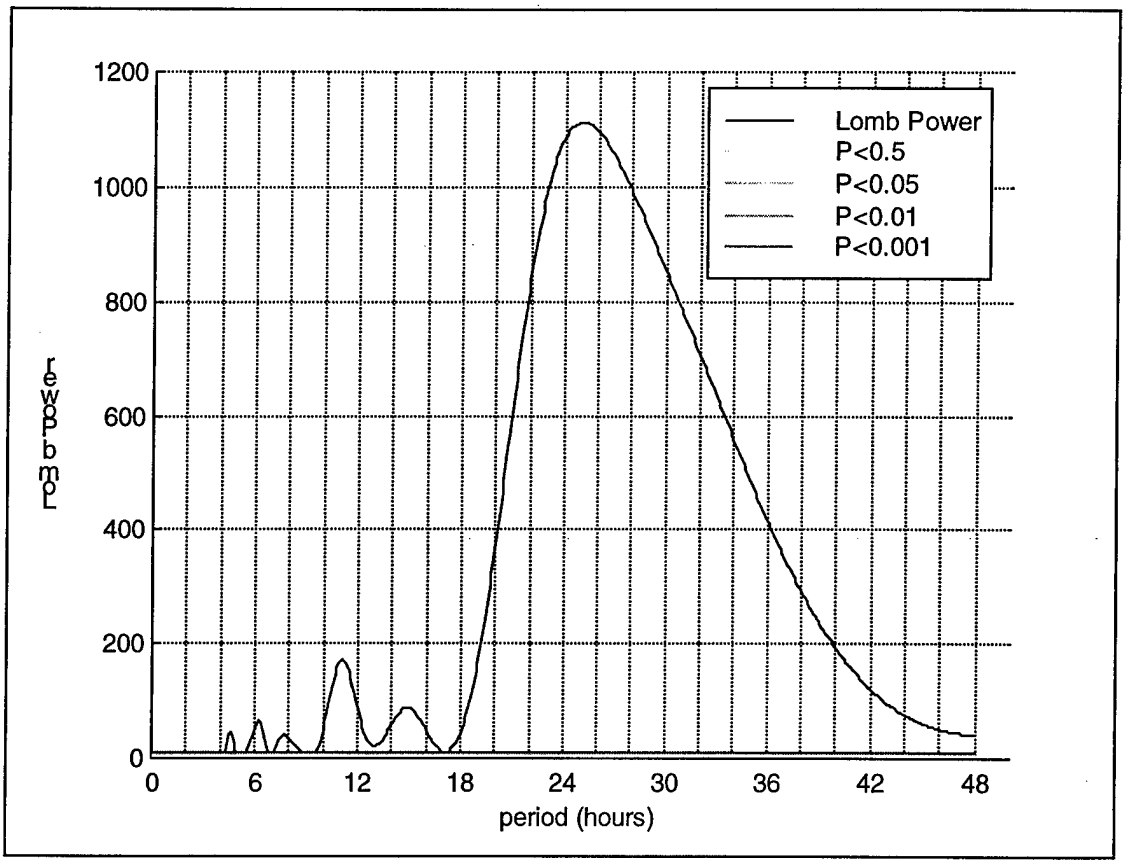

Figure 6. Example of a Lomb Periodogram produced by CIRCAD.

Time	Traw	Activity	Twrist	Sleep	Tcleaned	Tfilled	Tfilt1	Tfilt2	Tfilt4
13.92	36.40	0	0	0	36.40	36.40	36.63	36.69	36.88
13.93	36.58	0	0	0	36.58	36.58	36.65	36.70	36.88
13.95	36.69	0	0	0	36.69	36.69	36.67	36.71	36.89
13.97	36.75	0	0	0	36.75	36.75	36.69	36.72	36.89
13.98	36.78	0	0	0	36.78	36.78	36.71	36.73	36.90
14.00	36.82	0	0	0	36.82	36.82	36.73	36.74	36.91
14.02	36.86	0	0	0	36.86	36.86	36.76	36.75	36.91
14.03	36.89	0	0	0	36.89	36.89	36.78	36.76	36.92
14.05	36.93	0	0	0	36.93	36.93	36.80	36.78	36.93
14.07	36.93	0	0	0	36.93	36.93	36.82	36.79	36.93
14.08	36.95	0	0	0	36.95	36.95	36.84	36.80	36.94
14.10	36.97	0	0	0	36.97	36.97	36.87	36.81	36.94
14.12	36.97	0	0	0	36.97	36.97	36.89	36.83	36.95
14.13	37.02	0	0	0	37.02	37.02	36.91	36.84	36.96
14.15	37.02	0	0	0	37.02	37.02	36.93	36.85	36.96
14.17	37.02	0	0	0	37.02	37.02	36.95	36.87	36.97
14.18	37.02	0	0	0	37.02	37.02	36.97	36.88	36.98
14.20	37.04	0	0	0	37.04	37.04	36.99	36.89	36.98
14.22	37.06	0	0	0	37.06	37.06	37.01	36.91	36.99
14.23	37.06	0	0	0	37.06	37.06	37.03	36.92	36.99

Figure 7a. Example output from CIRCAD. This figure shows the first several columns of the worksheet "MATLAB Results for Graphs" which holds the meshed raw data (Traw), as well as the cleaned-up raw data (Tcleaned), cleaned-up raw data with estimated missing values (Tfilled), and filtered data using cutoff periods of 1, 2, and 4 hours (Tfilt1, Tfilt2, and Tfilt4, respectively).

Time	SmuP1B120Y1999: cosinor fit to cleaned-up Tpill data over 36-hour (6pm-to-6am) period set to 24 hours
13.92	0
13.93	0
...	...
23.97	0
23.98	0
24.00	36.74750838
24.02	36.74520584
24.03	36.74290698
24.05	36.74061183
24.07	36.73832044
24.08	36.73603286
24.10	36.73374912
24.12	36.73146927
24.13	36.72919335
24.15	36.72692141
24.17	36.72465349
24.18	36.72238963
24.20	36.72012988
24.22	36.71787427
24.23	36.71562286
24.25	36.71337568
24.27	36.71113278
24.28	36.7088942
24.30	36.70665999
24.32	36.70443018

Figure 7b. Example output from CIRCAD. This figure shows a results column in the worksheet named "MATLAB Results for Graphs". This column is the result of fitting cosine curves to 36-hour data windows, allowing the cosinor function to fit the period, and working with unfiltered data. Results for times 13.92 to 23.98 are set to zero (the missing value code) because curve fitting starts at midnight of the first day of data collection.

SmuP1B120Y1999:cosinor fit toTpill, cleaned, over 36-hour (6pm-to-6am)								
Day (rel to Event)	mesor	amplitude	period	gphase	clock_phase	phase_clockstring	R	N
-9.00	36.94	0.57	24.83	40.92	16.92	16:55	0.91	2142.00
-8.00	36.91	0.45	22.92	65.11	17.11	17:06	0.86	1789.00

MaxData	TimeOfMaxData	MinData	TimeOfMinData	MaxPred	TimeOfMaxPred	MinPred	TimeOfMinPred
37.70	13.20	36.03	3.18	37.51	16.92	36.37	5.33
37.68	18.23	36.36	5.10	37.36	18.18	36.46	5.65

Figure 8. Example output from CIRCAD. This is a portion of the "MATLAB Summary Results" worksheet showing the fit parameters that result from fitting cosine curves to 36-hour data windows, allowing the cosinor function to fit the period, and working with unfiltered data.

APPENDIX A

FILE LIST

Circad.m
CircadianDataPath.m
CosinorTpred.m
DecodeCircadFileName.m
DecodeJulian.m
DoCosinor.m
FFT_Filtering.m
GetFileHandle.m
InitializeDDE.m
Lomb.M
LombPeriod.m
MeshData.m
PokeTimeBasedColumn.m
ReadCircadData.m
RemoveOutliers.m
Sumsqerror.m
SumsqerrorFP.m
TestDDEConnection.m
TransformSleepWake.m
WriteSummaryResultsByMethod.m

APPENDIX B
CIRCAD.M

```
%-----------------------------------------------------------------------
%Circad.m                ver 1.12                              3 DEC 1999
%-----------------------------------------------------------------------
function circad
% CIRCAD  limited version 1.12 by Tammy Doherty, USARIEM
% Analyzes circadian pill temperature data.

%get rid of old variables and graphs
ClearWorkspace

%Read data from .mat or excel file
X=ReadCircadData;

%Mesh BCTM, Minimitter and sleep/wake data to common date/time reference
X=MeshData(X);

%Transform Sleep/Wake data to numeric values (38.2 for sleep, 0 for awake)
X=TransformSleepWake(X);

%Remove outliers from pill temperature data
X.Tclean=RemoveOutliers(X.Traw);

% Filter Data to exclude periods less than 1, 2, or 4 hours.
X=FFT_Filtering(X);

%Specify analyses to run:
%In this version, only one Analysis type is available:
AnalysisType(1)={'cosinor fit'};
AnalysisFunction(1)={'DoCosinor'};

%Perform analyses using raw or filtered data,
%using data intervals of various widths and starting locations,
%and using various methods of obtaining circadian period.
RunAnalyses(X,AnalysisType,AnalysisFunction);

return
%-----------------------------------------------------------------------
%------------------------ FUNCTION DEFINITIONS -------------------------
%-----------------------------------------------------------------------

function ClearWorkspace
   close all;
   clear all;
return
%-----------------------------------------------------------------------
function RunAnalyses(X,AnalysisType,AnalysisFunction)
   for i=1:prod(size(AnalysisType))
      disp(strcat('starting ',AnalysisType,' ...'));
      disp(' ');

      [Var,VarDesc]=GetVar2Use(X);
      while length(Var)>0
         [PeriodStr,PeriodDesc]=GetPeriodType2Use(X);
         while length(PeriodStr)>0

            Results=[];
            [Interval,IntervalDesc,Day,RelDay,LastDayYN,AppendYN]=GetInterval2Use(X);
            Period=AssignNumericCodeToPeriod(PeriodStr,X,Var,Interval)

            while length(Interval.X1)>0
               Title=strcat(X.FileHandle,': ',char(AnalysisType(i)),' to',...
                  VarDesc,' over',IntervalDesc,PeriodDesc);
```

```
                %get the results for the specified interval
                IntervalResults=feval(char(AnalysisFunction(i)),...
                   X.Time(Interval.Analysis1:Interval.Analysis2),...
                   Var(Interval.Analysis1:Interval.Analysis2),Period);

                %if results need to be appended together, do it.
                %Otherwise, the final results array equals the current array
                if AppendYN==1
                   Results=AppendResults(Results,IntervalResults,X,Day,RelDay,Interval);
                else
                   Results=IntervalResults;
                end

                %new fields (added outside of the analytical routines)
                if (AppendYN==1&Day==1)|AppendYN==0
                   Results.StartDayRelative2Event=RelDay;
                   Results.Title=Title;
                   Results.t=X.Time;
                end

                %put spacers at the beginning of the appended array so the length
                %and times correspond to X.Time
                if ~isempty(Day)
                   if Day==1
                      %fill all arrays up to i1 with NaN
                      Results=InitResults(Results,Interval.Append1-1);
                   end
                end

                %Output results
                if (AppendYN==0)|(LastDayYN==1)
                   %save summary results to an Excel worksheet
                   WriteSummaryResultsByMethod(Results);
                   %plot results
                   PlotResults(Results,X,Var,VarDesc);
                   %print graph to file

GraphFileName=strcat(X.FileHandle,'_Period',num2str(Period),'_',IntervalDesc,'_',VarDesc,
'.jpg');
                   print('-djpeg90',fullfile(X.PathName,GraphFileName));
                      %save time-based results to an Excel worksheet
                      PokeTimeBasedColumn(Results.Title,Results.Tpred);
                   end


    [Interval,IntervalDesc,Day,RelDay,LastDayYN,AppendYN]=GetInterval2Use(X);
             end
             [PeriodStr,PeriodDesc]=GetPeriodType2Use(X);
       end
        [Var,VarDesc]=GetVar2Use(X);
      end
   end

   disp('DONE!');

return
%---------------------------------------------------------------------

function PlotResults(Results,X,Var,VarDesc)
persistent FigNum
    if isempty(FigNum)
       FigNum=3;
    end
   N=length(Results.Tpred);
   i=find(Results.Tpred==0);
   Results.Tpred(i)=NaN;
   figure(FigNum);
    subplot(2,1,1),plot(X.Time,X.T,'b'),ylabel('unfiltered Tpill'),title(Results.Title);
   hold on
   %plot(X.Time,Var,'k');
   plot(X.Time(1:N),Results.Tpred(1:N),'r.')
```

```
    hold off
    subplot(2,1,2),plot(X.Time,Var,'b'),ylabel(VarDesc);
     hold on
    plot(X.Time(1:N),Results.Tpred(1:N),'r.')
    hold off
    %FigNum=FigNum+1;
    return

%-------------------------------------------------------------------------
function [composite]=AppendResults(composite,OneDay,X,Day,RelDay,Interval)
    if Day==1
    composite=[];
    end
   composite.t(Interval.X1:Interval.X2,1)=...
      X.Time(Interval.X1:Interval.X2);
   composite.Tpred(Interval.X1:Interval.X2,1)=...
      OneDay.Tpred(Interval.Append1:Interval.Append2);
    composite.R(Day,1)=OneDay.R;
    composite.N(Day,1)=OneDay.N;
    composite.MaxData(Day,1)=OneDay.MaxData;
    composite.MinData(Day,1)=OneDay.MinData;
    composite.MaxPred(Day,1)=OneDay.MaxPred;
    composite.MinPred(Day,1)=OneDay.MinPred;
    composite.TimeOfMaxData(Day,1)=OneDay.TimeOfMaxData;
    composite.TimeOfMinData(Day,1)=OneDay.TimeOfMinData;
    composite.TimeOfMaxPred(Day,1)=OneDay.TimeOfMaxPred;
    composite.TimeOfMinPred(Day,1)=OneDay.TimeOfMinPred;
    composite.mesor(Day,1)=OneDay.mesor;
    composite.amplitude(Day,1)=OneDay.amplitude;
    composite.period(Day,1)=OneDay.period;
    composite.phase(Day,1)=OneDay.phase;
    composite.gphase(Day,1)=OneDay.gphase;
    composite.clock_phase(Day,1)=OneDay.clock_phase;
    composite.phase_clockstring(Day,:)=OneDay.phase_clockstring(:);
   composite.DayRelative2Event(Day,1)=RelDay;
   return
%-------------------------------------------------------------------------
function [composite]=InitResults(composite,i1)
   composite.Tpred(1:i1,1)=NaN;
   composite.t(1:i1,1)=NaN;
return
%-------------------------------------------------------------------------
function [Var,VarDesc]=GetVar2Use(X)
persistent VarNum
    NumVars=4;
    if isempty(VarNum)
      VarNum=1;
   else
      VarNum=VarNum+1;
   end
   if VarNum<=NumVars
      switch VarNum
      case 1
         Var=X.Tclean;
         VarDesc='Tpill_cleaned';
      case 2
         Var=X.Tfilt1;
         VarDesc=' Tpill_filt(p=1hr)';
      case 3
         Var=X.Tfilt2;
         VarDesc=' Tpill_filt(p=2hr)';
      case 4
         Var=X.Tfilt4;
         VarDesc=' Tpill_filt(p=4hr)';
      case 5
         Var=X.Twrist;
         VarDesc=' Twrist_raw';
      end
   else
      VarNum=[];
      Var=[];
```

```
      VarDesc=[];
   end
return
%-----------------------------------------------------------------------
function [Interval,IntervalDesc,Day,RelDay,LastDayYN,AppendYN]=GetInterval2Use(X)
persistent IntervalNum
persistent NumIntervals
persistent CircadianInterval
persistent FirstMidnightIndex
persistent FirstNoonIndex
persistent NumM2MDays
persistent NumN2NDays
persistent N

    if isempty(IntervalNum)
      IntervalNum=1;
      N=length(X.Time);
      CircadianInterval=round(24/X.dTime);
      FirstMidnightIndex=round((24-X.Time(1))/X.dTime+1);
      while FirstMidnightIndex<1
         FirstMidnightIndex=FirstMidnightIndex+CircadianInterval;
      end
      LastMidnightIndex=FirstMidnightIndex+CircadianInterval;
      while LastMidnightIndex<=N
         LastMidnightIndex=LastMidnightIndex+CircadianInterval;
      end
      LastMidnightIndex=LastMidnightIndex-CircadianInterval;
      FirstNoonIndex=round((12-X.Time(1))/X.dTime);
      while FirstNoonIndex<1
         FirstNoonIndex=FirstNoonIndex+CircadianInterval;
      end
      LastNoonIndex=FirstNoonIndex+CircadianInterval;
      while LastNoonIndex<=N
         LastNoonIndex=LastNoonIndex+CircadianInterval;
      end
      LastNoonIndex=LastNoonIndex-CircadianInterval;
      NumM2MDays=floor((LastMidnightIndex-FirstMidnightIndex+1)/CircadianInterval);
      NumN2NDays=floor((LastNoonIndex-FirstNoonIndex+1)/CircadianInterval);
      NumIntervals=NumM2MDays*2+NumN2NDays+1;
   else
      IntervalNum=IntervalNum+1;
   end

   LastDayYN=0;

   if IntervalNum<=NumIntervals
      if IntervalNum==1  %Do Analysis and Lomb on whole curve
         Day=[];
         RelDay=[];
         AppendYN=0;
         Interval.X1=1;
         Interval.X2=N;
         Interval.Analysis1=1;
         Interval.Analysis2=N;
         Interval.Lomb1=1;
         Interval.Lomb2=N;
         Interval.Append1=1;
         Interval.Append2=N;
         IntervalDesc=' all';
      elseif (IntervalNum>1)&(IntervalNum<=NumM2MDays+1)
         %Do Analysis M2M, Lomb +/-12 from Midnight-to-Midnight
         Day=IntervalNum-1;
         if Day==NumM2MDays
            LastDayYN=1;
         end
         AppendYN=1;
         RelDay=Day+X.StartDateNum-floor(X.EventDateNum);
         Interval.X1=FirstMidnightIndex+(Day-1)*CircadianInterval;
         Interval.X2=FirstMidnightIndex+Day*CircadianInterval;
         Interval.Analysis1=Interval.X1;
         Interval.Analysis2=Interval.X2;
```

```
            Interval.Append1=1;
            Interval.Append2=Interval.X2-Interval.X1+1;
            [Interval.Lomb1,Interval.Lomb2]=GetOffsetInterval...
               (Interval.X1,Interval.X2,CircadianInterval,N,12);
            IntervalDesc=' 24-hour (midnight-to-midnight)';
         elseif (IntervalNum>1+NumM2MDays)&(IntervalNum<=NumM2MDays+NumN2NDays+1)
            %Do Analysis N2N, Lomb +/-12 from Noon-to-Noon
            Day=IntervalNum-NumM2MDays-1;
            if Day==NumN2NDays
               LastDayYN=1;
            end
            AppendYN=1;
            RelDay=Day+X.StartDateNum-floor(X.EventDateNum);
            Interval.X1=FirstNoonIndex+(Day-1)*CircadianInterval;
            Interval.X2=FirstNoonIndex+Day*CircadianInterval;
            Interval.Analysis1=Interval.X1;
            Interval.Analysis2=Interval.X2;
            Interval.Append1=1;
            Interval.Append2=Interval.X2-Interval.X1+1;
            [Interval.Lomb1,Interval.Lomb2]=GetOffsetInterval...
               (Interval.X1,Interval.X2,CircadianInterval,N,12);
            IntervalDesc=' 24-hour (noon-to-noon)';
         elseif (IntervalNum>1+NumM2MDays+NumN2NDays)
            %Do Analysis 6pm-to-6am, Lomb +/-12 from Midnight-to-Midnight
            Day=IntervalNum-NumM2MDays-NumN2NDays-1;
            if Day==NumM2MDays
               LastDayYN=1;
            end
              AppendYN=1;
            RelDay=Day+X.StartDateNum-floor(X.EventDateNum);
            Interval.X1=FirstMidnightIndex+(Day-1)*CircadianInterval;
            Interval.X2=FirstMidnightIndex+Day*CircadianInterval;
            [Interval.Analysis1,Interval.Analysis2]=GetOffsetInterval...
               (Interval.X1,Interval.X2,CircadianInterval,N,6);
            Interval.Append1=Interval.X1-Interval.Analysis1+1;
            Interval.Append2=Interval.Append1+Interval.X2-Interval.X1;
            [Interval.Lomb1,Interval.Lomb2]=GetOffsetInterval...
               (Interval.X1,Interval.X2,CircadianInterval,N,12);
            IntervalDesc=' 36-hour (6pm-to-6am)';
         end
      else
         IntervalNum=[];
         Interval.X1=[];
         Interval.X2=[];
         IntervalDesc=[];
         Day=[];
         RelDay=[];
         LastDayYN=[];
         AppendYN=[];
      end
      return
%---------------------------------------------------------------------
function [Period,PeriodDesc]=GetPeriodType2Use(X)
persistent PeriodType
   NumPeriodTypes=3;
   if isempty(PeriodType)
      PeriodType=1;
   else
      PeriodType=PeriodType+1;
   end
   if PeriodType<=NumPeriodTypes
   switch PeriodType
      case 1
         Period='free';
         PeriodDesc=' ';
      case 2
         Period='24hours';
         PeriodDesc=' period set to 24 hours';
      case 3
         Period='Lomb'
         PeriodDesc=' period set to Lomb period';
```

```
      end
   else
      Period=[];
      PeriodType=[];
      PeriodDesc=[];
   end
return
%-----------------------------------------------------------------------
function [OffsetIndex1,OffsetIndex2]=GetOffsetInterval...
   (Interval_X1,Interval_X2,CircadianInterval,N,OffsetHours)

   fOffset=OffsetHours/24;
   OffsetIndex1=Interval_X1-fOffset*CircadianInterval;
   if OffsetIndex1<1
      OffsetIndex1=1;
   end
   OffsetIndex2=Interval_X2+fOffset*CircadianInterval;
   if OffsetIndex2>N
      OffsetIndex2=N;
   end
return
%-----------------------------------------------------------------------
function [Period]=AssignNumericCodeToPeriod(PeriodStr,X,Var,Interval)
   if strcmp(PeriodStr,'free')
      Period=0;
   elseif strcmp(PeriodStr,'24hours')
      Period=24;
   elseif strcmp(PeriodStr,'Lomb')
      Period=LombPeriod(X.Time(Interval.Lomb1:Interval.Lomb2)-X.Time(Interval.Lomb1),...
         Var(Interval.Lomb1:Interval.Lomb2));
   end
return
```

APPENDIX C
READCIRCADDATA.M

```
%-----------------------------------------------------------------------
%ReadCircadData.m                                          3 DEC 1999
%-----------------------------------------------------------------------
function [X]=ReadCircadData
%ReadCircadData.m by Tammy Doherty, USARIEM
%reads circadian data from an Excel file
%assumes the header row containing variable names will be within the first 10 rows and 30
columns

%get file name
[X.PathName,X.FileName,X.FileHandle]=GetFileName;

%decide between the excel file and mat file (if it exists)
[ReadMatFileYN]=DecideBetweenExcelAndMatFile(X.PathName,X.FileHandle);

if ReadMatFileYN==1
   X=ReadMatData(X);
else
   X=ReadExcelData(X);
    %save data in a binary '.mat' file for next time
    Save2MatFile(X);
end

return
%-----------------------------------------------------------------------
function [ReadMatFileYN]=DecideBetweenExcelAndMatFile(PathName,FileHandle)
    MatFileName=strcat(FileHandle,'.mat');
    ExcelFileName=strcat(FileHandle,'.xls');

    DirList=dir(PathName);
    MatFileListNum=0; ExcelFileListNum=0;
    for i=1:length(DirList)
    if strcmp(DirList(i).name,MatFileName)
          MatFileListNum=i;
       elseif strcmp(DirList(i).name,ExcelFileName)
          ExcelFileListNum=i;
       end
    end
   if MatFileListNum>0
      ReadMatFileYN=1;
    else
      ReadMatFileYN=0;
    end

return
%-----------------------------------------------------------------------
function [X]=ReadMatData(X)
    TestDDEConnections(X.FileHandle);
    MatFileName=strcat(X.PathName,X.FileHandle);
    X=load(MatFileName);
return
%-----------------------------------------------------------------------
function [X]=ReadExcelData(X)

   %notify user that data read is starting
   disp('reading data (this could take several minutes) ...');
   disp(' ');

   TestDDEConnections(X.FileHandle);

   %Read BCTM, MiniMitter, and Sleep/Wake data
   X=ReadDeviceData(X,'BCTM','PillTemp','Pill Temp',1,1,1);
   X=ReadDeviceData(X,'Mini',['Twrist  ';'Tother  ';'Activity'],...
```

```
      ['Twrist  ';'Tother  ';'Activity'],[1 1 1],0,length(X.BCTM_Hour));
   if length(X.Mini_Hour)==0
      X.Mini_Hour=X.BCTM_Hour;
   end
   X=ReadDeviceData(X,'Sleep','SleepWake','Sleep/Wake',0,0,length(X.BCTM_Hour));
   if length(X.Sleep_Hour)==0
      X.Sleep_Hour=X.BCTM_Hour;
   end

   %Read General Information (i.e., LH peak, Cycle Start, Data Start and Data Stop)
   X=ReadGenInfo(X);

return
%-------------------------------------------------------------------------
function TestDDEConnections(FileHandle)
   %TestDDEConnection is located in TestDDEConnection.m and is responsible
   %for displaying an error box and stopping program execution if the "topic"
   %is not found (i.e., assigned channel number is <=0)

    %Test the raw data worksheet is there (same name as the file name)
   TestDDEConnection(FileHandle)

    %test whether the 'GenInfo' worksheet is there
   TestDDEConnection('GenInfo')

    %test whether the 'MATLAB Summary Results' worksheet is there
   TestDDEConnection('MATLAB Summary Results')

    %test whether the 'MATLAB Results for Graphs' worksheet is there
   TestDDEConnection('MATLAB Results for Graphs')
return
%-------------------------------------------------------------------------
function CloseDDEConnection(channel);
    ddeterm(channel);
return
%-------------------------------------------------------------------------

function [channel]=OpenDDEConnection(FileHandle);
    channel=InitializeDDE(FileHandle);
return
%-------------------------------------------------------------------------
function [X]=ReadDeviceData(X,Device,FieldName,ColHeading,VarType,RequiredYN,AltN);
    channel=OpenDDEConnection(X.FileHandle);
    if length(Device)>0
      TimeVar=strcat(Device,'_Time');
      DayVar=strcat(Device,'_Day');
      DateVar=strcat(Device,'_Date');
      ElDaysVar=strcat(Device,'_ElDays');
      TimeColHead=strcat(Device,'-Time');
      DayColHead=strcat(Device,'-Day');
      DateColHead=strcat(Device,'-Date');
      ElDaysColHead=strcat(Device,'-ElDays');
    else
      TimeVar='Time';
      DayVar='Day';
      TimeColHead='Time';
      DayColHead='Day';
   end
   eval(strcat('Time=ReadExcelColumn(channel,''',TimeColHead,''',50000,0,0);'))
   if length(Time)==0&RequiredYN==1
      errordlg(strcat('Must Enter Either Time or ',TimeColHead),'ERROR');
      error(strcat('Must Enter Either Time or ',TimeColHead));
   elseif length(Time)==0&RequiredYN==0
      N=AltN;
   elseif length(Time)>0
      [N,dummy]=size(Time);
   end
   eval(strcat('Day=ReadExcelColumn(channel,''',DayColHead,''',N,1,0);'));
   if ~isempty(DateColHead)
       eval(strcat('Date=ReadExcelColumn(channel,''',DateColHead,''',N,0,0);'));
   end
```

```
   if ~isempty(ElDaysColHead)
      eval(strcat('ElDays=ReadExcelColumn(channel,''',ElDaysColHead,''',N,1,0);'));
   end
   if length(ElDays)==0&length(Day)==0&length(Date)==0&RequiredYN==1
      errordlg(strcat('Must Enter Day, ',DayVar,', ',DateVar,', or ',ElDaysVar),'ERROR');
      error(strcat('Must Enter Day, ',DayVar,', ',DateVar,', or ',ElDaysVar));
   end
   [NumFields,dummy]=size(FieldName);
    for i=1:NumFields
      eval(strcat('X.',FieldName(i,:),'=ReadExcelColumn(channel,''',ColHeading(i,:),...
         ''',N,',num2str(VarType(i)),',0);'));
   end
   CloseDDEConnection(channel);
   disp(strcat('Done ',Device,' read...'));

   %convert Time/Date information to 'Device_Datenum' and 'Device_Hour' fields

eval(strcat('X.',Device,'_DateNum=Convert2DateNum(Day,Date,ElDays,Time,X.FileName);'));
    eval(strcat('X.',Device,'_Hour=Convert2Hour(X.',Device,'_DateNum,X.FileName);'));
   disp(strcat('Done ',Device,' Date/Time conversions...'));

   return
%-------------------------------------------------------------------------
function [X]=ReadGenInfo(X)

    [channel]=InitializeDDE('GenInfo');
   X.StartDateNum=GetStartDateNum(X.FileName);
   X.EventDateNum=datenum(ddereq(channel,'r5c2',[1 1]));

   DataStartDateString=(ddereq(channel,'r7c2',[1 1]));
   if double(DataStartDateString)==10 %condition upon reading a blank cell
      X.DataStartDateNum=X.StartDateNum;
   else
      X.DataStartDateNum=datenum(DataStartDateString);
   end

   DataStopDateString=(ddereq(channel,'r8c2',[1 1]));
   if double(DataStopDateString)==10  %condition upon reading a blank cell
      %use the date information from the last entry from the BCTM data
       X.DataStopDateNum=floor(X.BCTM_DateNum(length(X.BCTM_DateNum)));
   else
      X.DataStopDateNum=datenum(DataStopDateString);
   end

   ddeterm(channel);
   return
%-------------------------------------------------------------------------


function [PathName,FileName,FileHandle]=GetFileName

%get the current directory name
working_directory=cd;

%set current directory to Circadian Data Path (from CircadianDataPath.m)
cd(CircadianDataPath);

%get data file name
[FileName,PathName]=uigetfile('*.xls','select data file');
FileHandle=GetFileHandle(FileName);

%change back to original working directory
cd(working_directory);

return
%-------------------------------------------------------------------------
function [Row,Col]=GetStartRowCol(channel,Vname)
%Assumes the header row containing variable names will be within the first 10 rows
%and 30 columns
Row=0; Col=0;
maxrows=10; maxcols=30;
```

```
%find the row and column numbers of the recognized variable names
row=1;FoundMatch=0;
while row<=maxrows&~FoundMatch
   col=1;
   while col<=maxcols&~FoundMatch
      x=ddereq(channel,strcat('r',num2str(row),'c',num2str(col)),[1 1]);
      x=x(1:length(x)-1);
      if strcmp(upper(x),upper(Vname))==1
            Row=row; Col=col;
            FoundMatch=1;
      end
      col=col+1;
   end
    row=row+1;
end
return

%-----------------------------------------------------------------------
function [X]=ReadExcelColumn(channel,Vname,N,NumericYN,checkN)
    %string format (sformat) for reading from Excel
   %Indx=VnameIndex(Vname);
   [startrow,col]=GetStartRowCol(channel,Vname);
   if (startrow~=0)&(col~=0)
       lastrow=startrow+N;
       Xcellarray=ddereq(channel,strcat('r',num2str(startrow+1),'c',num2str(col),...
          ':r',num2str(lastrow),'c',num2str(col)),[1 1]);
        Xstringarray=char(Xcellarray);
        i=findstr(Xstringarray,char(10));
        N=length(i);
        i1=i(1:N-1)+1;
        i1=[1 i1];
        i2=i-1;
       if NumericYN
          X(1:N)=NaN;  %initialize to NaN in case of missing values
           X=X';
       end
       for z=1:N
          if i1(z)<=i2(z)
             Xstringmatrix(z,1:(i2(z)-i1(z)+1))=Xstringarray(i1(z):i2(z));
              if NumericYN&length(Xstringmatrix(z,:))>0
               if length(str2num(Xstringmatrix(z,:)))>0
                  X(z)=str2num(Xstringmatrix(z,:));
               end
             end
           end
       end
       if ~NumericYN
         if isempty(Xstringmatrix)
            X=[];
            N=0;
         else
            X=Xstringmatrix;
         end
       end
   else  %case if the variable is not present in the file (row=0, col=0)
      X=[];
   end
   %pad end of array to get N elements
   if checkN==1;
       [rows,cols]=size(X);
        if rows>0
          for row=rows+1:N
             if NumericYN
                X(row,:)=NaN;
             else
                X(row,:)={''};
             end
           end
      end
   end
return
```

```
%-------------------------------------------------------------------------
function [DateNum]=Convert2DateNum(Day,Date,ElapsedDays,Time,fname)
    [Subject,MPhase,FileNum,MCycle,Julian,StartYear]=DecodeCircadFileName(fname);
   [StartMonth,StartDay]=DecodeJulian(Julian,StartYear);
   if length(Date)>1   %date entered as string
      DateNum=datenum(Date)+datenum(Time);
   else
      if length(Day)>1
         N=length(Day);
         Year(1:N,1)=StartYear;
         Month(1:N,1)=StartMonth;
         iMonthChange=find((datenum(Year(2:N),Month(2:N),Day(2:N))-datenum(Year(1:N-
1),...
            Month(1:N-1),Day(1:N-1)))<0);
         while length(iMonthChange)>0
            NewYear=Year(iMonthChange(1));
            NewMonth=Month(iMonthChange(1))+1;
            if NewMonth>12
               NewMonth=1;
               NewYear=NewYear+1;
            end
            Month(iMonthChange(1)+1:N)=NewMonth;
            Year(iMonthChange(1)+1:N)=NewYear;
            iMonthChange=find((datenum(Year(2:N),Month(2:N),Day(2:N))-datenum(Year(1:N-
1),...
               Month(1:N-1),Day(1:N-1)))<0);
         end
         DateNum=datenum(Year,Month,Day)+datenum(Time);
      elseif length(ElapsedDays)>1
         DateNum=datenum(StartYear,StartMonth,StartDay)+ElapsedDays+datenum(Time);
      else
         DateNum=[];
      end
   end
return

%-------------------------------------------------------------------------
function [Hour]=Convert2Hour(DateNum,fname)
    if isempty(DateNum)
    Hour=[];
    else
       StartDateNum=GetStartDateNum(fname);
       Hour=(DateNum-StartDateNum)*24;
    end
   return
%-------------------------------------------------------------------------
function [StartDateNum]=GetStartDateNum(fname)
        [Subject,MPhase,FileNum,MCycle,Julian,StartYear]=DecodeCircadFileName(fname);
       [StartMonth,StartDay]=DecodeJulian(Julian,StartYear);
       StartDateNum=datenum(StartYear,StartMonth,StartDay);
return
%-------------------------------------------------------------------------

function Save2MatFile(X)
    MatFileName=strcat(X.PathName,X.FileHandle);
    FieldName=fieldnames(X);
    FileDate=datestr(now,0);
    save(MatFileName,'FileDate');
    for i=1:length(FieldName)
       eval(strcat(char(FieldName(i)),'=X.',char(FieldName(i)),';'));
       save(MatFileName,char(FieldName(i)),'-append');
    end
return
```

APPENDIX D
MESHDATA.M

```
%-----------------------------------------------------------------------
%MeshData.m                                                  3 DEC 1999
%-----------------------------------------------------------------------
function [Y]=MeshData(X);
    disp('meshing variables into a single time base ...');

   %copy non-meshed fields to Y
   Y=CopyNonMeshedFields2Y(X);

   %Get data start and stop times.  Start times are either the first time encountered
   %in any of the device date/time fields, or the data start time as entered into the
   %GenInfo worksheet.  Stop times are either the latest time encountered
   %in any of the device date/time fields, or the data stop time as entered into the
   %GenInfo worksheet.
   [Y.MinHour,Y.MaxHour]=GetDataStartStopTimes(X);

   %Compute new Hour array
   Y.dTime=1/60;  %interval is 1-minute
   Y.Time=[Y.MinHour:Y.dTime:Y.MaxHour]';

   Y=DoTheMapping(X,Y);

   %send results to 'MATLAB Results for Graphs' worksheet
   SendMeshedResultsToExcel(Y);

return
%-----------------------------------------------------------------------
function [Y]=CopyNonMeshedFields2Y(X)
   MeshedFields='_DateNum,_Hour,PillTemp,Twrist,Tother,Activity,SleepWake';
   FieldName=fieldnames(X);
   for i=1:length(FieldName)
      if isempty(findstr(MeshedFields,char(FieldName(i))))...
            &isempty(findstr(char(FieldName(i)),'_Hour'))&...
            isempty(findstr(char(FieldName(i)),'_DateNum'))
       eval(strcat('Y.',char(FieldName(i)),'=X.',char(FieldName(i)),';'));
       end
   end
return
%-----------------------------------------------------------------------
function [MinHour,MaxHour]=GetDataStartStopTimes(X)
    StartHour=(X.DataStartDateNum-floor(X.DataStartDateNum))*24;
    StopHour=(X.DataStopDateNum-floor(X.DataStartDateNum))*24;
   %get minimum of data time points
   MinHour=X.BCTM_Hour(1);
   if ~isempty(X.Mini_Hour)
      MinHour=min(MinHour,X.Mini_Hour(1));
   end
   if ~isempty(X.Sleep_Hour)
      MinHour=min(MinHour,X.Sleep_Hour(1));
   end
   %get maximum of StartHour and MinHour
   MinHour=max(MinHour,StartHour)
   %get maximum of data time points
   MaxHour=X.BCTM_Hour(length(X.BCTM_Hour));
   if ~isempty(X.Mini_Hour)
      MaxHour=max(MaxHour,X.Mini_Hour(length(X.Mini_Hour)));
   end
   if ~isempty(X.Sleep_Hour)
      MaxHour=max(MaxHour,X.Sleep_Hour(length(X.Sleep_Hour)));
   end
   MaxHour=min(StopHour,MaxHour);
   %round MaxHour and MinHour to nearest whole minute
   MaxHour=round(MaxHour*60)/60;
   MinHour=round(MinHour*60)/60;
```

```
   return
%-----------------------------------------------------------------------
function [iStart,iStop]=GetStartStopElement(Device_Hour,StartHour,StopHour)
   i=find(Device_Hour<StartHour);
   if isempty(i)
      iStart=1;
   else
      iStart=max(i)+1;
      if iStart>length(Device_Hour)
         iStart=length(Device_Hour);
      end
   end
   i=find(Device_Hour>StopHour);
   if isempty(i)
      iStop=length(Device_Hour);
   else
      iStop=min(i)-1;
      if iStop<iStart
         iStop=iStart;
      end
   end
return
%-----------------------------------------------------------------------
function [X]=Round2NearestMinute(X);
    X.BCTM_RoundHour=(X.BCTM_Hour.*60)/60
   if isempty(X.Mini_Hour)
      X.Mini_RoundHour=[];
   else
      X.Mini_RoundHour=(X.Mini_Hour.*60)/60
   end
   if isempty(X.Sleep_Hour)
      X.Sleep_RoundHour=[];
   else
      X.Sleep_RoundHour=(X.Sleep_Hour.*60)/60
   end
return

%-----------------------------------------------------------------------
function [Y]=DoTheMapping(X,Y)
   %Compute the indices for mapping the old arrays into the new arrays
   BCTM.i=round((round(X.BCTM_Hour./Y.dTime).*Y.dTime - Y.MinHour)./Y.dTime)+1;
   Mini.i=round((round(X.Mini_Hour./Y.dTime).*Y.dTime - Y.MinHour)./Y.dTime)+1;
   Sleep.i=round((round(X.Sleep_Hour./Y.dTime).*Y.dTime - Y.MinHour)./Y.dTime)+1;

   %Do the mapping
   N=length(Y.Time);
   %BCTM
   Y.Traw=Map(X.PillTemp,BCTM.i,N,1);
   %MiniMitter
   Y.Twrist=Map(X.Twrist,Mini.i,N,1);
   Y.Tother=Map(X.Tother,Mini.i,N,1);
   Y.Activity=Map(X.Activity,Mini.i,N,1);
   %Sleep/Wake
   Y.SleepWake=Map(X.SleepWake,Sleep.i,N,0);
return
%-----------------------------------------------------------------------
function [DataMapped]=Map(Data,iData,N,NumericYN)
    if length(Data)>0
        iGoodData=find((iData>0)&(iData<=N));
       Data=Data(iGoodData);
    iData=iData(iGoodData);
        if NumericYN==1
       DataMapped(1:N)=NaN;
       end
      DataMapped(iData)=Data; DataMapped=DataMapped';
      if length(DataMapped)<N
         if NumericYN==1
            DataMapped(N)=NaN;
         else
            DataMapped(N)=Data(length(Data));
         end
```

```
      end
   else
      DataMapped=[];
   end
return

%-----------------------------------------------------------------------
function SendMeshedResultsToExcel(X)

   %poke time and descriptive variables (pill#, event#,...)
      PokeTimeBasedColumn('Time',X.Time);
      PokeTimeBasedColumn('Traw',X.Traw);
      PokeTimeBasedColumn('Activity',X.Activity);
      PokeTimeBasedColumn('Twrist',X.Twrist);
      PokeTimeBasedColumn('Tother',X.Tother);
return
```

APPENDIX E
DATA TRANSFORMATION FUNCTIONS

```
%-----------------------------------------------------------------------
%RemoveOutliers.m                                          10 AUG 1999
%-----------------------------------------------------------------------
function [T]=RemoveOutliers(T)
    %set outliers to NaN
   %note that index will also identify string values
   %which are assigned numeric values of zero
   index=find((T>40)|(T<35));

   %Assign any elements of T that are in index the value NaN
   T(index)=NaN;

   %write results to Excel file
   PokeTimeBasedColumn('Tcleaned',T);

return


%-----------------------------------------------------------------------
%TransformSleepWake.m                                       11 AUG 1999
%-----------------------------------------------------------------------
function [X]=TransformSleepWake(X)
%transforms SleepWake from a string array to a numeric array
%where "sleep" is given a value of 38.2, "wake" is given a value of zero

%First, look to see whether Sleep/Wake was already converted
NotConverted=isempty(find(X.SleepWake==38.2));
if NotConverted
    x=double(upper(X.SleepWake));  %converts string characters to ascii code
    index=find(x==0);
   if length(index)>0
       if index(1)==1
          x(1)=87;  %if first entry is blank, set to "awake"
       end
       %for any blank values, drag down the previous value
       for j=2:length(index)
          x(index(j))=x(index(j)-1);
       end
   end
   %convert all code 83's to 38.2 and 87's to zero
   IsAsleep=(x==83);
   x=IsAsleep.*38.2;  %this step automatically sets non-83's to zero
   PokeTimeBasedColumn('Sleep',x);
end
return
```

APPENDIX F
FFT_FILTERING.M

```
%FFT_Filtering.m                                              11 AUG 1999
%--------------------------------------------------------------------------
%--------------------------------------------------------------------------
function [X]=FFT_Filtering(X)
%fill in blank temperature values so that the FFT will have
%a temperature array with a constant sampling interval
%Reminder: Tclean is the meshed pill temperature vector with text and outliers removed
    X.T=FillTemperatureBlanks(X.Tclean);

    N=length(X.T);

   %N needs to be an even number
   if (N/2-floor(N/2))>0
      N=floor(N/2)*2;
   end

   FFTofT=fft(X.T(1:N));

   %for the periodogram:
   freq=((1/X.dTime).*(0:N/2)./N)';
    period(2:floor(N/2))=(1./freq(2:floor(N/2)))';
    magnitude=abs(FFTofT);

    for f_cutoff=[.25 .5 1.0]
        %set to zero everything with freq>f_cutoff
        i_cutoff=ceil(f_cutoff/((1/X.dTime))*N);
        FFTfilt=FFTofT;
        FFTfilt(i_cutoff+1:N/2)=0;
        FFTfilt(N/2+2:N-i_cutoff)=0;

        %perform inverse fft to get filtered curve
      VarName=strcat('X.Tfilt',num2str(round(1/f_cutoff)));
      Var=abs(ifft(FFTfilt));
      if length(Var)<length(X.T)
         Var(N+1:length(X.T))=X.T(N+1:length(X.T));
      end
      %assign the filtered curve to X.Tfilt1, X.Tfilt2, or X.Tfilt4
      eval(strcat(VarName,'=Var;'));
   end


   %plot results
   FFT_plot(freq,period,magnitude,X);

   %send filtered data to 'MATLAB Results for Graphs'
   PokeTimeBasedColumn('Tfilt1',X.Tfilt1);
   PokeTimeBasedColumn('Tfilt2',X.Tfilt2);
   PokeTimeBasedColumn('Tfilt4',X.Tfilt4);

return

%--------------------------------------------------------------
function FFT_plot(freq,period,magnitude,X)
    figure(1);
   Np=length(period);

    %plot raw data
    subplot(5,1,1), plot(X.Time,X.T),ylabel('raw'),xlabel('time')
    title(X.FileHandle)
   %plot periodogram
    subplot(5,1,2),plot(period(2:Np),magnitude(2:Np)),ylabel('magnitude'),xlabel('period')
   %plot filtered data
   subplot(5,1,3),plot(X.Time,X.Tfilt1),ylabel('filtered'),xlabel('time')
```

```
    Title=strcat(X.FileHandle,':  period cutoff = 1hr');
   subplot(5,1,4),plot(X.Time,X.Tfilt2),ylabel('filtered'),xlabel('time')
    Title=strcat(X.FileHandle,':  period cutoff = 2hr');
   subplot(5,1,5),plot(X.Time,X.Tfilt4),ylabel('filtered'),xlabel('time')
   Title=strcat(X.FileHandle,':  period cutoff = 4hr');
return

%--------------------------------------------------------------------
function [T]=FillTemperatureBlanks(T);
NotANumber=isnan(T);
IsZero=(T==0);
IsBlank=NotANumber+IsZero;
index=find(IsBlank>0);
for j=1:length(index)
   if index(j)~=1
      T(index(j))=T(index(j)-1);
   end
end
%to estimate values for the beginning of the array, go backwards from 1st good value
NotANumber=isnan(T);
IsZero=(T==0);
IsBlank=NotANumber+IsZero;
index=find(IsBlank>0);
for j=length(index):-1:1
   T(index(j))=T(index(j)+1);
end
PokeTimeBasedColumn('Tfilled',T)
return
```

APPENDIX G
FUNCTIONS TO GET THE LOMB PERIOD

```
%-----------------------------------------------------------------------
%LombPeriod.m                                                  3 DEC 1999
%-----------------------------------------------------------------------
%Name changed to LombPeriod from LombAnalysis on 1 SEP 1999

function [Period]=LombPeriod(t,T)
    sig=0.001;
    disp('starting Lomb analysis ...');
    disp(' ');

   %set up axes
    [xticks,xticklabels]=XAxesSetup(t);

    %Run Lomb on Whole Curve
    %The lomb function used here was written by John Shimeld, June 1997
    %obtained at www.swarthmore.edu/epigone/comp.soft_sys.matlab/shermgermsku/
    %Pine.HPP.3.96.971118082015.3056A-100000@agc2
   [fLomb,PerLomb,PowLomb,Psig]=lomb(t,T,[.5 .05 .01 .001]);

   %plot results
   goplot(PerLomb,PowLomb,'period (hours)','Lomb Power','b',2,1,1,1,xticks,xticklabels);
   goplot([0 max(PerLomb)],[Psig(1) Psig(1)],'period (hours)',...
      'Lomb Power','y-',2,1,1,1,xticks,xticklabels);
   goplot([0 max(PerLomb)],[Psig(2) Psig(2)],'period (hours)',...
      'Lomb Power','c-',2,1,1,1,xticks,xticklabels);
   goplot([0 max(PerLomb)],[Psig(3) Psig(3)],'period (hours)',...
      'Lomb Power','g-',2,1,1,1,xticks,xticklabels);
   goplot([0 max(PerLomb)],[Psig(4) Psig(4)],'period (hours)',...
      'Lomb Power','r-',2,1,1,1,xticks,xticklabels);
    legend('Lomb Power','P<0.5','P<0.05','P<0.01','P<0.001');
   [MaxVal,MaxIndx]=max(PowLomb);
   Period=PerLomb(MaxIndx);
    disp('DONE LOMB!');
   return
%-----------------------------------------------------------------------

 function goplot(xdata,ydata,xlabel,ylabel,plotchar,fignum,numplothoriz,...
    numplotvert, plotnum,xticks,xticklabels);
    figure(fignum);
   grid on;
   hold on;
   subplot(numplotvert,numplothoriz,plotnum),plot(xdata,ydata,plotchar), XLABEL(xlabel),
YLABEL(ylabel);
    hold off;
   set(gca,'XTick',xticks);
   set(gca,'XTickLabel',xticklabels);
   return
%-------------------------------------------------------------------------------
function [xticks,xticklabels]=XAxesSetup(t)
maxtime=(floor(max(t)/24)+(mod(max(t),24)>0))*24;
xticks=[0:2:maxtime];
xticklabels=int2str(0);
for tick=6:6:maxtime
   xticklabels=strcat(xticklabels,'|||');
   xticklabels=strcat(xticklabels,int2str(tick));
end
return


%-----------------------------------------------------------------------
%Lomb.m                                                         3 DEC 1999
%-----------------------------------------------------------------------
% LOMB.M
```

```
%       Spectral Analysis of Unevenly Sampled Data
%
%   Implements the Lomb algorithm (cf., Numerical Recipes, 2nd ed.,
% page 575) to estimate the spectral power of an unevenly
% sampled, 2D dataset.
%
% INPUT PARAMETERS:
%   t   - sample times   [nx1]
%   Z   - measured value at each sample time   [nx1]
%   sig - significance levels for which a power estimate is
%         desired (e.g., [0.5 0.05] would give power values
%         for the 50% and 95% confidence limits).
%
% OUTPUT PARAMETERS:
%   F   - frequencies for the Lomb power estimates
%   P   - power estimates
%   Psig - powers at the desired confidence limits
%
% NOTE: this is not a particularly fast implementation of the
%       Lomb periodogram. See Numerical Recipes for a faster
%       algorithm.
%
%  John Shimeld, June 1997.
% ------------------------------------------------------------------
function [freq,period,power,Psig]=lomb(t,Z,sig)
%remove missing values from Z (modification 3 DEC 99 by TJD)
i=find(~isnan(Z));
t=t(i);
Z=Z(i);

ofac = 4;     % oversampling factor, should be >= 4
hifac = 1/12;  %2;   % determines the highest frequency for which
              % the powers are estimated. This corresponds
              % to hifac*(Nyquist frequency if dataset were evenly spaced)

N = length(t);
Zmean = mean(Z);
Zvar = cov(Z);

t_span = max(t)-min(t);
np = ofac*hifac*N/2;
fc = N/2/t_span;
fmin = 1/t_span;
fmax = hifac*fc;
fstep = (fmax-fmin)/np;
nf = (fmax-fmin)/fstep+1;

freq = [fmin:fstep:fmax];
pstep=(max(t)-min(t))/np;
period=min(t):pstep:max(t);

for z=1:np+1
   %This modification added 8/10/99 by TJD to avoid div-by-zero error in omega assignment
statement
   if period(z)==0
      period(z)=0.0000000001;
   end
    omega = 2.*pi/period(z);
    tau = atan(sum(sin(2.*omega.*t)) ./ ...
   sum(cos(2.*omega.*t)))./2./omega;
  a = cos(omega.*(t-tau));
  b = sin(omega.*(t-tau));
  power(z) = (sum((Z-Zmean).*a).^2./sum(a.^2) + sum((Z-
Zmean).*b).^2./sum(b.^2))./(2*Zvar);
end
Psig = log(hifac.*np./ofac./sig);
return
```

APPENDIX H
DOCOSINOR.M

```
%-----------------------------------------------------------------------
%DoCosinor.m                                                  4 OCT 1999
%-----------------------------------------------------------------------
function [cosinor]=DoCosinor(t,T,Period)
    %need to remove missing values
   i=find(~isnan(T));
   T2=T(i);
   t2=t(i);

   if Period==0
      p0=[40 .5 1 .25];
      p=fmins('sumsqerror',p0,[0],[],T2,t2);
    elseif Period==24
       p0=[40 .5 1];
      p=fmins('sumsqerrorFP',p0,[0],[],T2,t2,Period);
    p(4)=2*pi/Period;
   else %Period==LombPeriod
      p0=[40 .5 1];
      p=fmins('sumsqerrorFP',p0,[0],[],T2,t2,Period);
       p(4)=2*pi/Period;
   end
   cosinor.Tpred=(CosinorTpred(p,t)); %these use original time array
    cosinor.t=t;
   cosinor.N=length(t2);  %this is the number of ~isnan in T
   cosinor.mesor=p(1);
   cosinor.amplitude=p(2);
   cosinor.period=2*pi/p(4);
   cosinor.phase=-p(3)*cosinor.period/2/pi;  %for cosine cosinor
   FirstMidnightAfterTime1=24;
   while FirstMidnightAfterTime1<t(1)
      FirstMidnightAfterTime1=FirstMidnightAfterTime1+24;
   end
   cosinor.gphase=cosinor.phase;
   while cosinor.gphase<FirstMidnightAfterTime1
      cosinor.gphase=cosinor.gphase+cosinor.period;
   end
   cosinor.clock_phase=cosinor.gphase;
   while cosinor.clock_phase>24
    cosinor.clock_phase=cosinor.clock_phase-24;
   end
   cosinor.phase_clockstring=datestr(cosinor.clock_phase/24,13);

   %correlation statistics
   Cosinor_Tpred2=CosinorTpred(p,t2);
   R=corrcoef(T2,Cosinor_Tpred2);
   cosinor.R=R(2);  %R is a correlation matrix. 2nd coefficient is corr coeff

   %max and min info
   [cosinor.MaxData,iMaxData]=max(T2);
   [cosinor.MinData,iMinData]=min(T2);
   [cosinor.MaxPred,iMaxPred]=max(Cosinor_Tpred2);
   [cosinor.MinPred,iMinPred]=min(Cosinor_Tpred2);
   cosinor.TimeOfMaxData=t2(iMaxData);
   while cosinor.TimeOfMaxData>24
    cosinor.TimeOfMaxData=cosinor.TimeOfMaxData-24;
   end
   cosinor.TimeOfMinData=t2(iMinData);
   while cosinor.TimeOfMinData>24
    cosinor.TimeOfMinData=cosinor.TimeOfMinData-24;
   end
   cosinor.TimeOfMaxPred=t2(iMaxPred);
   while cosinor.TimeOfMaxPred>24
    cosinor.TimeOfMaxPred=cosinor.TimeOfMaxPred-24;
```

```
end
cosinor.TimeOfMinPred=t2(iMinPred);
while cosinor.TimeOfMinPred>24
 cosinor.TimeOfMinPred=cosinor.TimeOfMinPred-24;
end

return
```

APPENDIX I
UTILITY FUNCTIONS

```
%-----------------------------------------------------------------------
%CircadianDataPath.m
%-----------------------------------------------------------------------
function [Path]=CircadianDataPath
    Path='j:\ObservedData\Circadian\H99-11\Working Data\';
return


%-----------------------------------------------------------------------
%CosinorTpred.m                                          17 SEP 1999
%-----------------------------------------------------------------------
function Tpred=CosinorTpred(p,t)
    Tpred=p(1)+p(2)*cos(p(4).*t+p(3));
return


%-----------------------------------------------------------------------
%DecodeCircadFileName.m
%-----------------------------------------------------------------------
function [Subject,MPhase,FileNum,MCycle,Julian,Year]=DecodeCircadFileName(CircadFileName)
   fHandle=CircadFileName(1:findstr('.',CircadFileName)-1);
   Subject=fHandle(2:3);
   MPhase=fHandle(4);
   FileNum=str2num(fHandle(5));
   MCycle=fHandle(6);
   Julian=str2num(fHandle(7:9));
   Year=str2num(fHandle(11:14));
return


%-----------------------------------------------------------------------
%DecodeJulian.m
%-----------------------------------------------------------------------
function [M,D]=DecodeJulian(Julian,Year)
    cumEOMDay=cumsum(eomday(Year,1:12));
   M=min(find(cumEOMDay>=Julian));
    cumEOMDay=[0 cumEOMDay];
   D=Julian-cumEOMDay(M);
return


%-----------------------------------------------------------------------
%GetFileHandle.m
%-----------------------------------------------------------------------
function [fHandle]=GetFileHandle(fname)
    fHandle=fname(1:findstr('.',fname)-1);
return


%-----------------------------------------------------------------------
%InitializeDDE.m
%-----------------------------------------------------------------------
function [channel]=InitializeDDE(topic)
    %initialize the file for DDE
    channel=ddeinit('excel',topic);
    %repeat the initialization step (bug in Matlab 5.3)
    channel=ddeinit('excel',topic);
return


%-----------------------------------------------------------------------
%PokeTimeBasedColumn.m                                   11 AUG 1999
```

```
%-----------------------------------------------------------------------
function PokeTimeBasedColumn(x_heading,x)
persistent ColNum
    row=1;
   if isempty(ColNum)
      ColNum=1;
   end
   N=length(x);
   if N>0
        [channel]=InitializeDDE('MATLAB Results for Graphs');
      Cell=strcat('r',num2str(row),'c',num2str(ColNum));
       ddepoke(channel,Cell,x_heading);
      x=ConvertNaNs(x);

Cell=strcat('r',num2str(row+1),'c',num2str(ColNum),':r',num2str(N+1),'c',num2str(ColNum))
;
      ddepoke(channel,Cell,x);
       ColNum=ColNum+1;
       ddeterm(channel);
   end
return
%-----------------------------------------------------------------------
function [x]=ConvertNaNs(x)
    MissingValueCode=0;
    IsNotANumber=isnan(x);
   index=find(IsNotANumber>0);
   x(index)=MissingValueCode;
   return


%-----------------------------------------------------------------------
%sumsqerror.m                                          17 SEP 1999
%-----------------------------------------------------------------------
function sumEsq=sumsqerror(p,T,t)
   sumEsq=sum((T-CosinorTpred(p,t)).^2);
return


%-----------------------------------------------------------------------
%sumsqerrorFP.m                                        17 SEP 1999
%-----------------------------------------------------------------------
function sumEsq=sumsqerrorFP(p,T,t,Period)
    p(4)=2*pi/Period;
    sumEsq=sum((T-CosinorTpred(p,t)).^2);
return


%-----------------------------------------------------------------------
%TestDDEConnection.m                                   11 AUG 1999
%-----------------------------------------------------------------------
function TestDDEConnection(Worksheet)
   %Test that Excel file is open and active
   %InitializeDDE is an external function named InitializeDDE.m
    [channel]=InitializeDDE(Worksheet);
    if channel==0
       errordlg(strcat('Excel worksheet ''',Worksheet,''' must be open and
active'),'ERROR');
       error(strcat('Excel worksheet ''',Worksheet,''' must be open and active'));
    end
   ddeterm(channel);
return


%-----------------------------------------------------------------------
%WriteSummaryResultsByMethod.m                         1 OCT 1999
%-----------------------------------------------------------------------
function WriteSummaryResultsByMethod(FitData)
persistent RowNum
    %constants
    WorkSheet='MATLAB Summary Results';
   AHeading={'mesor'; 'amplitude'; 'period'; 'gphase'; 'clock_phase';...
```

```
        'phase_clockstring'; 'R'; 'N'; 'MaxData'; 'TimeOfMaxData';...
        'MinData'; 'TimeOfMinData'; 'MaxPred'; 'TimeOfMaxPred'; 'MinPred';
'TimeOfMinPred'};
   NumericYN=[1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1];

    %get row number to start at
    if isempty(RowNum)
      RowNum=1;
   end
   startcol=1;

   %open the DDE connection to the excel worksheet
    [channel]=InitializeDDE(WorkSheet);

   ndays=length(FitData.R);

   ddepoke(channel,strcat('r',num2str(RowNum),'c',num2str(startcol)),FitData.Title);
   col=startcol;
   if ndays>1
      PokeSummaryColumn(channel,RowNum+1,col,ndays,'Day (rel to Event)',...
         FitData.DayRelative2Event,1);col=col+1;
   else
      col=col+1;
   end

   [NumHeadings,dummy]=size(AHeading);
   for z=1:NumHeadings
      Heading=char(AHeading(z));
      Data=eval(strcat('FitData.',Heading));
      PokeSummaryColumn(channel,RowNum+1,col,ndays,Heading,Data,NumericYN(z));
      col=col+1;
   end

   %terminate DDE connection
   ddeterm(channel);

   %update starting row number
   RowNum=RowNum+ndays+3;

return
%-----------------------------------------------------------------------
function PokeSummaryColumn(channel,row,col,ndays,x_heading,x,NumericYN)
   ddepoke(channel,strcat('r',num2str(row),'c',num2str(col)),x_heading);
    if NumericYN==0
      for i=1:ndays
          ddepoke(channel,strcat('r',num2str(row+i),'c',num2str(col)),x(i,:));
      end
   else
      ddepoke(channel,strcat('r',num2str(row+1),'c',num2str(col),...
         ':r',num2str(row+ndays),'c',num2str(col)),x);
   end


return
```