(2)

WHOI-90-45

Woods Hole Oceanographic Institution

Altimeter Processing Tools for Analyzing Mesoscale Ocean Features

by

Michael J. Caruso, Ziv Sirkes, Pierre J. Flament, and M.K. Baker

September 1990

Technical Report

Funding was provided by the Office of Naval Research through Contract No. N00014-86-K-0751.

WHOI-90-45

Altimeter Processing Tools for Analyzing Mesoscale Ocean Features

by

Michael J. Caruso
Ziv Sirkes*
Woods Hole Oceanographic Institution
Woods Hole, Massachusetts 02543

Pierre J. Flament
M.K. Baker
Oceanography Department
University of Hawaii
Honolulu, HI

September 1990

Technical Report

Funding was provided by the Office of Naval Research through Grant No. N00014-86-K-0751.

Approved for Distribution:

James R. Luyten, Chairman
Department of Physical Oceanography

*Present address: Institute for Naval Oceanography, Stennis Space Center, MS

Abstract

Satellite altimeters provide many opportunities for oceanographers to supplement their research with a valuable new data set. The recent GEOSAT exact repeat mission is the first of several altimetry missions proposed during the next decade. To utilize this new data, a software package was developed at the Woods Hole Oceanographic Institution and the University of Hawaii to facilitate the extraction of useful information from the NODC distributed GEOSAT data tapes. This software package was written with portability and modularity in mind. It should be possible to use this package with little or no modifications on data from future altimeters. The code was written in C and tested on Sun workstations and is oriented toward UNIX operating systems. However, since standard code was used, the programs should port easily to other computer systems. The modularity of the code should enable users to create addition programs. Additional programs designed to handle collocated water vapor corrections are also included for comparison.

Altimeter Processing Tools for Analyzing Mesoscale Ocean Features

Michael J. Caruso
Ziv Sirkes*
Woods Hole Oceanographic Institution
Woods Hole, MA

Pierre J. Flament
M. K. Baker
Oceanography Department
University of Hawaii
Honolulu, HI

September 20, 1990

*Present address: Institute for Naval Oceanography, Stennis Space Center, MS

Contents

List of Figures

List of Tables

1 Introduction

The altimeter is an active microwave radar that measures the distance between itself and the ocean surface. A pulse of known power and duration is directed toward the sea surface. By measuring the power of the return pulse, it is possible to determine the altimeter height. By fitting the shape of the return pulse, it is possible to calculate the significant wave height and the near-surface wind speed. The uses of satellite altimetry include the determination of ocean currents, measurement of significant wave height and ocean tides as well as estimation of surface wind speeds.

The U.S. Navy altimeter satellite GEOSAT (GEOdetic SATellite) was designed to provide the U.S. military with a highly improved marine geoid. In October 1986, when the satellite had completed this classified work, it was moved into a 17-day exact repeat orbit. The new orbital parameters corresponded to the 1978 Seasat mission. This new unclassified orbit was corrected periodically to provide a groundtrack repeatability to within 1 km.

The primary purpose of this project is to perform a "repeat" or "collinear" track analysis. This analysis requires sorting the data into collinear tracks, correcting the sea surface heights for various measurement errors and regridding the along-track data to a common grid. We developed generalized programs to read and assimilate the data into a usable data set. These programs were developed on a Sun Workstation[1], but could be easily ported to other computer systems.

Since GEOSAT does not have an onboard sensor to measure the effects of water vapor, two separate estimated water vapor corrections are supplied with the data. One alternative used here is the first Special Sensor Microwave/Imager (SSMI), launched in June 1987 aboard a Defense Meteorological Satellite Program spacecraft. The SSMI senses brightness temperatures. From those brightness temperatures environmental parameters such as wind speed and water vapor can be derived (Hollinger et al. 1987).

We decided to write several simple programs to read in the binary data tapes, format the data and write out the ASCII equivalents. We also worked out a naming convention to facilitate the storage and retrieval of individual subtracks. Programs were also written for repeat track analysis and to interpolate the data to a uniform latitude/longitude grid. These programs were designed with mesoscale motions in mind. However, since the programs are modular, users can easily use their own orbit and geoid corrections to study basin scale problems. We left programs that interpret the data for implementation by the individual users.

Section 2 describes the GEOSAT geophysical data record (GDR) and section 3 describes the SSMI data record. Section 4 illustrates the approach to handling the expansive data set and section 5 describes the programs and subroutines developed to handle the GEOSAT data along with explanations of input and output data. Section 6 describes the use of these programs to perform a repeat track analysis of a section of the North Atlantic from 22° N to 48° N and 284° E to 316° E. The appendices contain UNIX[2]-style manual pages, program and subroutine listings and UNIX shell scripts described throughout the text.

[1]Sun Workstation is a registered trademark of Sun Microsystems, Inc.
[2]UNIX is a trademark of AT&T Bell Laboratories

2 Geophysical Data Record GDR

The raw altimeter data are collected at the Johns Hopkins University Applied Physics Laboratory (JHU/APL) and are processed by the National Oceanographic and Atmospheric Administration (NOAA). The data are merged with ephemerides and corrections are added for tides and refractions (Cheney et al. 1987.) The National Ocean Data Center (NODC) in Washington, D.C. distributes the user handbook(Cheney et al. 1987) and the completed GDR which is available on tape.

Table 1 shows the parameters contained in each GDR. The parameter column contains the names used in the user handbook and the abbreviation column contains the names used for each parameter in the programs and in the text.

The first 5 items are stored as 4-byte integers. Parameters *utc* and *utcm* contain the time of the record since the 00:00 UTC, 1 Jan. 1985. The time of the record may be calculated by $t = utc + utcm * 10^{-6}$. The parameters *Lat* and *Lon* contain the latitude and longitude in microdegrees. A positive latitude is north of the equator and the longitude is measured east of the Greenwich meridian. The satellite orbit height, *Orb*, is given in mm above the reference ellipsoid.

The next 29 parameters are stored as 2-byte integers. The first of these parameters, *m_h*, is the average sea surface height of the record given in cm above the ellipsoid. The standard deviation of the heights used to calculate *m_h* is *s_h* . The height of the geoid above the ellipsoid in cm is *Geoid.* The measured 10-per-second sea surface heights used to calculate *m_h* are *h[1]-h[10].* The average significant wave height in cm is *swh* and *s_swh* is the standard deviation of the measurements used to determine *swh.* The backscatter coefficient, *s_naught* is computed aboard the spacecraft in 0.01 dB. The automatic gain control, *agc*, is also determined aboard the spacecraft and *s_agc* is the standard deviation of the measurements used to determine *agc.* The height offset used for all measurements over land is *h_off* . The correction to *m_h* for the solid earth tide is *sol_tide* and the correction to *m_h* for the ocean tide is *oc_tide.* The correction to *m_h* to account for the time delay caused by water vapor in the troposphere, *wet_fnoc*, is derived from the Fleet Numerical Oceanographic Center (FNOC) NOGAPS model. An alternative correction for the water vapor is given as *wet_smmr.* The correction for the dry troposphere is given as *dry_fnoc.* A correction for the altimeter time delay due to molecules in the troposphere is given by *dry_fnoc*, which is also calculated from the FNOC NOGAPS model. The correction resulting from free electrons in the ionosphere is given by *iono_gps.* Two corrections are also given for height bias. The correction *dh_swh* is from a combination of significant wave height and attitude bias and *dh_fm* is due to compression of the altimeter pulse. The final parameter, *att* is the off-nadir satellite orientation angle. See the GEOSAT Altimeter GDR User Handbook (Cheney et al. 1987) for more information and references for these parameters.

3 SSMI Data Record

The raw SSMI data were collocated with the GEOSAT subtrack by Wentz [1989]. The collocated SSMI data records are at 10 second intervals and consist of 12 bytes as described in table 2. A wind speed value of 45 denotes no wind data available due to rain and a columnar water vapor value of 10 denotes no vapor data available due to rain.

Geophysical Data Record Contents					
Item	Parameter	Abbreviation	Units	Range	Bytes
1	UTC	utc	Seconds	0 to 2^{31}	4
2	UTC(cont'd)	utcm	Micro Second	0 to 1E6	4
3	Latitude	lat	Micro Degrees	+/- 7.21E7	4
4	Longitude	lon	Micro Degrees	0 to 360E8	4
5	Orbit	orb	Millimeter	7E8 to 9E8	4
6	H	m_h	Centimeter	+/- 32766	2
7	Sigma_H(σ_H)	s_h	Centimeter	0 to 32766	2
8	Geoid	geoid	Centimeter	+/- 1.5E5	2
9	H(1)	h[1]	Centimeter	+/- 32766	2
10	H(2)	h[2]	Centimeter	+/- 32766	2
11	H(3)	h[3]	Centimeter	+/- 32766	2
12	H(4)	h[4]	Centimeter	+/- 32766	2
13	H(5)	h[5]	Centimeter	+/- 32766	2
14	H(6)	h[6]	Centimeter	+/- 32766	2
15	H(7)	h[7]	Centimeter	+/- 32766	2
16	H(8)	h[8]	Centimeter	+/- 32766	2
17	H(9)	h[9]	Centimeter	+/- 32766	2
18	H(10)	h[10]	Centimeter	+/- 32766	2
19	SWH	swh	Centimeter	0 to 2E3	2
20	Sigma_SWH(σ_{swh})	s_swh	Centimeter	0 to 2E3	2
21	Sigma_naught(σ_o)	s_naught	0.01 dB	0 to 6.4E3	2
22	AGC	agc	0.01 dB	0 to 6.4E3	2
23	Sigma_AGC(σ_{AGC})	s_agc	0.01 dB	0 to 6.4E3	2
24	Flags				2
25	H Offset	h_off	Meters	0 to 5.4E4	2
26	Solid Tide	sol_tide	Millimeter	+/- 1000	2
27	Ocean Tide	oc_tide	Millimeter	+/- 10000	2
28	Wet (FNOC)	wet_fnoc	Millimeter	0 to -1000	2
29	Wet (SMMR)	wet_smmr	Millimeter	0 to -1000	2
30	Dry (FNOC)	dry_fnoc	Millimeter	-2000 to -3000	2
31	Iono (GPS)	iono_gps	Millimeter	0 to -500	2
32	dh (SWH/ATT)	dh_swh	Millimeter	+/- 9999	2
33	dh (FM)	dh_fm	Millimeter	+/- 999	2
34	Attitude	att	0.01 Degree	0 to 200	2

Table 1:

SSMI Data Record Contents				
Item	Parameter	Units	Range	Bytes
1	Time	Seconds	0 to 2^{31}	4
2	Latitude	Degrees	-	2
3	Longitude	Degrees	-	2
4	Encoded Data	-	See table 3	4

Table 2:

SSMI encoded data description				
Item	Parameter	Abbreviation	Range	Units
1	Data flag	fl	0 to 3	0 - Over ocean
				1 - No orbit altitude information
				2 - Over land
				3 - Over sea ice
2	Wind speed	ws	-	ms^{-1}
3	Columnar water vapor	vp	-	$gr \cdot cm^{-2}$
4	Columnar cloud	cl	-	$gr \cdot cm^{-2}$
	water vapor		-	
5	Rain rate	rn	-	$mm \cdot hr^{-1}$

Table 3:

4 Data Handling

In this text, the data received from NODC is referred to as "raw" and should not be confused with the data received directly from the satellite JHU/APL. Each data tape contains approximately 34 days of data for a total of more than 120 Megabytes so that it is impractical to keep all available data on disk.

Since these programs were developed to analyze mesoscale features, the data is split from the raw sequential input data into regional areas. The repeat analysis required developing an orbit numbering scheme to identify collinear orbits. This scheme separates the GDRs into ascending and descending orbit segments starting and ending at the most northern and most southern point of an orbit. An orbit is defined to be the combination of the ascending and descending segments beginning with the descending segment. A segment is defined as any part of a complete orbit. These orbits were numbered from 0 to 243 with zero being the first orbit on the first NODC data tape. Since the orbits repeat every 17.05 days, the orbits were also named by the repeat cycle from which they were extracted. A repeat cycle is defined as the combination of all orbits beginning with 0 and ending with 243. The cycles are also numbered consecutively starting with zero.

The resulting files are named *cmmm.dnnn* for descending orbit *nnn* and *cmmm.annn* for ascending orbit *nnn* from repeat cycle *mmm*. Table 4 shows the naming conventions used in this report for the various files created during analysis.

Naming conventions		
Convention	Example	Description
c*mmm*.a*nnn*	c002.a088	Raw GEOSAT binary files
c*mmm*.a*nnn*c	c002.a088c	Cleaned and corrected GEOSAT binary files
c*mmm*.a*nnn*cs	c002.a088cs	Cleaned, corrected and regridded GEOSAT binary files
c*mmm*.a*nnn*cs_r	c002.a088cs_r	Residuals from repeat analysis, ASCII format
a*nnn*cs_m	a088cs_m	Mean and variability for repeat analysis, ASCII format
c*mm*.a*nnn*asc	c002.a088asc	ASCII file containing extracted data

Table 4:

One alternative orbit numbering method is based on the longitude where the orbit crosses the equator. This method, however, does not convey the order of each orbit in time. Orbit c010.a045 passes the Gulf Stream approximately three days before c010.a088. Knowing the equatorial crossing of an orbit segment can be useful in quickly locating an orbit in space relative to another orbit or for comparing results with other numbering methods. A program was written to convert the sequential numbering to the equatorial numbering.

5 Programs and Subroutines

This section contains descriptions of programs and subroutines used to analyze GEOSAT data. In the examples given, the UNIX prompt is represented by a percent sign "%".

Most programs were designed to read and write the standard 78-byte GEOSAT GDR so that the output from one program may be used as the input for another. Programs are also simple and single-purpose. Instead of a program that removes spurious data points and applies orbit corrections, one program is used to apply the corrections and one program is used to remove unwanted data. This allows quick code modifications and substitutions. An alternative program to compute orbit corrections can be directly substituted for the supplied correction program. A single multi-purpose program would require major modifications to implement the new corrections.

Several programs were designed to read or write ASCII data for use with existing plotting packages and display programs. ASCII data allows users to choose their own display programs. One program which reads ASCII data was designed to interface directly with the high resolution color graphics capabilities of the Satellite Data Processing System (SDPS)(Caruso and Dunn, 1989) developed at the Woods Hole Oceanographic Institution. Complete UNIX style manual pages for all programs are included in appendix A.

Most programs have a single input file, a single output file and accept command line arguments as needed. This allows the output of one program to be piped into the input of another program. The simple and modular design of these programs allows users to combine programs to customize more complex programs. Several scripts were written for the UNIX shell (a command line interpreter)[3] to utilize this versatile feature. By combining several commands into a shell script, a user can quickly modify the analysis without changing program code and recompiling. For example, a simple shell script to perform a repeat analysis would look similar to this:

```
#
foreach i (c???.$1)
echo $i
#
cat $i | g_clean1 | g_correct | g_clean2 >! tmp
(cat tmp | g_spike | g_spline 1 22 48 3.3 0.97992165 > "$i"c)
end
#
echo Performing repeat analysis.
g_repeat "$i"c > mean."$1"
```

This uses three routines to clean the data, one routine to apply the standard corrections and one routine to spline the data onto an even grid for each cycle of a given

[3]Several shell programs are available. The examples given use the C shell.

orbit. Then the repeat analysis is done. The script takes as an argument the orbit number.

```
%repeat.sh a002
```

The user could use a program to apply non-standard corrections by substituting the program in the shell script.

```
#
foreach i (c???.$1)
echo $i
#
cat $i | g_clean1 | my_correct | g_clean2 >! tmp
(cat tmp | g_spike | g_spline 1 22 48 3.3 0.97992165 > "$i"c)
end
#
echo Performing repeat analysis.
g_repeat "$i"c > mean."$1"
```

5.1 GEOSAT Programs

A list of available **GEOSAT** analysis programs is given in table 5 with a brief synopsis. More detailed descriptions of programs are listed below in alphabetical order. All GEOSAT programs begin with *g_* to help provide unique program names.

5.1.1 g_clean1

This program is used to delete raw GEOSAT GDRs which contain obviously bad data. This includes all records that have any of the following variables set to 32767: the sea surface height, *ha*, and the corrections for earth tide, *cet*, ocean tide, *cot*, FNOC wet, *wet_fnoc*, or dry troposphere, *dry_fnoc*, or the ionosphere, *iono*. Records are also removed if the standard deviation, *s_h*, of the 10-per-second sea height values, *h[1]* - *h[10]*, is greater than 30 cm, or if the backscatter coefficient, *s_naught*, is greater than 35 dB. This program reads in a binary GEOSAT file and removes all bad records. The number of bad records is printed along with the criteria for rejection. For example the command

```
%cat c000.a002 | g_clean1 > c000.a002c
```

produces:

```
g_clean1: Valid points:      345
          Rejected points:    16
          Height:              0
          Solid Tide:          0
          Ocean Tide:          7
          Wet FNOC:            0
          Dry FNOC:            0
          Iono:                0
          Sigma Height:       14
          Sigma Naught:        0
          Flags:              15
```

List of GEOSAT programs	
Program	Description
g_clean1	Initial cleaning of raw GDRs
g_clean2	Secondary cleaning of GDRs
g_compress	Compresses GDR to 18 bytes
g_correct	Applies GDR suggested corrections to sea surface height
g_crossnum	Converts sequential orbit numbers to equatorial crossing longitudes
g_date	Prints start and end date for GDR segment
g_date2	Prints start and end date given cycle and orbit number
g_ext	Extracts one or more parameters from GDR and converts to SI units
g_image	Converts ASCII GEOSAT data to a bitmap image.
g_interp	Linearly interpolates to a specific grid
g_print	Decodes GDRs and prints to a terminal
g_region	Separates raw GEOSAT GDRs into sequential orbits in a specified region
g_repeat	Performs "collinear" or repeat track analysis using a quadratic orbit correction
g_repeats	Performs "collinear" or repeat track analysis using a sinusoidal orbit correction
g_seporb	Separates raw GEOSAT GDRs into sequential orbits
g_spike	Removes data spikes from GDRs
g_spline	Splines GDRs to a specific grid
g_uncompress	Uncompressis 18 byte data record
g_which	Prints orbit numbers in a given lat/lon box

Table 5:

This shows that a total of 16 records were rejected. Of those 16 records, 15 were rejected because the flag records were bad, 14 were rejected because the standard deviation of the 10-per-second sea height values were greater than 30 cm. Seven were rejected because the ocean tide value was set to 32767. By default, all records over land are also rejected. This most likely accounts for the 15 records rejected because of a bad flag value. This default may be changed to also remove all records over shallow water by specifying the correct flag mask. Any of the available flags supplied in the GEOSAT GDR may be tested. This is done by setting the UNIX environment variable *GMASK*:

```
%setenv GMASK 1 - - - - - - 0 - - - - - - 0 0
```

where a "-" means ignore this bit, a "0" means skip this record if this bit is not 0 and a "1" means skip this record if ths bit is not 1. For more information, see the manual page in appendix A. The results of this program are shown in figure 1.

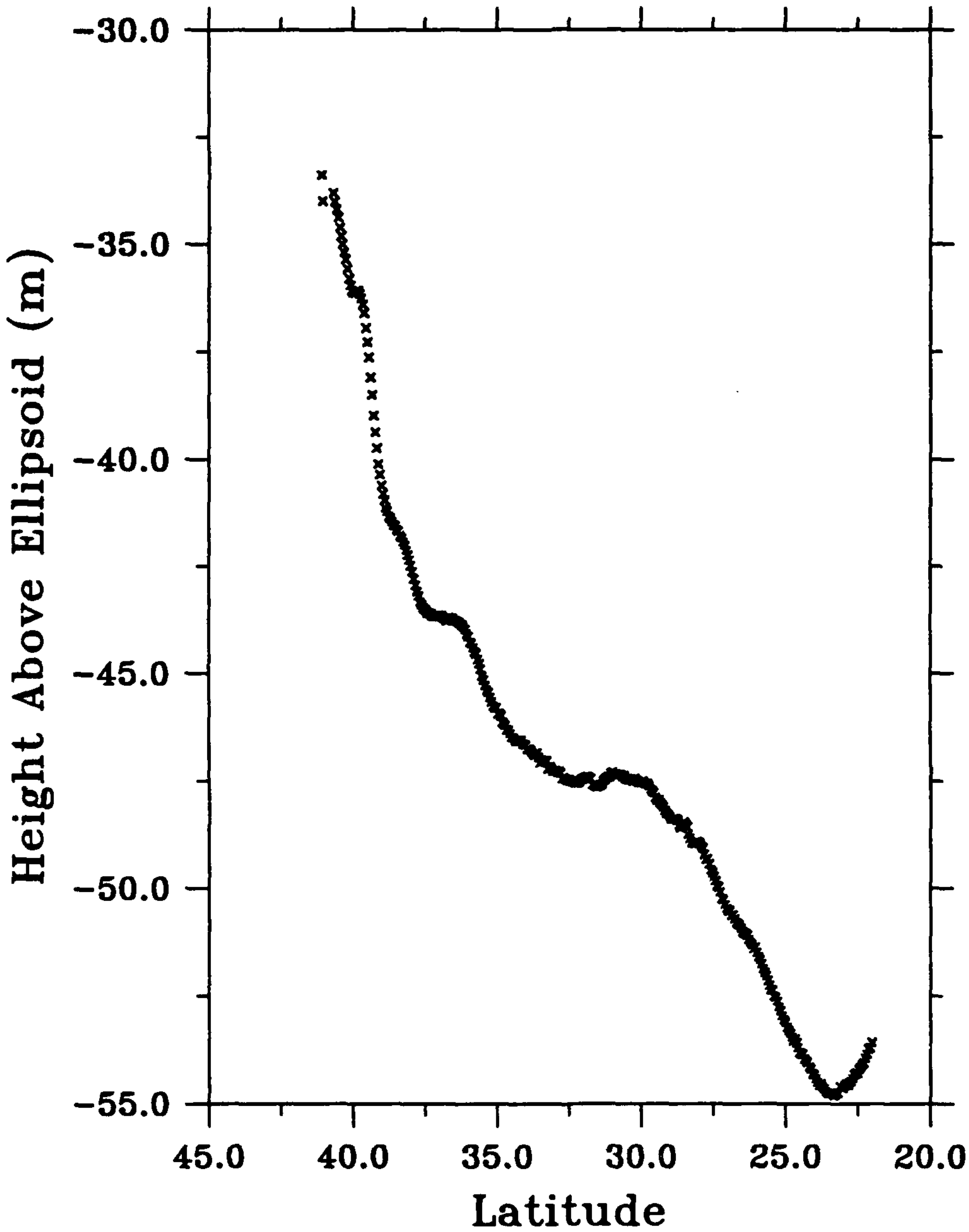

Figure 1: Raw sea surface heights plotted after running program g_clean1 to remove obviously bad data.

Compressed 18-byte Data Record Contents				
Item	Parameter	Units	Range	Type
1	Time	ms	0 to 1.47E9	long int (4-bytes)
2	Height	cm	0 to 32767	short int (2-bytes)
3	Cycle		0...	char (1-byte)
4	Latitude	10^4 Deg	0 to 18E5	unsigned int (3-bytes)
5	Longitude	10^4 Deg	0 to 36E5	unsigned int (3-bytes)
6	Sigma Height	cm	0 to 255	unsigned char (1-byte)
7	SWH	5cm	0 to 255	unsigned char (1-byte)
8	s_naught	0.1dB	0 to 255	unsigned char (1-byte)
9	Flags			char (1-byte)
10	Ocean Tide	cm	-128 to 128	char (1-byte)

Table 6:

5.1.2 g_clean2

This program is used to clean up records after *g_clean1* and *g_correct* have been used. It simply removes data records with sea surface heights greater than 10000 cm and less than −14000 cm. This removes any obvious outliers that may interfere with other analysis programs such as *g_spline*. As in *g_clean1*, a total of rejected points is printed. The command

```
%cat c000.a002 | g_clean2 > c000.a002c
```

produces:

```
g_clean2: Rejected points:   0
          Maximum Height:    0
          Minimum Height:    0
```

In this case, file *c000.a002* is the output from *g_clean1* and *g_correct*.

5.1.3 g_compress / g_uncompress

This is a set of programs designed to compress the standard 78-byte GDR to 18 bytes by reducing precision and removing less important fields such as the 10-per-second sea surface heights. The output is an 18-byte-per-record binary file and should be uncompressed before using any of the other analysis programs. These programs were designed for storing as much meaningful data as possible on limited systems. The format of the compressed 18-byte record is given in table 6. The time variable stored is the time since the start of *a000* for each cycle. The other variables are the same as for the full GDR except with reduced precision.

5.1.4 g_correct

This program allows the user to apply one or more of the suggested corrections to the sea surface height value of each record. All suggested corrections are optional and are applied by default. An example of applying all corrections would be:

```
%g_correct < c000.a002 > c000.a002c
```

In this example, the file *c000.a002* is the output from *g_clean1*. The output file *c000.a002c* has the same format as the original GDR, but the height field now contains the following corrections:

$$h = h - sol\_tide - oc\_tide - wet\_fnoc - dry\_fnoc - iono\_gps - inv\_bar$$

where the corrections are supplied in the GDR (table 1) except for *inv_bar* which is given as follows:

$$inv\_bar = -9.948(p - 1013.3)$$

and

$$p = \frac{dry\_fnoc}{(-2.277)\{1 + [0.0026 \cos(2LAT)]\}}$$

Individual corrections may be applied by specifying the abbreviation on the command line,

%g_correct cet cot < c000.a002c > c000.a002c

This would apply the corrections for the earth tide and the ocean tide supplied with the GEOSAT GDR. The list of available abbreviations is given in table 7 and in the manual page in appendix A. These abbreviations also correspond to the abbreviations for *g_ext*. The corrections for a section of *c000.a002* are given in figure 2.

Description of variables for program g_correct	
Abbreviation	Description
cet	correction for earth tide in m
cot	correction for ocean tide in m
cwf	correction for wet troposphere fnoc
cws	correction for wet troposphere smmr
cdf	correction for dry troposphere
ci	correction for ionosphere
cib	correction for inverse barometric effect

Table 7:

5.1.5 g_crossnum

This program finds the longitude where a given orbit crosses the equator. This program was designed to convert sequential orbit numbers to equatorial crossing numbers. The program may be used in two ways. First, the specific orbit can be specified:

```
%g_crossnum a002
306.43
```

Second, the program may be given a GEOSAT GDR:

```
%g_crossnum < c000.a002
306.43
```

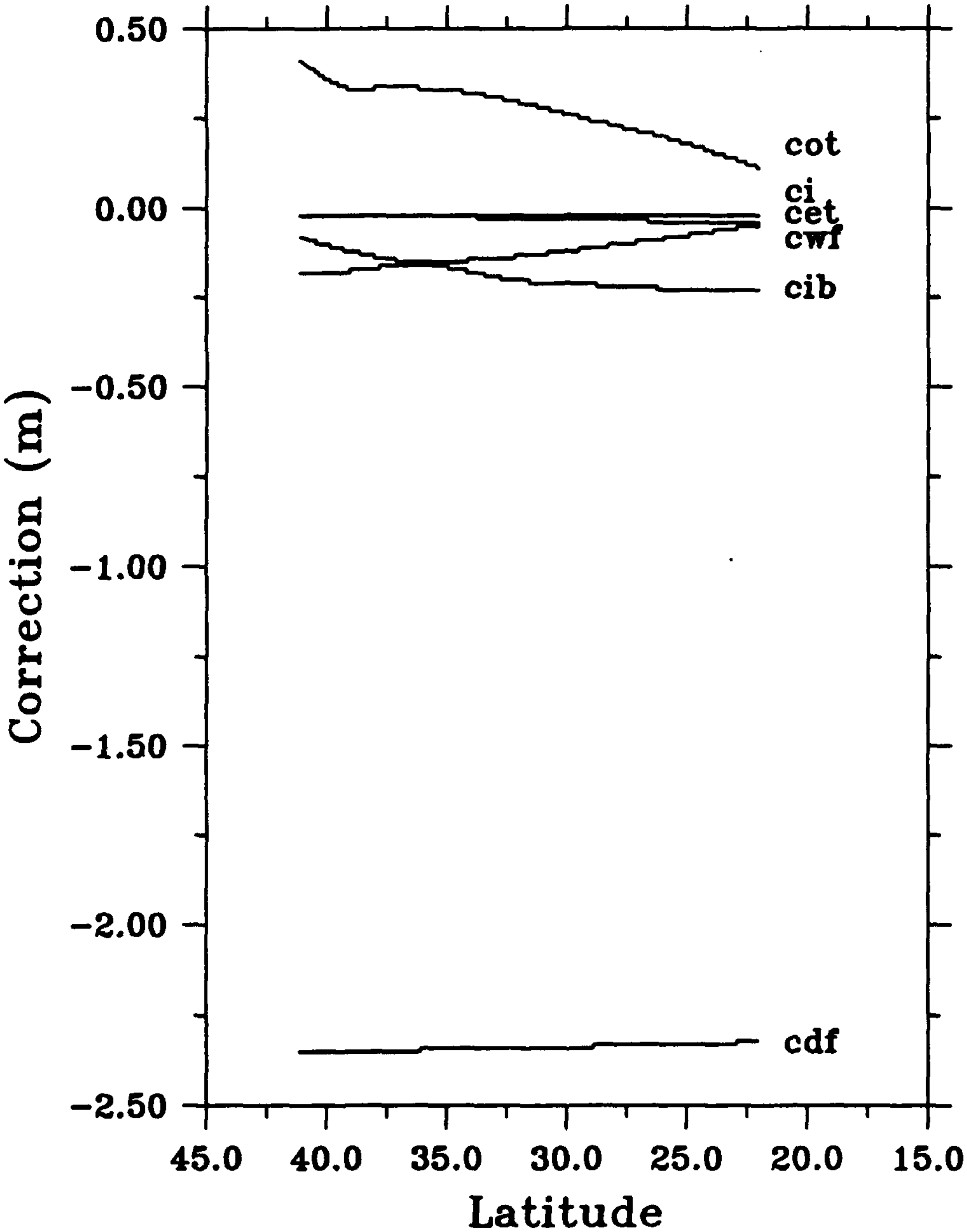

Figure 2: Corrections from a section of orbit c000.a002. *Cot* is for the ocean tide; *ci* is for the ionosphere; *cet* is for the earth tide; *cwf* is for the FNOC wet troposphere; *cib* is for the inverse barometric effect and *cdf* is for the FNOC dry troposphere.

5.1.6 g_date

This program prints the start and end date and time of a GEOSAT GDR segment. The program prints the *utc* value from the GDR, the date, the day of year, the Julian day and the day of the cycle.

```
%cat c053.a002 | g_date
UTC: 136497764.05
Date: 4/28/89 20:02:44
Day of year: 118
Julian: 1753685
Day of cycle: 0

UTC: 136498110.94
Date: 4/28/89 20:08:30
Day of year: 118
Julian: 1753685
Day of cycle: 0
```

5.1.7 g_date2

This program is similar to g_date except that it takes the orbit and cycle numbers as arguments. The program prints the approximate beginning and ending times of the specified orbit.

```
%g_date2 053 002
UTC: 136492860.00
Date: 4/28/89 18:41:00
Day of year: 118
Julian: 1753685
Day of cycle: 0

UTC: 136498897.00
Date: 4/28/89 20:21:37
Day of year: 118
Julian: 1753685
Day of cycle: 0
```

5.1.8 g_ext

This program was written to convert and extract one or more parameters in a GEOSAT GDR to ASCII format. It converts all parameters to SI units. To create an ASCII file of the latitude, longitude and uncorrected sea surface heights, the following command would be given:

```
%g_ext l L ha < c000.a002 > c000.a002asc
```

where c000.a002 is a file containing GDRs in binary format and c000.a002asc is the ASCII output from *g_ext*. This program allows the user to use almost any plotting package to display the data. For example, the command

```
%g_ext l w < c000.a002 | graph -g 1 -x 45 20 -5
```

uses the standard UNIX plotting utility *graph* to plot the significant wave height for orbit number 002 in cycle 000 over the Gulf Stream for figure 3. Compare this with the clean data plotted after using *g_clean1* in figure 1 to see how obviously bad points can be removed. The complete list of abbreviations is given in table 8 and in the manual page in appendix A. Figures 4 and 5 are examples of other fields that may be extracted and plotted using more sophisticated plotting packages.

Description of variables for program g_ext	
Abbreviation	Description
t	time in seconds since equator crossing of orbit c000.a000
l	latitude in degrees
L	east longitude in degrees
ho	orbit height above ellipsoid in m
ha	sea surface height above ellipsoid in m
sha	sigma ha
hg	geoid height above ellipsoid in m
w	significant wave height
sw	sigma w
so	backscatter coefficient in 0.01 dB
ag	agc in 0.01 dB
sag	sigma ag
fl	masked flags
HA	land surface height offset above ellipsoid
cet	correction for earth tide in m
cot	correction for ocean tide in m
cwf	correction for wet troposphere fnoc
cws	correction for wet troposphere smmr
cdf	correction for dry troposphere
ci	correction for ionosphere
b	attitude bias
bc	compression bias
att	attitude
cib	correction for inverse barometric effect
h	corrected sea surface height above ellipsoid
dh	corrected sea surface height above geoid

Table 8:

5.1.9 g_image

This program converts ASCII GEOSAT data in the form *latitude*, *longitude* and *z* to a bitmap image. An example of Gulf Stream variability calculated from the repeat track analysis using ascending orbits is shown in figure 6. The coastline and grid overlays on this figure were generated using SDPS.

This program takes six parameters, the minimum latitude and longitude, the maximum latitude and longitude and the number of rows and columns in the output image. The following was used to generate the image in figure 6:

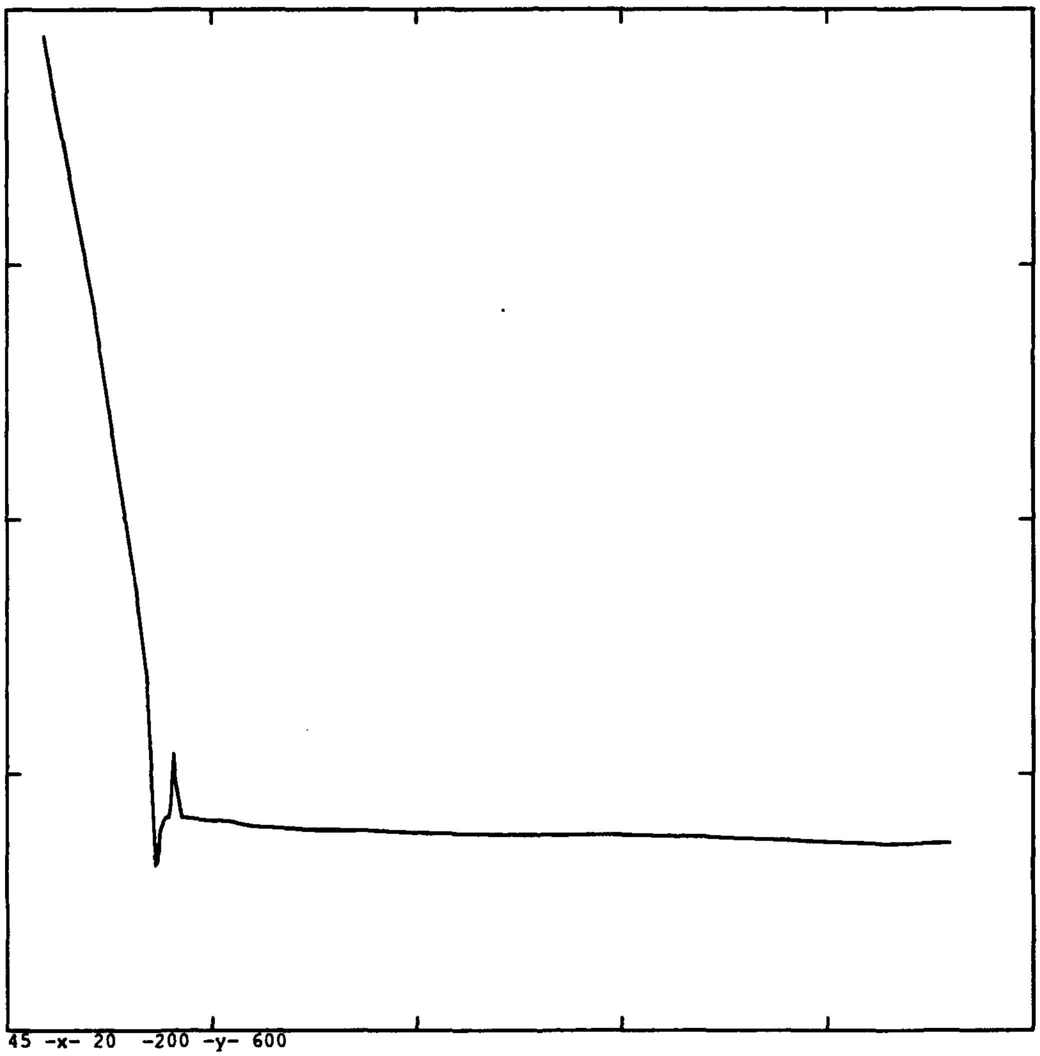

Figure 3: An example of using *g_ext* to extract raw sea surface heights. The UNIX utility *graph* was used to plot this figure.

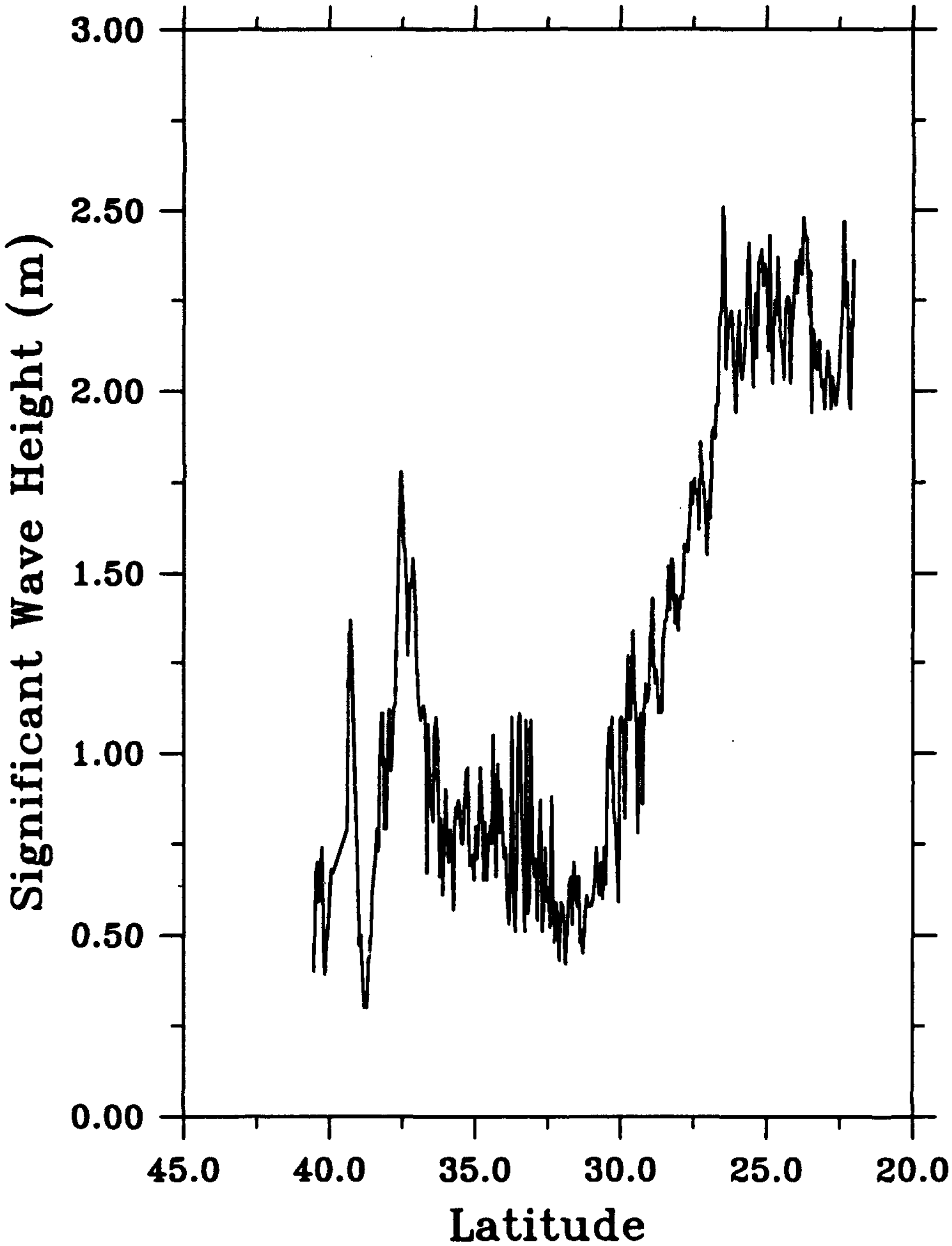

Figure 4: An example of using *g_ext* to extract the significant wave heights for orbit c000.a002 over the Gulf Stream.

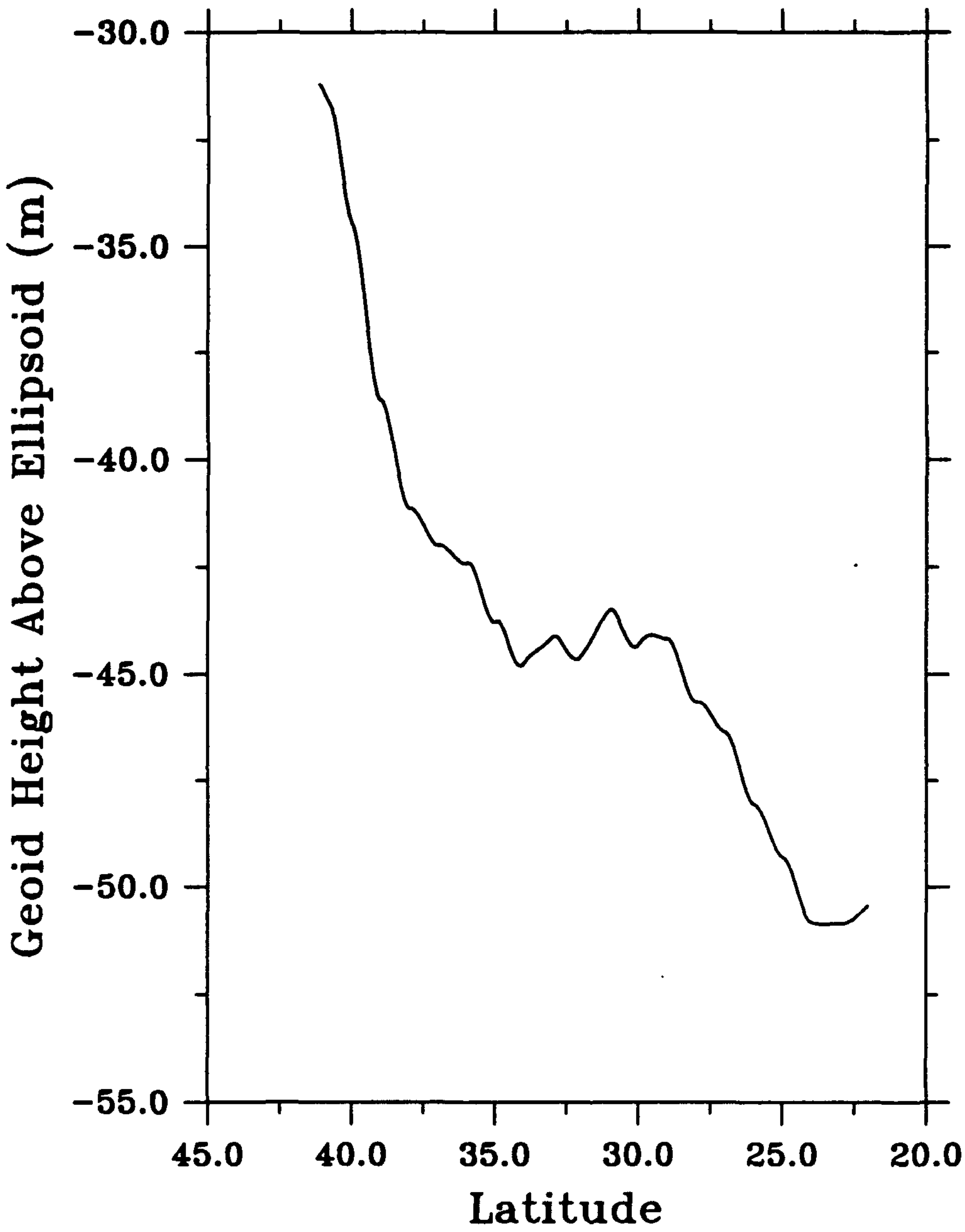

Figure 5: An example of using *g_ext* to extract included geoid heights for orbit c000.a002 over the Gulf Stream.

%cat a*cs_m | cut -f2,5 | g_image 22 48 284 316 416 512 > vara.sdpsf

The input is all the output files from *g_repeat* for each orbit in the region. The command *cut* is a standard UNIX command and illustrates how these programs are designed to be used with existing commands. The output image has 416 rows, 512 columns and is on an equirectangular grid 22N, 284E to 48N, 316E. This image is in SDPS floating point format and may be converted to byte format for display using the SDPS routine *sdps_ftb*:

%cat vara.sdpsf | sdps_ftb > vara.sdps

5.1.10 g_interp

This program is used to regrid the GEOSAT data to a common grid by linearly interpolating between supplied data points. All variables in the GDR are interpolated except the 10-per-second sea surface heights and the data flags since these fields are no longer meaningful to the regridded data. The output is regridded so that at least one value is positioned on the equator. This ensures that segments from areas that overlap, i.e. 10° N to 40° N and 25° N to 50° N, can be directly compared. The output file contains complete segments in GDR format.

Input segments should be cleaned and corrected and five arguments are required by the program:

%cat c000.a002 | g_interp *dir min max gap delta_t* > c000.a002c

where *dir* is 1 for an interpolation bounded by a minimum and maximum latitude and 2 for an interpolation bounded by a minimum and maximum longitude (see *g_region* section 5.1.12). A *gap* is the maximum time between good segments. The program does not spline across gaps, but labels the points as bad (32767). Gaps and incomplete cycles are filled to the boundaries defined by *min* and *max* with the correct latitude. The time between interpolated points is *delta_t*. One point is placed on the equator crossing and subsequent points are splined *delta_t* seconds apart. There are no default parameters. An example for the Gulf Stream region is:

%cat c000.a002 | g_interp 1 22 48 3.3 0.97992165 > c000.a002c

Here, the data is interpolated between 22° N and 48° N. If the segment has more than 3.3 seconds of missing data, it is considered to be a gap. The output points are interpolated to be 0.97992165 seconds apart, which is the same spacing as the raw GDRs.

5.1.11 g_print

This program decodes each GEOSAT GDR and prints the variables to a terminal. An example of the output is shown below:

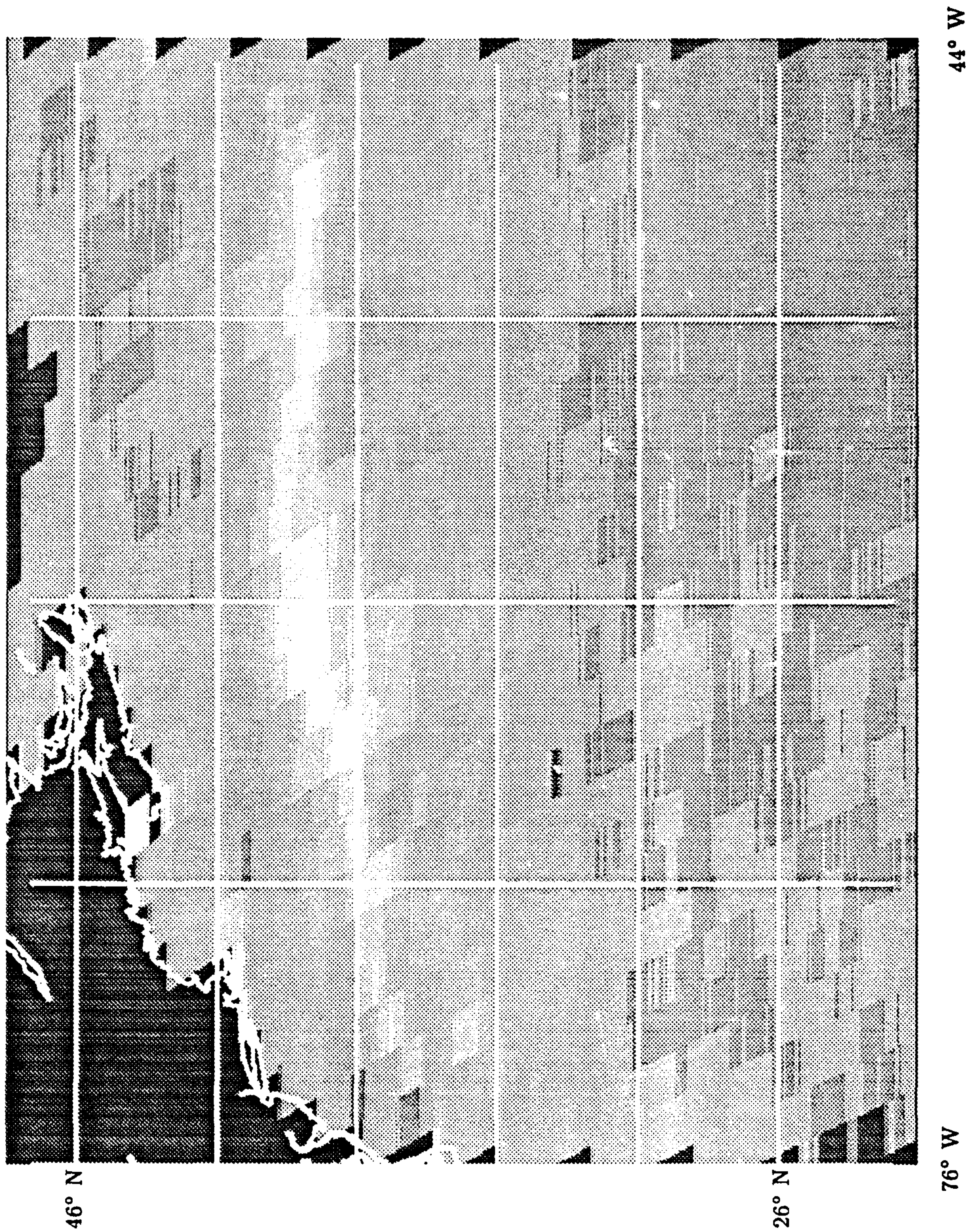

Figure 6: An example of an image generated using g_image. The grey shades represent the sea surface height variability and the coastlines were overlaid using SDPS.

Record Number: 1

utc	:	58939389	utcm	:	203366			
lat	:	22017008	lon	:	300236639			
orb	:	789454644						
m_h	:	−5353	s_h	:	4			
geoid	:	−4872						
h[1]	:	−5349	h[2]	:	−5348			
h[3]	:	−5349	h[4]	:	−5348			
h[5]	:	−5349	h[6]	:	−5349			
h[7]	:	−5358	h[8]	:	−5365			
h[9]	:	−5359	h[10]	:	−5359			
swh	:	253	s_swh	:	11			
s_naught	:	1088						
agc	:	2664	s_agc	:	2			

flags (0–15 right to left):			0000010000000011					
h_off	:	0						
sol_tide	:	188	oc_tide	:	-177			
wet_fnoc	:	−252	wet_smmr	:	−242	dry_fnoc	:	−2325
iono_gps	:	−16						
dh_swh	:	38						
dh_fm	:	30						
att	:	74						

where the units and names correspond to those given in table 1.

5.1.12 g_region

This program reads raw GEOSAT GDRs and separates them into individual ascending and descending orbits and extracts data from a user-specified region. The user may specify two types of regions. The first type of region is bounded by latitude lines, and the second is bounded by longitude lines. Figure 7 shows the ascending orbits extracted from a data set over the Gulf Stream bounded by latitude lines.

The smaller box in fig. 7 shows the latitude/longitude boundaries given to the program (22° N – 48° N, 284° E – 316° E). GDR segments were truncated at the minimum and maximum latitudes, but not at the minimum and maximum longitudes. This was done in order to keep reasonable ground track lengths in corners of the box since short segments would be useless for repeat analysis. Note that all the orbits to the right of the box actually extend until they intersect with the 22° N latitude line. This particular region was extracted using the command:

```
%g_region 1 22.0 48.0 284.0 316.0 < raw_geo
```

To extract files directly from the NODC HP format tape:

```
%dd if=/dev/rmt8 ibs=16380 files=34 | g_region 1 22.0 48.0 284.0 316.0
```

This command would extract the region shown in figure 7 and separate the data into ascending and descending orbits using the naming convention previously described in section 4. Since orbits may be split between tapes or tape files, the data is appended

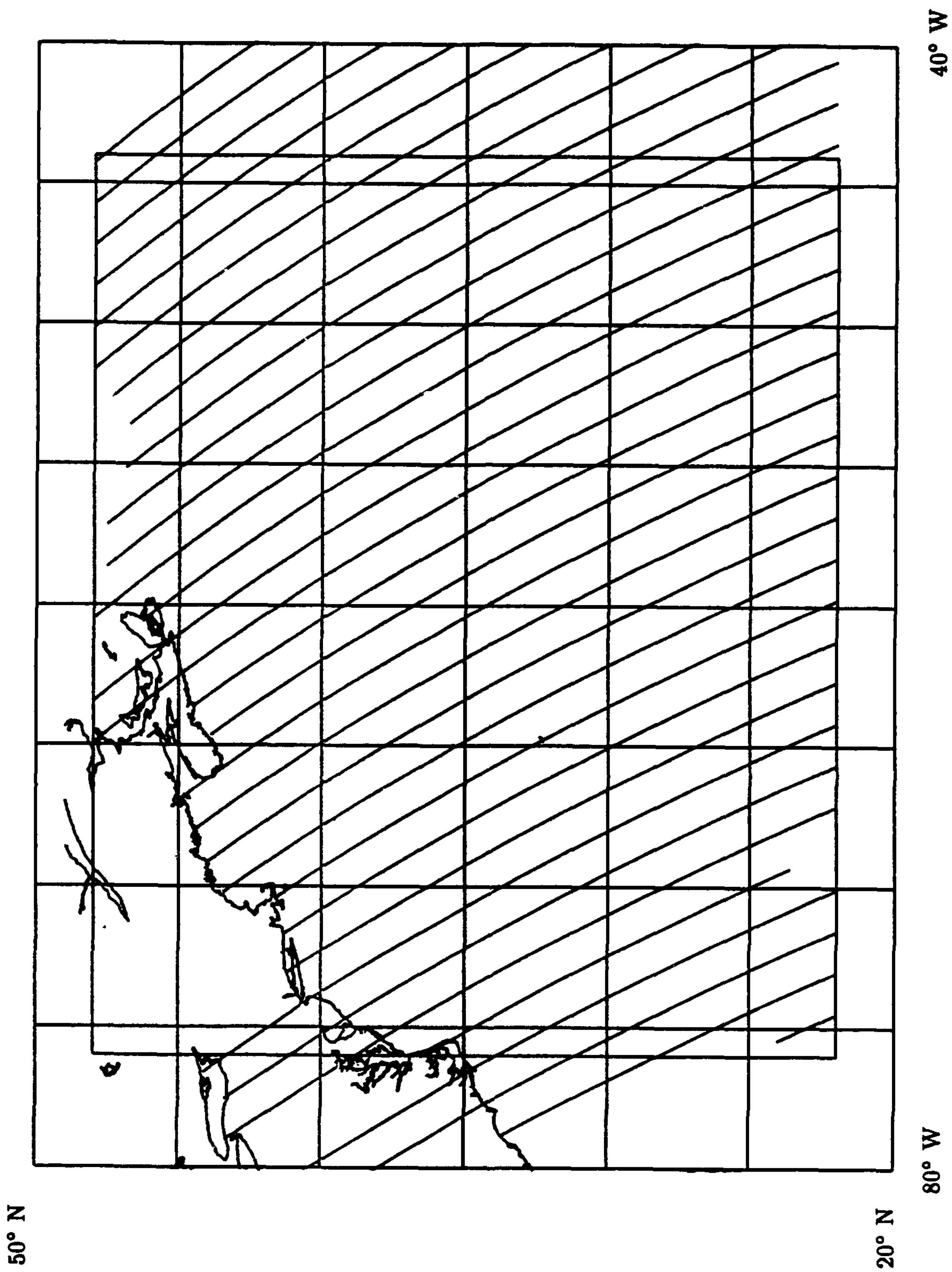

Figure 7: Ascending ground tracks of a region of the Northwest Atlantic.

to any existing files. This provides complete segments even if an orbit is split between data tapes or extracted data files. Since extracted regions do not have unique names, files should be moved or deleted if additional regions are to be extracted from the same tape. This prevents discontinuous regions from being appended together under the same file name.

Optional orbit numbers may be specified on the command line. If orbits are given, only the orbits which fall within the region are removed. To extract only orbits *002* and *088* the following command would be used:

```
%dd if=/dev/rmt8 ibs=16380 files=34 | g_region 1 22.0 48.0 284.0 316.0 2 88
```

5.1.13 g_repeat

This program performs a repeat track analysis of GEOSAT GDRs and assumes that all the GDRs have been cleaned (*g_clean1*, *g_clean2*), corrected (*g_correct*) and splined (*g_spline*) or interpolated (*g_interp*) to a uniform grid. It also assumes that each cycle contains the same start and end points. The program reads in all available GDRs and calculates the mean sea surface height for each grid point. If a sea surface height is set to 32767, the point is assumed to be bad and is not used to find the mean. This mean height profile, averaged over all cycles, is then subtracted from each individual track to produce a residual sea surface height profile:

$$h(x_i,t) - \langle h(x) \rangle = \widehat{y}(x_i,t) \tag{1}$$

where x_i is the location along the subtrack, t is the cycle number, $h(x_i,t)$ is the cleaned and corrected sea surface height profile, $\langle h(x) \rangle$ is the initial estimate for the mean height profile, and $\widehat{y}(x_i,t)$ is the residual sea surface height. A quadratic function, $a(t)x_i^2 + b(t)x_i + c(t)$, is calculated for each residual height for each cycle t using a least squares fit which minimizes:

$$\epsilon^2 = \sum_{x_i}^{n} \left\{ \widehat{y}(x_i,t) - [a_1(t)x_i^2 + b_1(t)x_i + c_1(t)] \right\}^2 \tag{2}$$

This quadratic estimate of the orbit error is removed from the residual height for each cycle to obtain a new residual height, $\widehat{z}$, where

$$\widehat{z}(x_i,t) = \widehat{y}(x_i,t) - [a_1(t)x_i^2 + b_1(t)x_i + c_1(t)] \tag{3}$$

The variance, σ^2, of all the height residuals $\widehat{z}$ for each subtrack is calculated by:

$$\sigma^2(x_i) = \frac{1}{N(x_i)} \sum_{t=0}^{N} \{\widehat{z}(x_i,t)\}^2 \tag{4}$$

where $N(x_i)$ is the number of good data at the point x_i. A second quadratic, weighted by the inverse of the variance, is fit to the residual $\widehat{z}$ to minimize:

$$\epsilon^2 = \sum_{x_i}^{n} \left\{ \widehat{y}(x_i,t) - [a_2(t)x_i^2 + b_2(t)x_i + c_2(t)] \right\}^2 \frac{1}{\sigma^2(x_i)} \tag{5}$$

The resulting quadratic orbit error estimate is then removed from each profile to obtain a corrected height profile:

$$\tilde{h}(x_i,t) = h(x_i,t) - [a_2(t)x_i^2 + b_2(t)x_i + c_2(t)] \tag{6}$$

and the geoid profile, $g(x_i)$, is calculated by averaging the corrected height profiles, $<\tilde{h}(x_i,t)>$.gegingrou The sea surface height residuals are calculated for each cycle:

$$h'(x_i,t) = \tilde{h}(x_i,t) - g(x_i) \tag{7}$$

and written to separate files based on the input file names. The geoid and sea surface height variability are also printed. Typically, the program is called using "*" or "?" wildcard file specifications:

%g_repeat c*/c*.a002c > a002cs_m

or

%g_repeat c???/c???.a002c > a002cs_m

The file *a002cs_m* contains the following information in tab delimited columns:

$$x_i \quad lat(x_i) \quad lon(x_i) \quad g(x_i) \quad \sigma^2(x_i) \quad \sum x^2 \quad N(x_i)$$

where x_i is a sequential counter of the points in the orbit section, $lat(x_i)$ and $lon(x_i)$ are the latitude and longitude at x_i, $g(x_i)$ is the estimated geoid, $\sigma^2(x_i)$ is the sea surface height variability, $\sum x^2$ is the sum of the squares of the sea surface heights and $N(x_i)$ is the number of cycles of good data found.

The sea surface height residuals, $h'(x_i,t)$, are written to a file in the same directory as the original raw data. The new file name is the same as the original with an *_r* appended to it, e.g., c000.a002 would become c000.a002_r. Each file contains the following information:

$$x_i \quad lat(x_i) \quad lon(x_i) \quad h'(x_i,t) \quad f_1(x_i,t) \quad f_2(x_i,t)$$

where x_i, $lat(x_i)$, $lon(x_i)$ are the same as in the file described above; $h'(x_i,t)$ is the corrected sea surface heights with the estimated geoid removed; $f_1(x_i,t)$ is the original quadratic fit $[a_1(t)x_i^2+b_1(t)x_i+c_1(t)]$ and $f_2(x_i,t)$ is the weighted quadratic fit $[a_2(t)x_i^2+b_2(t)x_i+c_2(t)]$.

5.1.14 g_repeats

This program is identical to *g_repeat*, except that a sinusoidal orbit correction is used. Here a sinusoidal estimate of the orbit error is removed from the residual height to obtain a new residual, $\hat{z}$, where equation 3 becomes

$$\hat{z}(x_i,t) = \hat{y}(x_i,t) - [a_1 sin(\frac{2\pi t}{T} + \phi_1) + b_1] \tag{8}$$

where t is the time of the GDR and T is the orbital period.

Similarly, equations 5 and 6 become

$$\epsilon^2 = \sum_{x_i}^{n} \left\{\hat{y}(x_i,t) - [a_2 sin(\frac{2\pi t}{T} + \phi_2) + b_2]\right\}^2 \frac{1}{\sigma^2(x_i)} \tag{9}$$

$$\tilde{h}(x_i,t) = h(x_i,t) - [a_2 sin(\frac{2\pi t}{T} + \phi_2) + b_2] \tag{10}$$

The program is used the same as *g_repeat* using "*" or "?" wildcard specifications:

%g_repeat c*/c*.a002c > a002cs_m

or

%g_repeat c???/c???.a002c > a002cs_m

Similarly, the file *a002cs_m* contains the following information in tab delimited columns:

$$x_i \quad lat(x_i) \quad lon(x_i) \quad g(x_i) \quad \sigma^2(x_i) \quad \sum x^2 \quad N(x_i)$$

The sea surface height residual files are similar to those created by *g_repeat*:

$$x_i \quad lat(x_i) \quad lon(x_i) \quad h'(x_i,t) \quad f_1(x_i,t) \quad f_2(x_i,t)$$

except that $f_1(x_i,t)$ is the original sinusoidal fit $a_1 sin(\frac{2\pi t}{T} + \phi_1) + b_1$ and $f_2(x_i,t)$ is the weighted sinusoidal fit $a_2 sin(\frac{2\pi t}{T} + \phi_2) + b_2$.

Figure 8 shows a comparision of the orbit corrections for both the quadratic and sinusoidal fit. The solid line represents the initial correction and the dashed line represents the weighted correction. The initial corrections both peak at 33° N which is near where the ground track crosses Bermuda. The weighted corrections are less influenced by Bermuda, but clearly the quadratic is still influenced.

5.1.15 g_seporb

This program was designed to separate raw GEOSAT data into separate orbits and number the files as described above. This is similar to *g_region* except that complete orbits are extracted from the original data instead of partial orbits within specific regions. The file naming conventions are consistent with *g_region* as described in section 4. To separate all orbits from the NODC HP format tape:

%dd if=/dev/rmt8 ibs=16380 files=34 | g_seporb

5.1.16 g_spike

This program was designed to remove data spikes from the data record. An example of data spikes is given in figure 9. This is data that passes through *g_clean1* and *g_clean2* without being removed.

This type of point may cause overshoot problems when the GDRs are splined using *g_spline* or may bias the repeat analysis. In any case, the data point is questionable and should be removed.

This program filters spikes by fitting a quadratic function or polynomial to a set of points in a least squares sense. Each orbit is split into contiguous segments where a discontinuity is defined as a gap between data points of 3.3 seconds or more. A polynomial is fit through each segment that contains at least 13 points. If a segment contains less than 13 points, it is removed from the record. If the point in question is more than 0.20 meters different from the quadratic fit, the two worst points are removed and a second 11 point quadratic is fit. If the point is still more than 0.20 meters from the polynomial, a straight line is fit through the data and the point is finally rejected if it is more that 0.20 meters from the line. The plot in figure 9 shows the result of *g_spike* for orbit *c022.a160*:

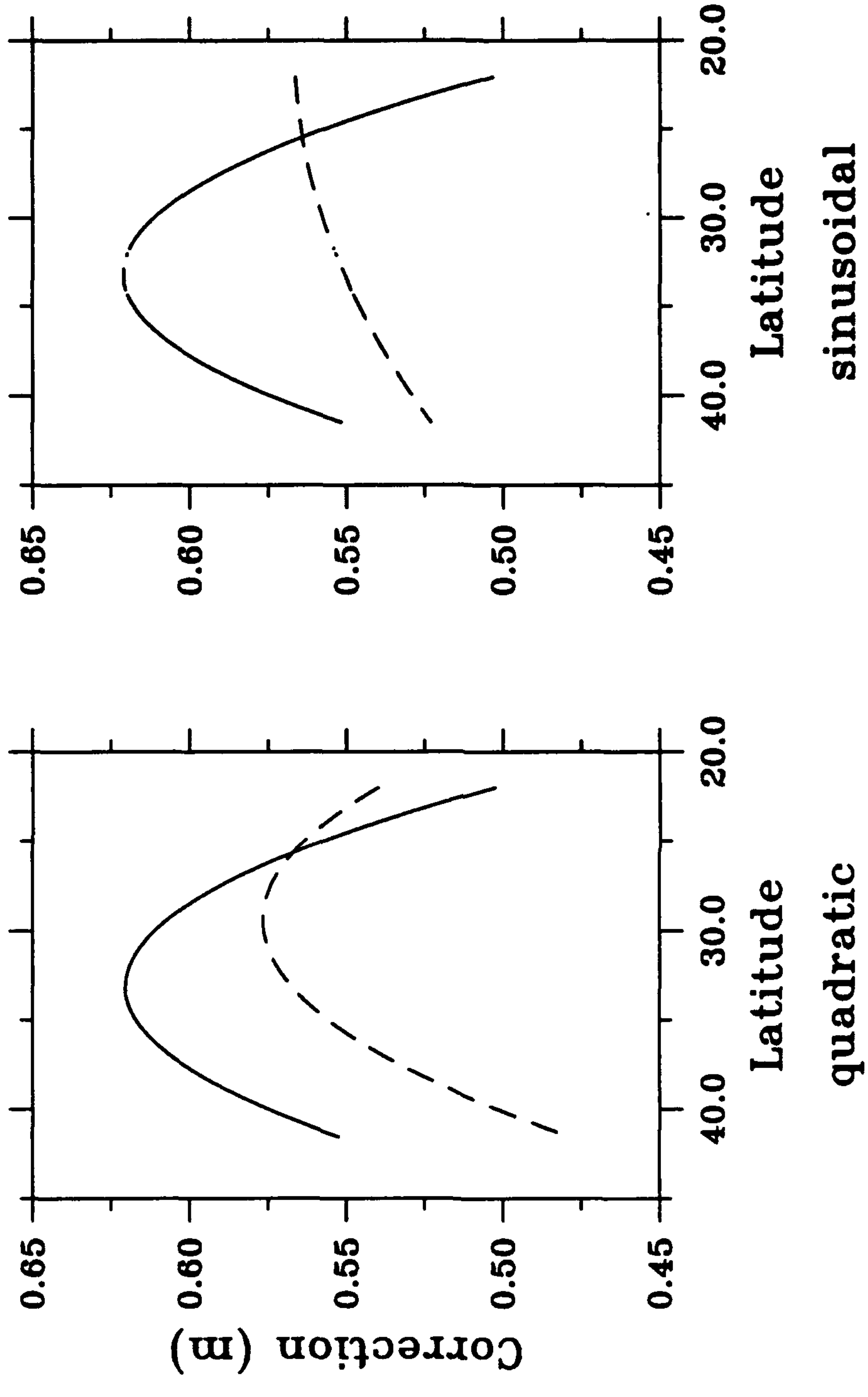

Figure 8: Orbit corrections of cycle *c000*, orbit *a088* for a quadratic fit (left) and a sinusoidal fit (right). The solid line is the initial correction and the dashed line is the weighted correction.

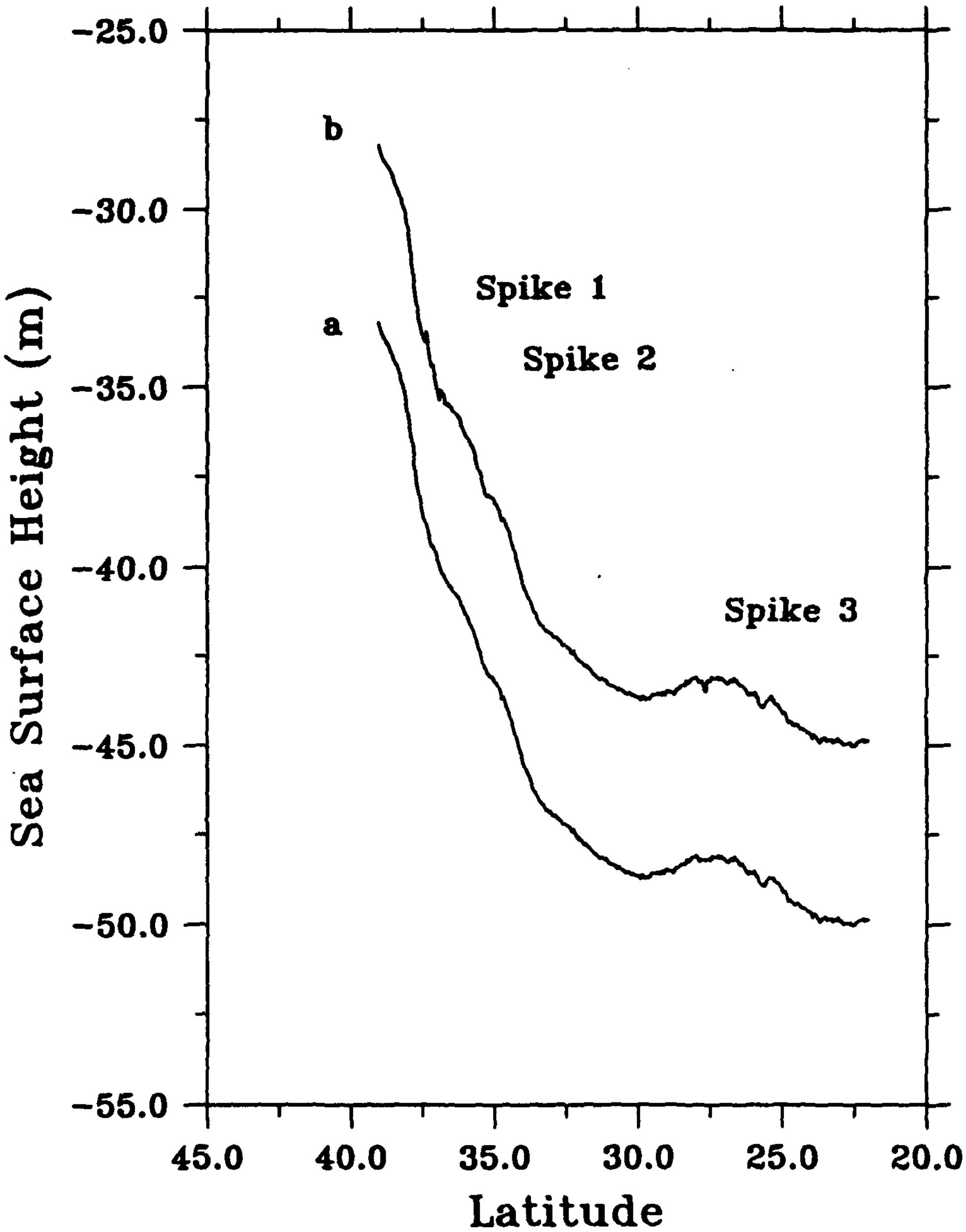

Figure 9: An example of the effect *g_spike* parameters have on data spikes for orbit c022.a160 over the Gulf Stream offset for clarity. Line a results from the default parameters where 3.3 seconds data gap, 13 fit points and 0.20 meter tolerance; b results from 3.3 second data gaps, 9 fit points and 0.50 meter tolerance.

%cat c022.a160c | g_spike > c022.a160cs

Since the default parameters are moderately restrictive, some data spikes may be retained or some valid data points may be rejected. It is important to check the results for spikes or missing data. The size of the data gap, the number of points and the height difference between the spline and the point being tested may be specified to fine tune the program:

%cat c022.a160 | g_spike 3.3 9 0.5 > c022.a160c

This command would split the data into segments separated by 3.3 seconds or more. The initial spline would contain 9 points and the second spline would contain 7 points. Each point would be rejected if it differed by more than 0.5 meters from each of the splines described above. For some orbits such as *a160,* these parameters can retain spikes (figure 9.) Although these spikes are negligible in the mean (figure 10), they can be important in the height residual (figure 11).

5.1.17 g_spline

This program will spline all the data in a given GDR except the 10-per-second heights and the data flags to a uniform calculated latitude grid, which has at least one value on the equator. This program is designed to be interchangeable with *g_interp* so the output also contains complete segments and the input is assumed to be cleaned and corrected GDRs. Also, the same 5 arguments are given on the command line and there are no defaults:

cat c000.a002 | g_spline *dir min max gap delta_t* > c000.a002c

where *dir* is 1 for a spline bounded by a minimum and maximum latitude and 2 for a spline bounded by a minimum and maximum longitude. Missing records are filled with the correct latitude and data values are labeled as bad points (32767). *Min* and *max* are the minimum latitude or longitude to spline between. *Gap* is the maximum time in seconds between continuous segments. The program does not spline across gaps, but labels the points as bad. *Delta_t* is the interpolation time step. One point is placed on the equator crossing and subsequent points are splined *delta_t* seconds apart. For the ascending orbits shown in figure 7, commands similar to the following were used:

cat c000.a002 | g_spline 1 22.0 48.0 3.3 0.97992165 > c000.a002c

The value of 0.97992165 was chosen to correspond to the actual separation of one-per-second GDRs.

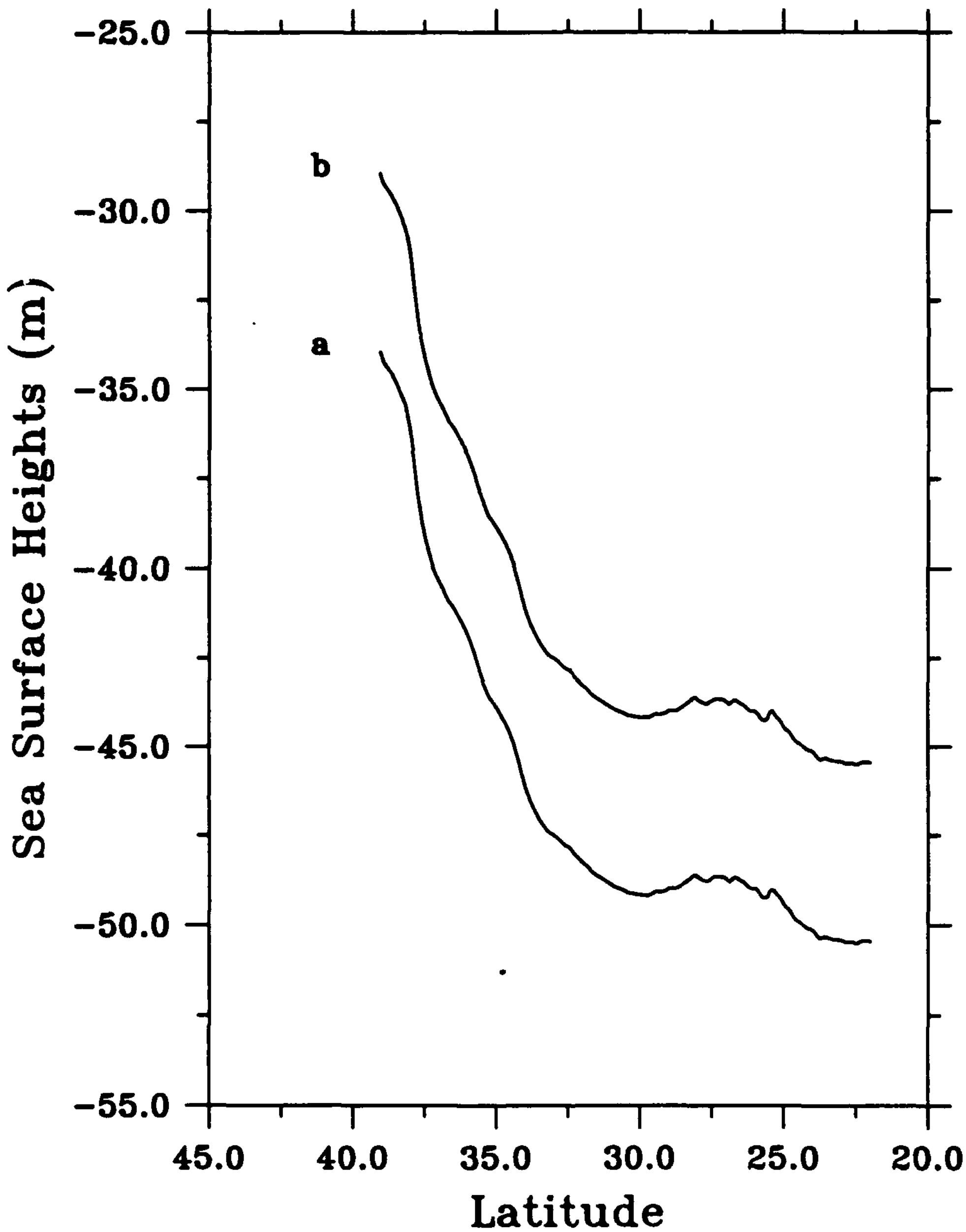

Figure 10: An example of the effect data spikes have on the mean for orbit c022.a160 over the Gulf Stream. Line a results from the default parameters and b results from 3.3 second data gaps, 9 fit points and 0.50 meter tolerance.

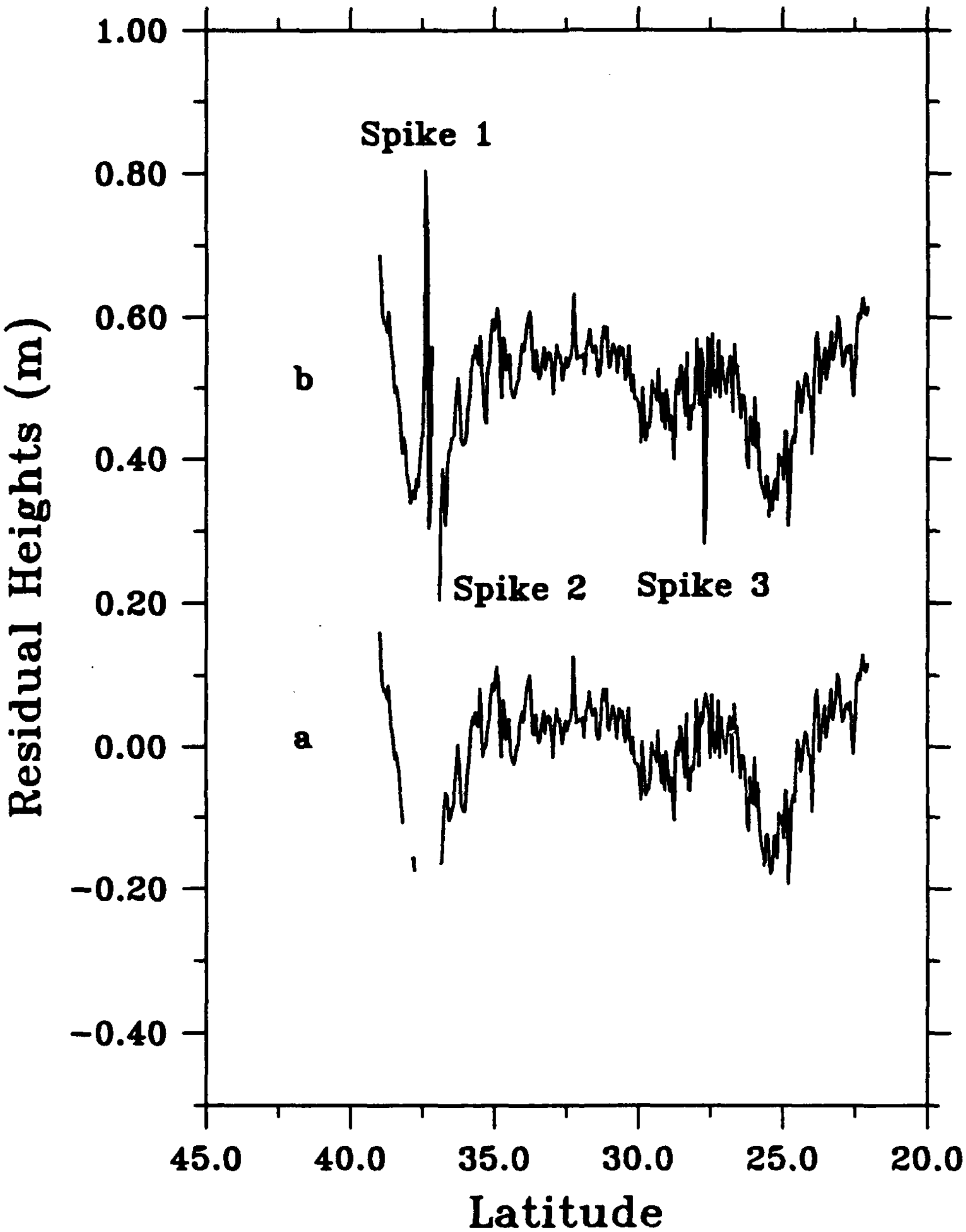

Figure 11: An example of the effect data spikes have on the residual for orbit c022.a160 over the Gulf Stream. Line a results from the default parameters and b results from 3.3 second data gaps, 9 fit points and 0.50 meter tolerance.

5.1.18 g_which

This program is designed to return all orbit numbers that cross within a specified latitude/longitude box. The arguments given to the program are the minimum and maximum latitudes and the minimum and maximum longitudes. To find all the orbits which cross a box 22° N to 48° N and 284° E to 316° E, the following command would be used:

```
g_which 22 48 284 316
```

With the following printout:

```
{a001,a002,d010,d011,a015,a016,d025,a030,a031,d039,d040,a044,
a045,d053,d054,a059,d068,a073,a074,d082,d083,a087,a088,d096,
d097,a102,d111,d112,a116,a117,d125,d126,a130,a131,d139,d140,
a145,a146,d154,d155,a159,a160,d168,d169,a173,a174,d182,d183,
a188,a189,d197,d198,a202,a203,d211,d212,a216,a217,d225,d226,
a231,a232,d240,d241}
```

If only a single latitude/longitude point is given, the program finds the closest ascending and descending track and prints that:

```
g_which 30 280
d083,a103
```

5.2 SSMI Programs

A list of available SSMI analysis programs is given in table 9 with a brief synopsis. More detailed descriptions are given below.

List of SSMI programs	
Program	Description
s_ext	Extracts one or more parameters and converts to SI units
s_region	Separates collocated SSMI records in sequential orbits in a specified region

Table 9:

5.2.1 s_ext

This program is similar to program *g_ext* except that it is designed to work on the SSMI data record. Usage is similar to *g_ext*. To extract the latitude, longitude and SSMI water vapor correction, the following command would be given:

```
%s_ext l L cws < s000.a002 > s000.a002asc
```

The complete list of abbreviations is given in table 10 and in the manual page in appendix A. A comparison between the water vapor corrections given in the GEOSAT GDR for a section of c015.a045 is given in figure 12.

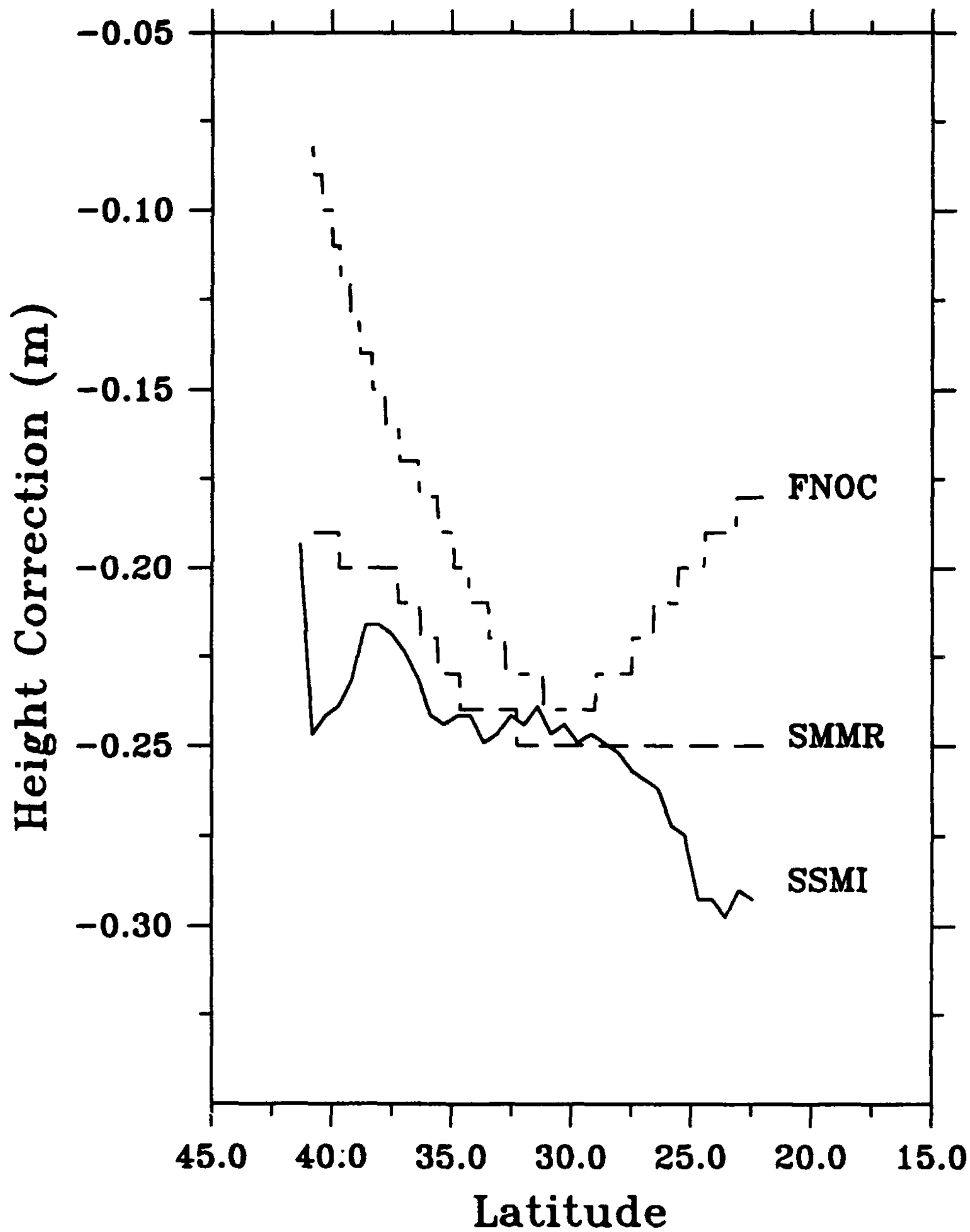

Figure 12: A comparison of the water vapor corrections from **GEOSAT GDR** and **SSMI**

Abbreviation	Description
t	time in seconds since equator crossing of orbit c000.a000
l	latitude in degrees
L	east longitude in degrees
fl	flag indicating data characteristic 1 – Over ocean 2 – No orbit altitude information 3 – Over land 4 – Over ice
ws	wind speed in ms^{-1}
vp	columnar water vapor in $kg\ m^{-2}$
cl	columnar cloud vapor in $kg\ m^{-2}$
rn	rain rate in $mm\ hr^{-1}$
cws	SSMI correction for water vapor in meters

Table 10: Description of variables for program s_ext

5.2.2 s_region

This program is similar to *g_region* except that it is designed to extract regions from SSMI data records. The orbits are numbered exactly the same as in *g_region*. The output cycles are also named the same except that the cycle numbers are preceded by an "s" instead of a "c". The program takes the first five arguments from *g_region*. To extract the area shown in figure 7, the following command would be given:

%s_region 1 22.0 48.0 284.0 316.0 < raw_ssmi

The output file named *s000.a002* would correspond to the **GEOSAT** file *c000.a002*.

5.3 Subroutines

A list of the subroutines developed for this project is given below with a brief synopsis of each routine.

5.3.1 geo_cyc_orb

This subroutine returns a cycle and orbit number for a given time. The subroutine is called:

geo_cyc_orb(time, &cyc, &orb)

where *time* is double precision and *cyc* and *orb* are integers. This can be used by any program that needs to know the cycle and orbit number for a given record, by passing the time variable in that record.

5.3.2 geo_error

This is a subroutine that is called to print out common error messages. The subroutine is called:

geo_error(num, str)

where *num* is the number of the error to print and *str* is the name of the program calling geo_error. The current messages are:

Number	Message
0	Unrecoverable error
1	c???.[ad]???
2	Error reading file
3	Error writing file

5.3.3 geo_mask

This subroutine reads the UNIX environment variable **GMASK** if it is available and converts it to an integer. **GMASK** is used to indicate to various programs which GDR flags should be checked and which flags should be ignored. See section 5.1.1 and A for more details on the use of **GMASK**. The subroutine is called:

geo_mask(&mask, &valid)

where *mask* and *valid* are short (16-bit) integers. *Mask* is returned with its bits set to 1 for each 1 in **GMASK**. *Valid* is returned with its bits set to 1 for each 0 or 1 in **GMASK**. See section 5.1.1 and A for more details on the use of **GMASK**.

5.3.4 geo_which

This subroutine performs the same function as the program g_which. It can be used by any program that needs to determine which orbit numbers fall within a given latitude-longitude box. The subroutine is called:

geo_which(min_lat, max_lat, min_lon, max_lon, a, d)

where *min_lat, max_lat, min_lon* and *max_lat* are the boundaries of the region, *a* and *d* are unsigned character arrays that are returned. The arrays *a* and *d* have 244 elements, one for each orbit. Each will be returned with a 1 in the element that corresponds to an orbit that passes through the specified box. Otherwise, the program returns a 0 in that element. Thus, if orbit *a002* passes through the given region, $a[2] = 1$.

6 Repeat Orbit Analysis

The programs described in section 5 were written to facilitate repeat orbit analysis. This section describes how these programs were used in conjunction to analyze a section of North Atlantic covering the Gulf Stream. The area of interest, 22° N to 48° N, 284° E to 316° E, is shown in figure 7, and is restricted to the ascending orbits. The analysis could also be performed for the descending orbits in a similar manner. The shell script in appendix C shows how these programs may be used together.

To perform a repeat analysis, the GDRs must first be removed from the data tape with the following command:

```
%dd if=/dev/rmt8 ibs=16380 files=34 | g_region 1 22.0 48.0 284.0 316.0 2 88
```

This extracts all the ascending and descending orbits within the specified region and places them in the current directory following the naming convention described in section 4. For the first GEOSAT tape, the directory listing is as follows:

c000.a001	c000.a145	c000.d053	c000.d197	c001.a074	c001.a216	c001.d126
c000.a002	c000.a146	c000.d068	c000.d198	c001.a087	c001.a217	c001.d139
c000.a015	c000.a159	c000.d082	c000.d211	c001.a088	c001.a232	c001.d140
c000.a016	c000.a160	c000.d083	c000.d212	c001.a102	c001.d010	c001.d154
c000.a030	c000.a173	c000.d096	c000.d225	c001.a116	c001.d011	c001.d155
c000.a031	c000.a174	c000.d097	c000.d226	c001.a117	c001.d025	c001.d168
c000.a044	c000.a188	c000.d111	c000.d240	c001.a130	c001.d039	c001.d169
c000.a045	c000.a189	c000.d112	c000.d241	c001.a131	c001.d040	c001.d182
c000.a059	c000.a202	c000.d125	c001.a001	c001.a145	c001.d053	c001.d183
c000.a073	c000.a203	c000.d126	c001.a002	c001.a146	c001.d054	c001.d197
c000.a074	c000.a216	c000.d139	c001.a015	c001.a159	c001.d068	c001.d198
c000.a087	c000.a217	c000.d140	c001.a016	c001.a160	c001.d082	c001.d211
c000.a088	c000.a231	c000.d154	c001.a030	c001.a173	c001.d083	c001.d212
c000.a102	c000.a232	c000.d155	c001.a031	c001.a174	c001.d096	c001.d225
c000.a116	c000.d010	c000.d168	c001.a044	c001.a188	c001.d097	c001.d226
c000.a117	c000.d025	c000.d169	c001.a045	c001.a189	c001.d111	c001.d240
c000.a130	c000.d039	c000.d182	c001.a059	c001.a202	c001.d112	c001.d241
c000.a131	c000.d040	c000.d183	c001.a073	c001.a203	c001.d125	

These files should then be moved into subdirectories named with the cycle number:

c000	c005	c010	c015	c020	c025	c030	c035	c040	c045	c050
c001	c006	c011	c016	c021	c026	c031	c036	c041	c046	c051
c002	c007	c012	c017	c022	c027	c032	c037	c042	c047	c052
c003	c008	c013	c018	c023	c028	c033	c038	c043	c048	c053
c004	c009	c014	c019	c024	c029	c034	c039	c044	c049	

The GDRs must then be cleaned, corrected and regridded. Data anomalies such as spikes or gaps should also be removed.

Following the shell script step-by-step, the GDRs are first cleaned and corrected using default values and stored in a temporary file *tmp:*

```
cat c000/c000.a002 | g_clean1 | g_correct | g_clean2 >! tmp
```

This temporary file is then cleaned of any remaining spikes and splined to a uniform grid and stored as a new file:

```
(cat tmp | g_spike | g_spline 1 22 48 3.3 0.97992165 > c000/c000.a002c)
```

These two steps are repeated using the *foreach* command for each cycle until all cycles are processed. Then the repeat analysis is performed:

```
g_repeat c*/c*.a002c > means/mean.a002c
```

G_repeat automatically writes the residual files to the same directory as the input clean files and appends an *_r* to the end of the filename. The output file *mean.a002c* contains the geoid and variability of the orbit *a002.*

Acknowledgements

The authors would like to thank Dr. Kathryn Kelly for her advice on algorithm development, Dr. Robert Beardsley for his support of this project and Debbie Barber for her suggestions on the text. Funding for this project was provided by the Office of Naval Research under contract number N00014-86-k-0751.

References

[1] J. Hollinger, R. Lo, G. Poe, R. Savage, and J. Pierce. *Special Sensor Microwave/Imager User's Guide*. Technical Report, Naval Research Laboratory, Washington, D.C., 1987.

[2] Robert E. Cheney, Bruce C. Douglas, Russell W. Agreen, Laury Miller, David L. Porter, and Nancy S. Doyle. *Geosat Altimeter Geophysical Data Record User Handbook*. Technical Report NOS NGS-46, National Oceanographic and Atmospheric Administration, July 1987.

[3] F. J. Wentz. *User's Manual: Collocated GEOSAT - SSM/I tape*. Technical Report RSS 083189, Remote Sensing Systems, Santa Rosa, CA, 1989.

[4] Michael Caruso and Chris Dunn. *Satellite Data Processing System (SDPS) Users Manual V1.0*. Technical Report WHOI89-13, Woods Hole Oceanographic Institution, Woods Hole, MA, 1989.

A Manual Pages

This section contains the UNIX style manual pages for each of the programs and subroutines listed in sections 5.1, 5.2 and 5.3.

NAME

geosat – Programs and subroutines for processing GEOSAT GDR files.

DESCRIPTION

This manual page describes the various programs available for processing **GEOSAT** GDR files. These programs were developed at the Woods Hole Oceanographic Institution to simplify the handling of raw NODC data tapes. These programs were designed to run under the 4.2/4.3 BSD UNIX operating system.

These programs were designed to take full advantage of existing UNIX commands as well as the UNIX file system.

ORBIT INFORMATION

LABELING

The original data from NODC comes in 17 files that contain 14 or 15 complete orbits each. These 17 files make a complete repeat cycle. To facilitate data handling, these files are broken up into individual orbits which are further broken up to an ascending component and a descending component. The ascending and descending components are broken at the most northern and southern excursion of the satellite. The resulting files are named:

cnnn.dmmm for descending orbit *mmm* of repeat cycle *nnn*
cnnn.ammm for ascending orbit *mmm* of repeat cycle *nnn*

By convention, orbit numbers *mmm* and cycle numbers *nnn* begin with 000 for the first orbit and cycle on the first data tape sent out from NOAA which begins on November 8, 1986. Also, by convention, the ascending orbit follows the descending orbit. The last point of an ascending or descending orbit is the most northern or most southern point of that orbit.

PARAMETERS

The following parameters were used in the various programs listed below. These parameters were computed by a least squares fit over cycles 000 and 001.

orbital period PERIOD	6037.5515	sec
repeat cycle 244*PERIOD	17.0504	days
distance between adjacent crossings 360/244	1.4754	deg
distance between successive crossings 17*360/244	-25.0820	deg
time of the equator crossing of c000.a000	58407697.82	sec
longitude of the equator crossing of c000.a000	356.58783	deg

LIST OF PROGRAMS

Name	Manual Page	Description
g_ext	g_ext(l)	Extracts GDR variables in ASCII format
g_clean1	g_clean1(l)	Removes obviously bad data from GDRs
g_clean2	g_clean2(l)	Removes bad data from corrected GDRs
g_compress	g_compress(l)	Compress GDRs to 18 bytes
g_correct	g_correct(l)	Applies suggested correction to GDRs
g_crossnum	g_crossnum(l)	Finds equator crossing of given orbit
g_date	g_date(l)	Prints start and end date for GDR segment
g_date2	g_date2(l)	Prints start and end date for given orbit and cycle
g_image	g_image(l)	Creates an bitmap image of GEOSAT data
g_interp	g_interp(l)	Interpolates to an even grid
g_print	g_print(l)	Decodes GDRs to ASCII format
g_region	g_region(l)	Extracts a region from continuous GDRs
g_repeat	g_repeat(l)	Preforms repeat analysis on GDRs
g_seporb	g_seporb(l)	Separates continuous GDRs into asc & desc orbits
g_spike	g_spike(l)	Removes spikes from GDRs
g_spline	g_spline(l)	Splines GDRs to an even grid
g_uncompress	g_uncompress(l)	Uncompresses 18 byte data record
g_which	g_which(l)	Prints orbit numbers in a given box

LIST OF SUBROUTINES

Name	Description
geo_error	Prints error messages to standard output.
geo_cyc_orb	Returns cycle and orbit number for given GDR
geo_mask	Reads environment variable GMASK
geo_which	Returns arrays of orbits which cross an area.

BUGS

Please report bugs to mcaruso@aqua.whoi.edu or pierre@io.soest.hawaii.edu

AUTHORS

Mike Caruso
Ziv Sirkes
Woods Hole Oceanographic Institution
Woods Hole, MA 02543

Pierre Flament
Mimi Baker
University of Hawaii
Honolulu, HI 96822

NAME

g_clean1 - cleans GEOSAT GDR data

SYNOPSIS

g_clean1

DESCRIPTION

This program reads a binary GEOSAT file from *stdin* and removes data records with one or more bad data flags, or if any of the following variables are set:

Variable	Description	Bad value
ha	sea surface height above ellipsoid	32767cm
sha	sigma ha	>30cm
so	sigma naught (backscatter coef)	>35dB
cet	correction for earth tide	32767mm
cot	correction for ocean tide	32767mm
cwf	correction for wet troposphere fnoc	32767mm
cdf	correction for dry troposphere	32767mm
ci	correction for ionosphere	32767mm

The user may also specify which data flags are to be used. A record will be skipped if the flag bits do not match the mask given by the environment variable GMASK. This variable should contain a string of characters describing the flag bits from 0 to 15 from left to right. A "-" means ignore this bit; a "0" means skip this record if this bit is not 0; a "1" means skip this record if this bit is not 1. If the variable GMASK is not set, the default mask "1 - - - - - - 0 - - - - - - 0 0" is assumed, i.e., the data over land is not printed. Examples of possible masks:

setenv GMASK 1 1 - - 0 0 0 0 - - - - - 0 0

will skip records over land and shallow water and for which the VATT is dubious;

setenv GMASK 0 0 - - - - - 0 - - - - -0 0

will print the data over land only.

AUTHOR

Mike Caruso
Woods Hole Oceanographic Institution
Woods Hole, MA 02543

SEE ALSO

geosat(l)

NAME

g_clean2 – cleans GEOSAT GDR data

SYNOPSIS

g_clean2

DESCRIPTION

This program reads a binary GEOSAT file from *stdin* and removes data records with sea surface heights greater than 10,000 cm or less than -14,000 cm. Output is in GEOSAT GDR format. The program assumes that corrections have been applied to the data previously.

AUTHOR

Mike Caruso
Woods Hole Oceanographic Institution
Woods Hole, MA 02543

SEE ALSO

geosat(l) g_correct(l)

NAME

g_compress, g_uncompress – compresses/uncompresses special 18-byte data record

SYNOPSIS

g_compress
g_uncompress

DESCRIPTION

These programs were designed to compress and uncompress a standard GDR to 18 bytes by reducing precision and removing less important variables. The format of the 18-byte record is given below.

Time	ms	0 to 1.47E9	long int (4-bytes)
Height	cm	0 to 32767	short int (2-bytes)
Cycle		0...	char (1-byte)
Latitude	10^4 Deg	0 to 18E5	unsigned int (3-bytes)
Longitude	10^4 Deg	0 to 36E5	unsigned int (3-bytes)
Sigma Height	cm	0 to 255	unsigned char (1-byte)
SWH	5cm	0 to 255	unsigned char (1-byte)
s_naught	0.1dB	0 to 255	unsigned char (1-byte)
Flags			char (1-byte)
Ocean Tide	cm	-128 to 128	char (1-byte)

AUTHOR

Pierre Flament
Oceanography Department
University of Hawaii
Honolulu, HI 96822

SEE ALSO

geosat(l)

NAME

g_correct – corrects GEOSAT GDR data

SYNOPSIS

g_correct *[cot cet cwf cdf ci cib]*

DESCRIPTION

This program reads a binary GEOSAT file from *stdin* and applies the following correction to the sea surface height.

ha(corrected) = ha - cet - cot - cwf - cdf - ci - cib

where

cib = -9.948 * (p - 1013.3)

and

p = cdf / ((-2.277)*(1. + (0.0026 * cos(2*latitude))))

using the following variables from the GDR.

Variable	Description
ha	sea surface height above ellipsoid
cet	correction for earth tide
cot	correction for ocean tide
cwf	correction for wet troposphere fnoc
cdf	correction for dry troposphere
ci	correction for ionosphere
cib	correction for inverse barometer

The program assumes that the input GDR has been previously cleaned up. The default is to apply all corrections. By selecting one or more of the option arguments, only those arguments given will be applied.

REFERENCE

Geosat Altimeter Geophysical Data Record User Handbook,
Cheney et al., NOAA Technical Memorandum NOS NGS-46, July 1987

AUTHOR

Mike Caruso
Woods Hole Oceanographic Institution
Woods Hole, MA 02543

SEE ALSO

geosat(l) g_clean1(l)

NAME

g_crossnum – finds the orbit crossing for a sequential **GEOSAT** orbit

SYNOPSIS

g_crossnum *[orbit]*

DESCRIPTION

This program may be used in two ways. First, it can be called with an orbit number:

g_crossnum a002

Second, if no arguments are given, the program will read a GEOSAT GDR from *stdin*:

g_crossnum < c000.a002

The output is the approximate longitude of the orbit crossing at the equator. This is useful for comparing sequentially numbered orbits with orbits numbered by equatorial crossing.

AUTHOR

Mike Caruso
Woods Hole Oceanographic Institution
Woods Hole, MA 02543

SEE ALSO

geosat(1)

NAME

g_date – prints the start and end date of GEOSAT GDR data

SYNOPSIS

g_date

DESCRIPTION

This program reads a binary GEOSAT file from *stdin* and prints the date and time of the first and last record in the file. The program also lists the Julian day, the year day and the day of the cycle.

AUTHORS

Ken Borowski
Mike Caruso
Woods Hole Oceanographic Institution
Woods Hole, MA 02543

SEE ALSO

geosat(l) g_date2(l)

NAME

g_date2 – prints the start and end date of GEOSAT GDR data

SYNOPSIS

g_date2 *cycle orbit*

DESCRIPTION

This program reads the cycle and orbit number from the command line and prints the date and time of the first and last record of that orbit. The program also lists the Julian day, the year day and the day of the orbit.

AUTHORS

Ken Borowski
Mike Caruso
Woods Hole Oceanographic Institution
Woods Hole, MA 02543

SEE ALSO

geosat(l) g_date(l)

NAME

g_ext – extracts GEOSAT GDR data and prints variables in ASCII

SYNOPSIS

g_ext *list*

DESCRIPTION

This program reads a binary GEOSAT file from *stdin* and extracts the specified variables on *stdout*. The variable *list* may be a combination of one or more of the following variables, in any order separated by spaces:

Variable	Description
t	time in seconds since START_TIME
l	latitude in degrees
L	east longitude in degrees
ho	orbit height above ellipsoid in m
ha	sea surface height above ellipsoid in m
sha	sigma ha
hg	geoid height above ellipsoid in m
w	significant wave height
sw	sigma w
so	backscatter coefficient in 0.01 dB
ag	agc in 0.01 dB
sag	sigma ag
fl	masked flags
HA	land surface height offset above ellipsoid
cet	correction for earth tide in m
cot	correction for ocean tide in m
cwf	correction for wet troposphere fnoc
cws	correction for wet troposphere smmr
cdf	correction for dry troposphere
ci	correction for ionosphere
ba	attitude bias
bc	compression bias
att	attitude
cib	correction for inverse barometric effect
h	corrected sea surface height above ellipsoid
dh	corrected sea surface height above geoid

A record will not be printed if any of the requested variables contains invalid data (32767).

AUTHORS

Pierre Flament
Oceanography Department
University of Hawaii
Honolulu, HI 96822

Mike Caruso

Woods Hole Oceanographic Institution
Woods Hole, MA 02543

SEE ALSO

geosat(l)

NAME

g_image – Generates a bitmap image of GEOSAT GDR data

SYNOPSIS

g_image *min_lat min_lon max_lat max_lon rows cols*

DESCRIPTION

This program reads an ASCII file with the latitude, longitude and *s* value on one line from *stdin*. The program writes an SDPS floating point file to *stdout*. For information on converting this to a byte image see sdps_ftb(l). The output file is an equirectangular image *rows* high by *columns* wide with coordinates from *min_lat* to *max_lat* and *min_lon* to *max_lon*.

AUTHORS

Mike Caruso
Woods Hole Oceanographic Institution
Woods Hole, MA 02543
Pierre Flament
Oceanography Department
University of Hawaii
Honolulu, HI 96822

REFERENCE

Caruso, M. and C. Dunn, Satellite Data Processing System (SDPS) Users Manual V1.0, Woods Hole Oceanog. Inst. Tech. Rept., WHOI-89-13, 1989

SEE ALSO

geosat(l) sdps(l) sdpsutil(l) sdps_ftb(l)

NAME

g_interp - linearly interpolates the GEOSAT GDR data

SYNOPSIS

g_interp *[dir min max deltmax timestep]*

DESCRIPTION

This program reads a binary GEOSAT file from *stdin* and breaks the data into continuous segments where a gap is defined by a gap of *deltmax* seconds between records. This program is used to regrid the GDRs to a consistent latitude-longitude grid. The algorithm was designed so that at least one point lies on the equator crossing and each successive point is *timestep* seconds from the previous point. The program will fill gaps with a calculated latitude and bad values (32767) for the longitude and height variables. Since not all orbits will be complete, the user also needs to specify a minimum and maximum latitude or longitude with *min* and *max* as well as the direction of the boundary with *dir*. If *dir* is 1, the record is filled between a minimum and maximum latitude and if *dir* is 2, the record is filled between the minimum and maximum longitude. The program assumes that corrections have been applied to the data previously and the data is free of abnormal values that appear as spikes.

AUTHOR

Mike Caruso
Woods Hole Oceanographic Institution
Woods Hole, MA 02543

SEE ALSO

geosat(l) g_correct(l) g_spike(l) g_spline(l)

NAME

g_print – Decodes GEOSAT GDR records in a lengthy format

SYNOPSIS

g_print

DESCRIPTION

This program takes GEOSAT GDR files and prints in a lengthy ASCII format. Each parameter is printed in its raw format with an identifier to *stdout*. Of special note is the data flags parameter. The flags are ordered as follows:

FEDCBA9876543210

where 0 is the zeroth flag bit and F is the fifteenth flag bit as listed in the GEOSAT Altimeter GDR User Handbook.

AUTHORS

Pierre Flament
University of Hawaii
Honolulu, HI 96822

Mike Caruso
Woods Hole Oceanographic Institution
Woods Hole, MA 02543

SEE ALSO

geosat(1)

NAME

g_region – Extracts a specified region from a GEOSAT GDR data set

SYNOPSIS

g_region *dir min_lat max_lat min_lon max_lon [orb#]*

DESCRIPTION

This program reads a binary GEOSAT file from *stdin*, extracts a specified region and separates the data into ascending and descending orbits that comply with the naming conventions described in geosat(l) manual page. This program finds the data that fall within the latitude-longitude box given by the parameters *min_lat, max_lat, min_lon, max_lon*. In order to maintain reasonable orbit lengths, the program will clip the orbits with either a constant latitude boundary, or a constant longitude boundary. Therefore, orbits that would normally be clipped in the corners, are extended to either the constant latitude or longitude boundary. The constant direction is chosen with the parameter *dir*. If *dir* is 1, the program extracts all data from that orbit that is between *min_lat* and *max_lat*. If *dir* is 2, the program extracts all data that is between *min_lon* and *max_lon*. If the optional orbit numbers are given, only those orbits that fall within the given box are extracted.

This program can be used to extract data directly from the NODC data tapes:

```
dd if=/dev/rmt8 ibs=16380 files=34 | g_region 1 10.0 30.0 280.0 300.0
```

BUGS

This program should only be used on complete orbits. It does not work on files that have already been extracted using g_region or g_seporb.

AUTHOR

Mike Caruso
Woods Hole Oceanographic Institution
Woods Hole, MA 02543

SEE ALSO

geosat(l) g_seporb(l)

NAME

g_repeat – performs a repeat analysis on GEOSAT GDR data

SYNOPSIS

g_repeat *c???.axxx*

DESCRIPTION

This program reads a binary GEOSAT file from each of the file names on the command line. The mean sea surface height is calculated for each point along the track and subtracted from each cycle. A quadratic polynomial is fit to the difference and subtracted from each cycle. The variance of the remainder is then used to calculate a new quadratic fit. This polynomial is then removed from the original sea surface height as an orbit error. The residual height for each cycle is written to a file with the record number, the latitude, the longitude, the residual heights along with the two quadratic polynomial. Also the statistics for each point are printed to standard output. This file contains the record number, the latitude, the longitude, the mean and variance of the corrected heights, the sum of the squared heights and the number of points used for the statistics of each record. All heights are given in meters. The program assumes that corrections have been applied to the data previously and the data has been splined to a uniform latitude-longitude grid.

AUTHOR

Mike Caruso
Woods Hole Oceanographic Institution
Woods Hole, MA 02543

SEE ALSO

geosat(l) g_correct(l) g_spline(l) g_repeats(l)

NAME

g_repeats – performs a repeat analysis on GEOSAT GDR data

SYNOPSIS

g_repeats *c???.axxx*

DESCRIPTION

This program reads a binary GEOSAT file from each of the file names on the command line. The mean sea surface height is calculated for each point along the track and subtracted from each cycle. A sinusoidal is fit to the difference and subtracted from each cycle. The variance of the remainder is then used to calculate a new sinusoidal fit. This polynomial is then removed from the original sea surface height as an orbit error. The residual height for each cycle is written to a file with the record number, the latitude, the longitude, the residual heights along with the two sine results. Also the statistics for each point are printed to standard output. This file contains the record number, the latitude, the longitude, the mean and variance of the corrected heights, the sum of the squared heights and the number of points used for the statistics of each record. All heights are given in meters. The program assumes that corrections have been applied to the data previously and the data has been splined to a uniform latitude-longitude grid.

AUTHOR

Mike Caruso
Woods Hole Oceanographic Institution
Woods Hole, MA 02543

SEE ALSO

geosat(l) g_correct(l) g_spline(l) g_repeat(l)

NAME

g_seporb – separates raw GEOSAT GDR data into ascending and descending orbits

SYNOPSIS

g_seporb

DESCRIPTION

This program reads a binary GEOSAT file from *stdin* and separates the data into ascending and descending orbits that comply with the naming conventions described in geosat(l) manual page. This program can be used to separate data directly from the NODC data tapes:

```
dd if=/dev/rmt8 ibs=16380 files=34 | g_seporb
```

BUGS

Input is expected to have at least two valid GDRs.

AUTHOR

Mike Caruso
Woods Hole Oceanographic Institution
Woods Hole, MA 02543

SEE ALSO

geosat(l) g_region(l)

NAME

g_spike – removes spikes from GEOSAT GDR data

SYNOPSIS

g_spike *[deltmax neighbors outlier]*

DESCRIPTION

This program reads a binary GEOSAT file from *stdin* and removes data records that appear as spikes in the sea surface height. The program assumes that corrections have been applied to the data previously. *Deltmax* is the amount of time that constitutes a gap between continuous segments (default is 3.3 seconds), *neighbors* is the number of points to use for least squares fit to a quadratic polynomial (default is 13 points) and *outlier* is the maximum acceptable deviation from the least squares fit (default is 0.20 meters)

AUTHOR

Mike Caruso
Woods Hole Oceanographic Institution
Woods Hole, MA 02543

SEE ALSO

geosat(l) g_correct(l)

NAME

g_spline – splines the GEOSAT GDR data

SYNOPSIS

g_spline *[dir min max deltmax timestep]*

DESCRIPTION

This program reads a binary GEOSAT file from *stdin* and breaks the data into continuous segments where a gap is defined by *deltmax* seconds between records. This program is used to regrid the GDRs to a consistent latitude-longitude grid. The algorithm was designed so that at least one point lies on the equator crossing and each successive point is *timestep* seconds from the previous point. The program will fill gaps with a calculated latitude and bad values (32767) for the longitude and height variables. Since not all orbits will be complete, the user also needs to specify a minimum and maximum latitude or longitude with *min* and *max* and the direction of the boundary with *dir*. If *dir* is 1, the record is filled between a minimum and maximum latitude and if *dir* is 2, the record is filled between the minimum and maximum longitude. The program assumes that corrections have been applied to the data previously and the data is free of abnormal values that appear as spikes.

AUTHOR

Mike Caruso
Woods Hole Oceanographic Institution
Woods Hole, MA 02543

SEE ALSO

geosat(l) g_correct(l) g_spike(l) g_interp(l)

NAME

g_which – Prints GEOSAT orbit numbers from a specified lat/lon box

SYNOPSIS

g_which ***min_lat max_lat min_lon max_lon***

DESCRIPTION

This program reads the minimum and maximum latitudes and minimum and maximum longitudes from the command line and prints the orbit numbers contained in the box. The orbits are printed within curly braces, separated by commas and may be used in a set of pipes.

cat c000.'g_which 30 45 280 300' | g_ext 1 L

If only two arguments are given to the program, they are assumed to be a point and the nearest ascending and descending orbits are given.

AUTHORS

Mike Caruso
Woods Hole Oceanographic Institution
Woods Hole, MA 02543

Pierre Flament
Oceanography Department
University of Hawaii
Honolulu, HI 96822

SEE ALSO

geosat(1)

NAME

s_ext – extracts SSMI data and prints variables in ASCII

SYNOPSIS

s_ext *list*

DESCRIPTION

This program reads a binary SSMI file from *stdin* and extracts the specified variables on *stdout*. *List* may be a combination of one or more of the following variables, in any order separated by spaces:

Variable	Description
t	time in seconds since START_TIME
l	latitude in degrees
L	east longitude in degrees
fl	flag indicating data characteristics 0 - Over ocean 1 - No orbit altitude information 2 - Over land 3 - Over sea ice
ws	Wind Speed
vp	columnar water vapor
cl	columnar cloud water
rn	rain rate
cws	SSMI correction for water vapor

A record will not be printed if any of the requested variables contains invalid data (32767).

AUTHORS

Pierre Flament
Mimi Baker
Oceanography Department
University of Hawaii
Honolulu, HI 96822

SEE ALSO

geosat(l)

NAME

s_region – Extracts a specified region from a SSMI data set

SYNOPSIS

s_region *dir min_lat max_lat min_lon max_lon*

DESCRIPTION

This program reads a binary SSMI file from *stdin*, extracts a specified region and separates the data into ascending and descending orbits that comply with the naming conventions described in geosat(l) manual page. This program finds the data that fall within the latitude-longitude box given by the parameters *min_lat max_lat min_lon max_lon* In order to maintain reasonable orbit lengths, the program will clip the orbits with either a constant latitude boundary, or a constant longitude boundary. Therefore, orbits that would normally be clipped in the corners, are extended to either the constant latitude or longitude boundary. The constant direction is chosen with the parameter *dir*dir. If *dir* is 1, the program extracts all data from that orbit that is between *min_lat* and *max_lat*. If *dir* is 2, the program extracts all data that is between *min_lon* and *max_lon*.

This program can be used to extract data directly from the data tapes:

dd if=/dev/rmt8 ibs=14400 | s_region 1 10.0 30.0 280.0 300.0

AUTHORS

Mimi Baker
Oceanography Department
University of Hawaii
Honolulu, HI 96822

Mike Caruso
Woods Hole Oceanographic Institution
Woods Hole, MA 02543

SEE ALSO

geosat(l) g_region(l) g_seporb(l)

B Program Listings

This section contains listing of all programs and subroutines discussed in sections 5.1, 5.2 and 5.3. Programs are listed alphabetically and subroutines follow the programs.

Program g_clean1.c

```
/*
  @(#)g_clean1.c        1.5     6/13/90
  Program g_clean1.c

  Written by:
  --------------
  Michael Caruso
  Woods Hole Oceanograhic Institution
  Woods Hole, MA

  Purpose:
  --------
  This is the first step in cleaning up GEOSAT data.
  This program will remove bad data points as specified
  by the shell variable GMASK (See users manual) as well
  as points with sigma height > 30 cm and sigma naught
  > 35 Db.

  Method:
  -------
  Reads raw GEOSAT GDRs from standard input and deletes any
  records that are not within specified parameters.  Output
  is in GDR format.

  Usage:
  ------
  The GEOSAT GDR is read from standard input.

  cat c000.a002 | g_clean1 > c000.a002c

  will remove all bad records from the GDR in c000.a002.


  Input:
  ------
  Stdin          Raw GEOSAT GDRs

  Output:
  -------
  Stdout:        Clean GEOSAT GDRs

  Subroutines Required:
  ---------------------
  geo_error      Prints errors to standard error
  geo_mask       Gets mask variable GMASK

  References:
  -----------
```

```
*/
# include <math.h>
# include <stdio.h>
# include <string.h>
# include "geos.h"

#define SHBAD   30.0            /* Value for bad sigma height */
#define SNBAD   3500.0          /* Value for bad sigma naught */
#define CHECKFL 12415           /* Used to get rid of check sum flags */

int i,j;
int bad;
int fl_bad;
short int msk, valid;

union
{
  struct flags  fl;
  short int     flagint;
} cl_flags;                     /* To allow bit operations on flags */


main (argc,argv)
     int argc;
     char *argv[];

{

/*
  Set counters to zero...
*/

  int   good_count     = 0;
  int   m_h_count      = 0;
  int   s_tide_count   = 0;
  int   o_tide_count   = 0;
  int   w_fnoc_count   = 0;
  int   d_fnoc_count   = 0;
  int   iono_count     = 0;
  int   s_h_count      = 0;
  int   s_nght_count   = 0;
  int   flag_count     = 0;
  int   tot_count      = 0;

/*
  Check for arguments...
*/

  if (argc != 1)
    {
      fprintf(stderr,"Usage: %s < filein > filout\n",argv[0]);
      exit(1);
    }
```

```
/* get from the environment which bits of the flags should be masked
     and which values constitute a valid frame */

  geo_mask(&msk,&valid);

/*
  Loop over all points in input file...
*/

  while (fread((char*)&fr,1,REC_LEN,stdin)==REC_LEN)
    {

      /* get rid of checksum flags  0011000001111111      */

      cl_flags.fl = fr.fl;

      cl_flags.flagint &= CHECKFL;

      /* set bad to false if mask of the data flags */
      /* is not equal to valid mask                 */

      bad = (cl_flags.flagint & msk) != valid;
      if(bad) fl_bad = BAD;

      /*
        Check all records for any bad data.  Don't check Time,
        lat or lon.  Reject if ha at that point is bad, or if cet,
        cot, cwf, cdf or ci is bad.
        */

      bad = bad
        || (fr.m_h         == BAD
            || fr.s_tide == BAD
            || fr.o_tide == BAD
            || fr.w_fnoc == BAD
            || fr.d_fnoc == BAD
            || fr.iono   == BAD
            || fr.s_h    > SHBAD
            || fr.s_nght > SNBAD);


      if (bad)
        {
          if (fr.m_h == BAD) m_h_count += 1;
          if (fr.s_tide == BAD) s_tide_count += 1;
          if (fr.o_tide == BAD) o_tide_count += 1;
          if (fr.w_fnoc == BAD) w_fnoc_count += 1;
          if (fr.d_fnoc == BAD) d_fnoc_count += 1;
          if (fr.iono == BAD) iono_count += 1;
          if (fr.s_h > SHBAD) s_h_count += 1;
          if (fr.s_nght > SNBAD) s_nght_count += 1;
          if (fl_bad == BAD)    flag_count += 1;
          tot_count += 1;
```

```
        }
      fl_bad = 0;
      if (bad) continue;        /* If bad skip print and get next point. */

      /*
        Write out good data records.
        */

      if (fwrite((char *)&fr, 1, REC_LEN, stdout) != REC_LEN)
        {
          geo_error(3,argv[0]);
          exit(3);
        }
      good_count += 1;

    }
  /*
    Write out statistics on rejected points to standard output...
  */

    fprintf(stderr,"%s: Valid points:\t%8d\n",argv[0],good_count);
    fprintf(stderr,"\tRejected points:%8d\n",tot_count);
    fprintf(stderr,"\tHeight:\t\t%8d\n",m_h_count);
    fprintf(stderr,"\tSolid Tide:\t%8d\n",s_tide_count);
    fprintf(stderr,"\tOcean Tide:\t%8d\n",o_tide_count);
    fprintf(stderr,"\tWet FNOC:\t%8d\n", w_fnoc_count);
    fprintf(stderr,"\tDry FNOC:\t%8d\n", d_fnoc_count);
    fprintf(stderr,"\tIono:\t\t%8d\n", iono_count);
    fprintf(stderr,"\tSigma Height:\t%8d\n", s_h_count);
    fprintf(stderr,"\tSigma Naught:\t%8d\n", s_nght_count);
    fprintf(stderr,"\tFlags:\t\t%8d\n", flag_count);

  }
```

Program g_clean2.c

```
/*
  @(#)g_clean2.c        1.3 12/15/89

  Program g_clean2.c

  Written by:
  --------------
  Michael Caruso
  Woods Hole Oceanograhic Institution
  Woods Hole, MA

  Modifications:
  --------------
  12-15-89  MC now also prints total number of points rejected.

  Purpose:
  --------
  This program will clean a GEOSAT GDR by removing
  records with heights greater than 10,000 cm and less
  than -14,000 cm.

  Method:
  -------
  Reads raw GEOSAT GDRs from standard input and checks
  height.  Does not check for validity of input and assumes
  input data has had corrections applied . Output is in
  GDR format.

  Usage:
  ------
  The GEOSAT GDR is read from standard input.

  cat c000.a002 | g_clean2 > c000.a002c

  will clean all bad data records from the GDR in c000.a002.


  Input:
  ------
  Stdin         Raw GEOSAT GDRs

  Output:
  -------
  Stdout:       Corrects GEOSAT GDRs

  Subroutines Required:
  ---------------------
  geo_error     Prints error messages.
```

```
  References:
  -----------

*/
# include <math.h>
# include <stdio.h>
# include <string.h>
# include "geos.h"

#define MIN_HEIGHT -14000
#define MAX_HEIGHT  10000

main (argc,argv)
     int argc;
     char *argv[];
{

  int min_count = 0;
  int max_count = 0;
  int tot_count = 0;

  if (argc != 1)
    {
      fprintf(stderr,"Usage: %s < filein > fileout\n",argv[0]);
      exit(1);
    }



  while (fread((char*)&fr,1,REC_LEN,stdin)==REC_LEN)
    {
      /*
        Check GDR heights here.
        */

      if ((fr.m_h < MAX_HEIGHT) || (fr.m_h > MIN_HEIGHT))
        {
          /*
            Write out good data records.
            */

          if (fwrite((char *)&fr, 1, REC_LEN, stdout) != REC_LEN)
            {
              geo_error(3,argv[0]);
              exit(3);
            }
        }

      else
        {
          if (fr.m_h > MAX_HEIGHT) max_count += 1;
          else if (fr.m_h < MIN_HEIGHT) min_count += 1;

          if ((fr.m_h > MAX_HEIGHT) || (fr.m_h < MIN_HEIGHT)) tot_count += 1;
```

```
        }

    }
  /*
    Print rejection numbers...
    */
  fprintf(stderr,"%s: Rejected points:%6d\n", argv[0], tot_count);
  fprintf(stderr,"\tMaximum Height:\t%8d\n", max_count);
  fprintf(stderr,"\tMinimum Height:\t%8d\n\n", min_count);

}
```

Program g_compress.c

```
/*
  @(#)g_compress.c      1.2 6/13/90

  Written by:
  -----------
  Pierre Flament
  Oceanography Department
  University of Hawaii
  Honolulu, HI

  Modifications:
  --------------
  Mike Caruso
  Woods Hole Oceanographic Institution
  Woods Hole, MA

  Purpose:
  --------

  compress geosat data into 18 bytes/frame

  item  parameter               units    range          type

  1     TIME since start a000   ms       0 to 1.47e9    long int (4)
  2     HEIGHT                  cm       0 to 32766     short int (2)
  3     CYCLE number                     0...           char (1)
  4     LATITUDE                10-4deg  0 to 18e5      unsigned int (3)
  5     LONGITUDE               10-4deg  0 to 36e5      unsigned int (3)
  6     SIGMA HEIGHT            cm       0 to 255       unsigned char (1)
  7     SWH                     5cm      0 to 255       unsigned char (1)
  8     So                      .1db     0 to 255       unsigned char (1)
  9     FLAGS                                           char (1)
  10    OCEAN TIDE              cm       -128 to 128    char (1)

  Method:
  ------
  Reads in GDR, converts to 18 byte GDR and writes to
  standard output

  Usage:
  -----
  g_compress < file.gdr > file.18b

  Input:
  -----
  GEOSAT GDR

  Output:
  -------
```

```
   18-byte data record

   Subroutines required:
   ---------------------

   References:
   -----------

*/
# include <stdio.h>

# define PERIOD 6037.551518571
# define START_TIME 58406188.43 /* equator xing orbit c000.a000 */
# define BAD 32767

/* this is the standard geosat frame */

struct  in_frame {
        long int utc,utcm,lat,lon,orb;
        short int m_h,s_h,geoid,h[10],swh,s_swh,s_nght,agc,s_agc;
        char fl[2];
        short int h_off,s_tide,o_tide,w_fnoc,w_smmr,d_fnoc,iono,
                  dh_swh,dh_fm,att;
        };

struct in_frame in;

/* this is the compressed frame. Order is important since compiler
forces short int on even word boundaries */

struct  out_frame {
        long int utc;
        short int m_h;
        char cycle_n;
        char lat[3],lon[3];
        unsigned char s_h,swh,s_nght;
        char fl;
        char o_tide;
        };

struct out_frame out;

double time,cycle=244*PERIOD;


int i,j;

struct flags *f;

main()

{
while(fread((char*)&in,1,78,stdin)==78)
        {
```

```
        out.cycle_n=0;
        time = in.utc - START_TIME;

        while(time >cycle)
                {
                out.cycle_n++;
                time -= cycle;
                }

        out.utc = nint(time*1000. + in.utcm/1000.);

        in.lat = nint(in.lat/100.);
        in.lat += 900000;
        out.lat[0]=*((char*)&in.lat+1);
        out.lat[1]=*((char*)&in.lat+2);
        out.lat[2]=*((char*)&in.lat+3);

        in.lon = nint(in.lon/100.);
        out.lon[0]=*((char*)&in.lon+1);
        out.lon[1]=*((char*)&in.lon+2);
        out.lon[2]=*((char*)&in.lon+3);

        out.m_h=in.m_h;

        out.s_h=(in.s_h>255?255:(unsigned char)in.s_h);

        in.swh = nint(in.swh/5.);
        out.swh=(in.swh>255?255:(unsigned char)in.swh);

        in.s_nght = nint(in.s_nght/10.);
        out.s_nght=(in.s_nght>255?255:(unsigned char)in.s_nght);

        out.fl=in.fl[1];

        in.o_tide = nint(in.o_tide/10.);
        out.o_tide=(abs(in.o_tide)>127?127:(char)in.o_tide);
        fwrite((char*)&out,1,18,stdout);
        }
}
```

Program g_correct.c

```
/*
  @(#)g_correct.c        1.2 6/13/90

  Program g_correct.c

  Written by:
  --------------
  Michael Caruso
  Woods Hole Oceanograhic Institution
  Woods Hole, MA

  Added comments and cleaned up some code.

  Purpose:
  --------
  This program will apply corrections to a GEOSAT
  GDR as specified in the GEOSAT users manual.  The following
  correction will be applied:

  h     =  h    - solid tide
                - ocean tide
                - wet tropospheric correction (fnoc)
                - dry tropospheric correction (fnoc)
                - ionosphere correction
                - inverse barometer effect

  Where
        inverse barometer effect = -9.948 * (p - 1013.3)
  And
        p = dry (fnoc) / (-2.277)(1 + (0.0026 * cos(2 * latitude)))

  Method:
  -------
  Reads raw GEOSAT GDRs from standard input and applies
  corrections.  Does not check for validity of input. Output
  is in GDR format.

  Usage:
  ------
  The GEOSAT GDR is read from standard input.

  cat c000.a002 | g_correct > c000.a002c

  will apply corrections to all records from the GDR in c000.a002.


  Input:
  ------
  Stdin          Raw GEOSAT GDRs
```

```
  Output:
  -------
  Stdout:       Corrects GEOSAT GDRs

  Subroutines Required:
  ---------------------
  bar.c         Included, Calculates the inverse barometer effect.

  References:
  -----------

*/
# include <math.h>
# include <stdio.h>
# include <string.h>
# include "../../include/geos.h"
# include "../../include/g_ext.h"

#define NUMARGS 6

int i,j,iarg;
main (argc,argv)
     int argc;
     char *argv[];
{


  float bar();
  float corr;

  if ((argc < 1) && (argc > NUMARGS+1))
    {
      fprintf(stderr,"Usage: %s [cet cot cwf cdf ci cib] < filein > fileout\n",
                      argv[0]);
      exit(1);
    }


  while (fread((char*)&fr,1,REC_LEN,stdin)==REC_LEN)
    {
      /*
        Apply corrections here.
        */
      if(argc == 1)             /* default apply all corrections */
        {
          fr.m_h = fr.m_h - nint((fr.s_tide + fr.o_tide + fr.w_fnoc
                                  + fr.d_fnoc + fr.iono
                                  + bar(fr.d_fnoc, fr.lat))/10.0);
        }
      else
        {
          corr = 0.0;
```

```
        for (iarg=1; iarg<argc; iarg++)
          {
            if(!strcmp(argv[iarg],val[25]))
              {
                corr += fr.s_tide;
              }
            else if (!strcmp(argv[iarg],val[26]))
              {
                corr += fr.o_tide;
              }
            else if (!strcmp(argv[iarg],val[27]))
              {
                corr += fr.w_fnoc;
              }
            else if (!strcmp(argv[iarg],val[29]))
              {
                corr += fr.d_fnoc;
              }
            else if (!strcmp(argv[iarg],val[30]))
              {
                corr += fr.iono;
              }
            else if (!strcmp(argv[iarg],val[34]))
              {
                corr += bar(fr.d_fnoc, fr.lat);
              }
            else
              {
                fprintf(stderr,"%s: illegal option %s ignored.\n",argv[0],
                        argv[iarg]);
              }

          }
      }

    fr.m_h = fr.m_h - nint(corr/1C.0);

    /*
      Write out good data records.
      */

    if (fwrite((char *)&fr, 1, REC_LEN, stdout) != REC_LEN)
      {
        geo_error(3,argv[0]);
        exit(3);
      }
  }
}


float bar(d_fnoc, lat)
     short d_fnoc;
     long       lat;
```

```
{
  float p;

  p = (d_fnoc / ((-2.277)*(1 + (0.0026 * cos(2*M_PI*lat*1.e-6/180.)))));

  p = -9.948 * (p - 1013.3);

  return(p);
}
```

Program g_crossnum.c

```
/*
 @(#)g_crossnum.c      1.3 6/14/90

 Program g_crossnum.c

 Written by:
 -----------
 Michael Caruso
 Woods Hole Oceanographic Institution
 Woods Hole, MA

 Purpose:
 --------
 To print out the longitude of the equator crossing of
 a given orbit.

 Method:
 -------
 Reads the orbit number from the command line or reads a
 GDR from standard input. Calculates the equator crossing
 and prints the result.

 Usage:
 ------
 g_crossnum a002

 or

 g_crossnum < file.gdr

 Input:
 ------
 Stdin         GDR file

 Output:
 ------
 Stdout        Longitude of orbit crossing

 Assumptions:
 ------------
   Longitude is given as E positive from 0 to 360 degrees.

 Subroutines Required:
 ---------------------
 geo_cyc_orb.c          Determines cyc and orb from GDR
 orb_cross.c            Determines where a particular orbit crosses
                        the equator.

 References:
```

```
  -----------

*/


#include <stdio.h>
#include <math.h>
#include <string.h>
#include "geos.h"

#define C_DEG   360.0/244.0     /* Degrees between successive geosat crossings */

main(argc,argv)
int argc;
char *argv[];
{

  float orb_cross();            /* Determines long. of equator crossing */

  char str[10];                 /* Strings to parse input */
  char *s;

  int i, j;                     /* Counters */
  int orb_num;                  /* Orbit number */
  int asc;                      /* True if asc orbit */
  int cyc;                      /* Cycle number */

  struct frame fr2;             /* GEOSAT GDR */

  float crossing;               /* Equator crossing in degrees */

/*
  Read command line arguments...
*/

  if (argc > 2)
    {
      fprintf(stderr,"Usage: %s orbit\n",argv[0]);
      fprintf(stderr,"Or\n");
      fprintf(stderr,"%s < file.geo\n", argv[0]);
      exit(1);
    }

  if (argc == 2)        /* Read orbit number from command line. */
    {
      strncpy(str,argv[1]+1,3);
      sscanf(str,"%d",&orb_num);
      s = argv[1];
      switch(*s)
        {
        case 'a':
          asc = TRUE;
          break;
```

```
        case 'd':
          asc = FALSE;
          break;
        default:
          fprintf("%s: Illegal argument: %s\n",argv[0], argv[1]);
          exit(1);
        }

    }
  else          /* Read GDR from standard input */
    {
      if(fread((char *)&fr, 1, REC_LEN, stdin) != REC_LEN)
        {
          geo_error(3, argv[0]);
          exit(1);
        }
      if(fread((char *)&fr2, 1, REC_LEN, stdin) != REC_LEN)
        {
          geo_error(3, argv[0]);
          exit(1);
        }
      if ((fr2.lat - fr.lat) > 0)      /* Ascending orbit */
        {
          asc = TRUE;
        }
      else
        {
          asc = FALSE;
        }
      geo_cyc_orb(fr, &cyc, &orb_num); /* Get orbit number */
    }

  crossing = orb_cross(orb_num, asc);   /* Get crossing */

  fprintf(stdout,"%f\n", crossing);

}


float orb_cross(orb_num, asc)
     int orb_num, asc;
{
  float lon;

  if (asc)
    {
      lon = 356.59 - orb_num*360.0*17./244.;
    }
  else
    {
      lon = 189.14 - orb_num*360.0*17./244.;
    }
  while (lon < 0.)
    {
```

```
      lon += 360.;
    }
  return (lon);
}
```

Program g_date.c

```
/*
  @(#)g_date.c  1.3 6/14/90

  Written by:
  -----------
  Kenneth Borowski
  Woods Hole Oceanographic Institution
  Woods Hole, MA 02543

  Modifications
  -------------
  Mike Caruso
  Woods Hole Oceanographic Institution
  Woods Hole, MA

  Added Comments and restructed code to use more consistent
  naming convention.

  Purpose:
  --------
  Decode geosat data and convert Universal Time Coordinates to
  month, day, year, Julian day, and day of cycle for the first
  and last record in a file

  Method:
  -------

  Usage:
  ------
  cat c000.a002 | g_date        Gives start and end date of c000.a002

  cat c000.*    | g_date        Gives start and end date of all orbits
                                in cycle c000

  Input:
  ------
  Geosat GDR

  Output:
  -------
  Start and end time of GDR

  Subroutines Required:
  ---------------------
  output                Outputs start and end times
  julday                Calculates julian day
  kdate                 Converts to month day and year

*/
```

```
# include <stdio.h>
# include <math.h>
# include "geos.h"

long int utcs, utcm;

main()

{
    fread((char*)&fr,1,REC_LEN,stdin);
    output(fr.utc,fr.utcm);
    while(fread((char*)&fr,1,REC_LEN,stdin) == REC_LEN) {
        utcs = fr.utc;
        utcm = fr.utcm;
    }
    output(utcs,utcm);
}


output(utcs,utcm)
long int utcs, utcm;
{
    int days,y,m,d;
    int seconds,minutes,hours;
    int orbit_num_tot;
    int cycle_num;
    int orbit_num;
    int day_of_year;

    double time;

    days = utcs / 86400;
    seconds = utcs % 86400;
    hours = seconds / 3600;
    minutes = (seconds % 3600) / 60;
    seconds = (seconds % 3600) % 60;
    kdate(days,&y,&m,&d);
    time = utcs +  utcm / 1.0e6;
    orbit_num_tot = (int)floor((time-TIME_ZERO)/PERIOD);
    orbit_num = orbit_num_tot % 244;
    cycle_num = orbit_num * 17 / 244;
    day_of_year = julday(y,m,d) - julday(y-1,12,31);
    printf("\n     UTC: %11.2f\n", time);
    printf("     Date: %d/%d/%d %d:%.2d:%.2d\n", m,d,y,hours,minutes,seconds);
    printf("     Day of year: %d\n", day_of_year);
    printf("     Julian: %d\n", julday(y,m,d));
    printf("     Day of cycle: %d\n\n",cycle_num);
}
/* URI Julian day algorithm */
julday(y,m,d)
int y,m,d;{
return(367*y -7*(y + (m+9)/12)/4 - 3*((y + (m-9)/7)/100 +1)/4
        + 275*m/9 + d + 1721029);
```

```
        }
kdate(k,y,m,d)
/* converts the day, k, to month, day and year */
/* assumes that k = 1 corresponds to Jan 1, 1985 */
int k,*y,*m,*d;
{
    k = k + 30987;
    *y = (4 * k - 1) / 1461;
    *d = 4 * k - 1 - 1461 * (*y);
    *d = (*d + 4) / 4;
    *m = (5 * (*d) - 3) / 153;
    *d = 5 * (*d) - 3 - 153 * (*m);
    *d = (*d + 5) / 5;
    if (*m < 10)
        *m = *m + 3;
    else {
        *m = *m - 9;
        *y = *y + 1;
    }
}
```

Program g_date2.c

```
/*
  @(#)g_date2.c 1.3 6/14/90

  Written by:
  -----------
  Kenneth Borowski
  Woods Hole Oceanographic Institution
  Woods Hole, MA 02543

  Modified by:
  -----------
  Mike Caruso
  Woods Hole Oceanographic Institution
  Woods Hole, MA 02543

  Originally began as g_date.c and converted so that the input is
  a cycle number and an orbit number.

  Purpose:
  --------
  Read a cycle and orbit number and determine the Universal Time
  Coordinates of the beginning and end of that orbit and convert to
  month, day, year, Julian day, and day of cycle for the first
  and last record in a file.

  Usage:
  ------
  g_date2 000 192

  Input:
  ------
  cycle number
  orbit number

  Output:
  -------
  Start and end time of GDR in various formats

  Subroutines Required:
  ---------------------
  output        Outputs start and end times
  julday        Calculates Julian Day
  kdate         Converts to month, day and year
  geo_rcyc_orb  Returns time for a given cycle and orbit

*/

#include <stdio.h>
#include <math.h>
```

```
#include "geos.h"

#define NUMARGS 2       /* Number of command line args */
#define CYARG   1       /* Number of cycle arg. */
#define ORARG   2       /* Number of orbit arg. */

long int utcs, utcm;    /* Universal time variables secs, microsecs */

main(argc, argv)
     int argc;
     char *argv[];
{
  double time;
  int cycle, orbit;

  if (argc != NUMARGS+1)
    {
      fprintf(stderr,"Usage: %s cycle orbit\n", argv[0]);
      exit(1);
    }

  sscanf(argv[CYARG],"%d", &cycle);
  sscanf(argv[ORARG],"%d", &orbit);

  geo_rcyc_orb(&time, cycle, orbit);
  output((int)time,0);
  geo_rcyc_orb(&time, cycle, orbit+1);
  output((int)time,0);

}


output(utcs,utcm)
long int utcs, utcm;
{
    int days,y,m,d;
    int seconds,minutes,hours;
    int orbit_num_tot;
    int cycle_num;
    int orbit_num;
    int day_of_year;

    double time;

    days = utcs / 86400;
    seconds = utcs % 86400;
    hours = seconds / 3600;
    minutes = (seconds % 3600) / 60;
    seconds = (seconds % 3600) % 60;
    kdate(days,&y,&m,&d);
    time = utcs +  utcm / 1.0e6;
    orbit_num_tot = (int)floor((time-TIME_ZERO)/PERIOD);
    orbit_num = orbit_num_tot % 244;
    cycle_num = orbit_num * 17 / 244;
```

```
    day_of_year = julday(y,m,d) - julday(y-1,12,31);
    printf("\n     UTC: %11.2f\n", time);
    printf("     Date: %d/%d/%d %d:%.2d:%.2d\n", m,d,y,hours,minutes,seconds);
    printf("     Day of year: %d\n", day_of_year);
    printf("     Julian: %d\n", julday(y,m,d));
    printf("     Day of cycle: %d\n\n",cycle_num);
}
/* URI Julian day algorithm */
julday(y,m,d)
int y,m,d;{
return(367*y -7*(y + (m+9)/12)/4 - 3*((y + (m-9)/7)/100 +1)/4
        + 275*m/9 + d + 1721029);
        }
kdate(k,y,m,d)
/* converts the day, k, to month, day and year */
/* assumes that k = 1 corresponds to Jan 1, 1985 */
int k,*y,*m,*d;
{
    k = k + 30987;
    *y = (4 * k - 1) / 1461;
    *d = 4 * k - 1 - 1461 * (*y);
    *d = (*d + 4) / 4;
    *m = (5 * (*d) - 3) / 153;
    *d = 5 * (*d) - 3 - 153 * (*m);
    *d = (*d + 5) / 5;
    if (*m < 10)
        *m = *m + 3;
    else {
        *m = *m - 9;
        *y = *y + 1;
    }
}


/*
  Subroutine geo_rcyc_orb.c
*/

int geo_rcyc_orb(time, cyc, orb)
     double *time;
     int cyc;
     int orb;
{
  int orbit;

  orbit = cyc * ORB_PER_CYC + orb;


  *time = orbit*PERIOD + TIME_ZERO;
}
```

Program g_ext.c

```
/*
  @(#)g_ext.c   1.4 4/25/90
  Program g_ext.c

  Written by:
  -----------
  Pierre Flament
  University of Hawaii
  Honolulu, HI  96822

  Modifications:
  --------------
  Michael Caruso
  Woods Hole Oceanograhic Institution
  Woods Hole, MA

  Added comments and cleaned up some code.

  Purpose:
  --------
  To extract user specified data from a GEOSAT GDR.

  Method:
  -------
  Reads raw GEOSAT GDRs from standard input and applies
  corrections. Reads user desired output variables from
  command line arguments.  Write output on standard output.
  Output is in ASCII format.

  Usage:
  ------
  The GEOSAT GDR is read from standard input and output variables
  are read from command line.

  cat c000.a002 | g_ext t l L > file.asc

  will extract the time, the latitude and the longitude for each
  good point in the file c000.a002.

  cat c000.a002 | gext l L ha > file.asc

  will extract the latitude, the longitude and the sea surface height
  above the ellipsoid.

  Input:
  ------
  Stdin         Raw GEOSAT GDRs

  Output:
```

```
  -------
  Stdout:       Extracted data in ASCII format

  Subroutines Required:
  --------------------

  References:
  -----------

*/
# include <math.h>
# include <stdio.h>
# include <string.h>
# include "geos.h"
# include "g_ext.h"

# define MXP    26              /* max number of parameters */

# define PRINT(X) printf(form[col[i]],X)



int i,j;
int col[MXP];
int bad;
short int msk,valid;

char * getenv();
double h(),bar();

/* geosat data frame.  Use this instead of fr to get two arrays, one
 long int and one short int.  Need to be careful with the indices. */

struct  {
        long int x[5];
        short int y[NCHAN-5];
        } v;

main (argc,argv)
     int argc;
     char *argv[];

{
  for(j=0;j<MXP;j++)
    col[i] = -1;

  argc--;
  argv++;

  if (argc==0)
    {
      fprintf(stderr,"gext: argument error\n");
      exit(1);
    }
```

```
  /* find which channels should be processed
     i: argument/column index
     j: channel number
     col[i]: channel number corresponding to column i
     */

  for (i=0;i<argc;i++)
    for (j=0;j<NCHAN+3;j++)
      if(!strcmp(argv[i],val[j]))
        {
          col[i]=j;
          break;
        }

  /* get from the environment which bits of the flags should be masked
     and which values constitute a valid frame */

  geo_mask(&msk,&valid);

  while (fread((char*)&v,1,78,stdin)==78)
    {

      if (bad) continue;        /* If bad skip print and get next point. */

      for (i=0;i<argc;i++)      /* Print requested data */
        if (col[i]==0)
          PRINT(v.x[0]*conv[0]+v.x[1]*conv[1]-START_TIME);
        else if (col[i]<5)
          PRINT(v.x[col[i]]*conv[col[i]]);
        else if (col[i]==5)
          PRINT(v.y[0]*conv[5]+v.y[19]*conv[24]);
        else if (col[i]<NCHAN)
          PRINT(v.y[col[i]-5]*conv[col[i]]);
        else if (col[i]==NCHAN)
          PRINT(bar());
        else if (col[i]==NCHAN+1)
          PRINT(h());
        else if (col[i]==NCHAN+2)
          PRINT(h()-v.y[2]*conv[7]);
      printf("\n");
    }
}

double h()
/* compute corrected height based on suggested
 corrections in the GEOSAT altimeter GDR user handbook
 */
{
double x;

x=v.y[0]*conv[5]+v.y[19]*conv[24]
        -v.y[20]*conv[25]
```

```
        -v.y[21]*conv[26]
        -v.y[22]*conv[27]
        -v.y[24]*conv[29]
        -v.y[25]*conv[30];
return(x-bar());
}

double bar()
/* inverse barometric effect in m using the formula
 provided in the GEOSAT altimeter GDR user handbook
 */
{
double p;

p= v.y[24]/(-2.277*(1+0.0026*cos(2*M_PI*v.x[2]*conv[2]/180.)));
return(-9.948*(p-1013.3)*1.e-3);
}
```

Program g_image.c

```
/*
 @(#)g_image.c 1.4 6/14/90

 Written by:
 ----------
 Pierre Flament
 University of Hawaii
 Honolulu, HI  96822


 Modified by:
 ------------
 Michael Caruso
 Woods Hole Oceanographic Institution
 Woods Hole, MA

 Changed to allow different output sizes and to
 output sdps floating point format.

 Purpose:
 --------
 To generate bitmap images of GEOSAT data.

 Method:
 -------
 Reads ascii files of lat, lon, z and converts to
 a bitmap image in sdps floating point format.

 Usage:
 ------
 g_image 20 40 190 210 512 512 < file.asc > file.sdpsf

 Input:
 ------
 Stdin All lat, lon and z values.

 Output:
 -------
 Stdout        Image in SDPS floating point format

 Assumptions:
 ------------
 Longitude is E positive from 0 to 360 degrees.

 Subroutines Required:
 ---------------------
 write_sdps    Writes out an SDPS format file.

 References:
```

```
  ----------
  Satellite Data Processing System (SDPS) Users Manual V1.0,
  Michael Caruso and Chris Dunn, Woods Hole Oceanog. Inst. Tech. Rept.,
  WHOI-89-13

*/

#include <stdio.h>
#include <math.h>
#include "sdpsutil.h"

/*
  Define argument places here.
*/
#define NUMARGS 6
#define MNLAT   1
#define MXLAT   2
#define MNLON   3
#define MXLON   4
#define ROW     5
#define COL     6

#define S_DEG 20        /* number of GEOSAT samples per degree*/

main(argc,argv)
int argc;
char *argv[];
{

  double min_lat, min_lon, max_lat, max_lon;
  double lat, lon, z;
  int i, j, dlat, dlon, ilat, ilon;
  int row, col, rowoff, ij;

  struct sdpsheader     header;
  struct sdpscmap       cmap;
  float                 *im;

/*
  Read command line arguments...
*/

  if (argc != NUMARGS+1)
    {
      fprintf(stderr,
      "Usage: g_image  min_lat, max_lat, min_lon, max_lon row col\n");
      exit(1);
    }

  sscanf(argv[MNLAT],"%lf",&min_lat);
  sscanf(argv[MXLAT],"%lf",&max_lat);
  sscanf(argv[MNLON],"%lf",&min_lon);
  sscanf(argv[MXLON],"%lf",&max_lon);
  sscanf(argv[ROW],"%d",&row);
```

```
  sscanf(argv[COL],"%d",&col);
/*
  Allocate storage for im...
*/

  im = (float *)malloc(sizeof(float) * row * col);


  while (!feof(stdin))
    {
      fscanf(stdin,"%lf %lf %lf",&lat,&lon,&z);


/* simple registration algorithm, image starts at top left corner */

      ilon=(lon-min_lon)/(max_lon-min_lon)*(col-1);
      if (ilon<0 || ilon > col-1) continue;
      ilat=row-1-(lat-min_lat)/(max_lat-min_lat)*(row-1);
      if (ilat<0 || ilat > row-1) continue;

/* get lon and lat `ectangle size */

      dlon=col*(360./244.)/(max_lon-min_lon)/2.+1;
      dlat=row/(max_lat-min_lat)/S_DEG+2;


      for(j=(ilat<0?0:ilat);j<ilat+dlat && j<row;j++)
        {
          rowoff = j * col;
          for(i= (ilon-dlon<0?0:ilon-dlon);i<=ilon+dlon && i<col;i++)
            {
              ij = rowoff + i;
              im[ij] = (float)z;
            }
        }
    }

/*
  Create sdps format file..

  header.annot = "Geosat Image";
*/
  strcpy(header.annot, "Geosat Image");

  header.type   = FLOAT;
  header.dim    = 2;
  header.ind[0] = col;
  header.ind[1] = row;
  header.ind[2] = 1;
  header.ind[3] = 1;
  header.inc[0] = dlon;
  header.inc[1] = dlat;
  header.inc[2] = 1;
  header.inc[3] = 1;
```

```
  header.slope  = 1.0;
  header.intrcp = 0.0;
  header.cmap   = 0;

  write_sdps(stdout, header, cmap, im);

}
```

Program g_interp.c

```
/*

  @(#)g_interp.c        1.3 12/15/89

  Program g_interp.c

  Written by:
  --------------
  Michael Caruso
  Woods Hole Oceanograhic Institution
  Woods Hole, MA

  Modifications:
  --------------
  12-15-89  MC Changed call to geo_cyc_orb so that time is
            passed and not a GDR.

  Purpose:
  --------
  This program will linearly interpolate all geosat GDR data except the
  10 per second heights and the flags against the latitude value.

  Method:
  -------
  1.    Break data into continuous segments
  2.    Interpolate each data value to a common grid.
  3.    Write out all data between min and max.

  Usage:
  ------
  The GEOSAT GDR is read from standard input.

  cat c000.a002 | g_interp dir min max deltmax timestep> c000.a002s

  will spline the records from the GDR in c000.a002 between min
  and max latitude if dir is 1 and between min and max longitudes
  if dir is 2.


  Input:
  ------
  Stdin         Clean GEOSAT GDRs
  dir           Determines boundaries of spline:
                1       - Use latitude
                2       - Use longitude
  min           Minimum lat or lon.
  max           Maximum lat or lon.
  deltmax       Maximum gap in seconds for a contiguous
                segment.
```

```
  timestep      Interval between interpolated points.

  Output:
  -------
  Stdout:       Interpolated GEOSAT GDRs.  Values in gaps
                between valid segments are set to all
                zeros.

  Subroutines and Subprograms Required:
  ---------------------
  geo_cyc_orb   Returns the cycle number and orbit number
                for a given time.
  fit_time      Initializes interpolation variables and fits an
                interpolation to the time variable.
  fit_data      Interpolates between points of a given data
                set.

  References:
  ----------

*/
# include <math.h>
# include <stdio.h>
# include "geos.h"

#define NUMARGS         5
#define DIARG           1
#define MIARG           2
#define MAARG           3
#define DEARG           4
#define TIARG           5
#define MICRO           1e-6   /* Conv for utcm                                    */
#define MAXPOINTS       2915   /* +/-  degrees latitude                            */

/*
  Define some globals to share with fit subprograms...
*/

int  ii, j, k;
int latstart, latstop;
struct frame frsin[MAXPOINTS];  /* frame MAXPOINTS/2 is defined at the equator */
struct frame frout[MAXPOINTS];
float timestep;
short int points=0;
float *x;
float *y2;
float *lat;
float *y;

float x0, minval, maxval;

/*
  Variables for orbit analysis.
*/
```

```
int cyc, orb;
double rs, rs3, cosinc, prec, rot, dthdt;
double time, theta, sinth, lat0, lon0, tmp;

short int seg_len[1000];
short int seg_beg[1000];

unsigned char ascorb;

main (argc,argv)
     int argc;
     char *argv[];
{

/*
  Declare subroutines and subprograms...
*/

  int fit_time();
  int fit_data();
  int geo_cyc_orb();


  double       time1, time2;   /* time variable used to determine gap  */

  float deltmax;

  short int i;

  short int dir;
  short int segments = 0;


  static char SccsId[] = "@(#)g_spline.c       1.4\t6/23/89";

  x = (float *)calloc(MAXPOINTS, sizeof(float));
  y2 = (float *)calloc(MAXPOINTS, sizeof(float));
  y = (float *)calloc(MAXPOINTS, sizeof(float));
  lat = (float *)calloc(MAXPOINTS, sizeof(float));


  if ((argc != NUMARGS+1) && (argc != 1))
    {
      fprintf(stderr,
      "Usage: %s [dir min max deltmax timestep] < filein > filout\n",
       argv[0]);
      exit(1);
    }
  if (argc == NUMARGS+1)
    {
      sscanf(argv[DIARG],"%hd", &dir);
      sscanf(argv[MIARG],"%f", &minval);
      sscanf(argv[MAARG],"%f", &maxval);
      sscanf(argv[DEARG],"%f", &deltmax);
```

```
      sscanf(argv[TIARG],"%f", &timestep);
    }
  else
    {
      fprintf(stderr,
      "Usage: %s [dir min max deltmax timestep] < filein > filout\n",
      argv[0]);
      exit(1);
    }


  /*
    Read in all GDRs...
    */
  while (fread((char*)&frsin[points],1,REC_LEN,stdin)==REC_LEN) points++;
  points--;

  if (points <= 0)
    {
      fprintf(stderr,"%s: No points to spline\n\n",argv[0]);
      exit(0);
    }

/*
  Determine if orbit is ascending or descending...
*/
  if((frsin[0].lat < frsin[1].lat) && (frsin[1].lat < frsin[points].lat))
    {
      ascorb = TRUE;
    }
  else if((frsin[0].lat > frsin[1].lat) && (frsin[1].lat > frsin[points].lat))
    {
      ascorb = FALSE;
    }
  else
    {
      fprintf(stderr,"%s: Unable to determine if orbit is ascending or ");
      fprintf(stderr,"descending\n", argv[0]);
      exit(1);
    }
/*
  Set up for direction.  If dir != 1, then we
  find the latitude that corresponds to minlon
  and maxlon.
*/
  if (dir != 1)
    {
      lon_to_lat();
    }

  /*
    Fill latitude array and find starting index...
    */
  for (i=0; i< MAXPOINTS; i++)
```

```
  {
    if (ascorb)
      {
        lat[i] = DEG*asin(sin((i-MAXPOINTS/2)*timestep*M2PI/PERIOD)*sin(INC));
        if (minval > lat[i])  latstart = i+1;
      }
    else
      {
        lat[i] = DEG*asin(sin(((MAXPOINTS/2)-i)*timestep*M2PI/PERIOD)*sin(INC));
        if (maxval < lat[i])  latstart = i+1;
      }

  }

/*
  1. Break into data segments...
  */
seg_len[0] = 1;
seg_beg[0] = 0;
time2 = frsin[0].utc + frsin[0].utcm*MICRO;
for (i=0; i<points; i++)
  {

    time1 = time2;
    time2 = frsin[i+1].utc + frsin[i+1].utcm*MICRO;

    if ((time2 - time1) <= deltmax)
      {
        seg_len[segments]++;
      }
    else
      {
        segments++;
        seg_beg[segments] = i + 1;
        seg_len[segments] = 1;
      }
  }

if (seg_len[0] == 0)  exit(1);

/*
  2. Initialize data and fit a spline to each segment...
  */

for(i=latstart; i<MAXPOINTS; i++)
  {
    frout[i].lat = (int)(lat[i]/MICRO);
    frout[i].lon = BAD;
    frout[i].m_h = BAD;
  }

for (i=0; i<=segments; i++)
  {
    fit_time(i);              /* initialize spline */
```

```
      fit_data(i,0);                /* lon */
      fit_data(i,1);                /* h */
      fit_data(i,2);                /* orb */
      fit_data(i,3);                /* s_h */
      fit_data(i,4);                /* geoid */
      fit_data(i,5);                /* swh */
      fit_data(i,6);                /* s_swh */
      fit_data(i,7);                /* s_nght */
      fit_data(i,8);                /* agc */
      fit_data(i,9);                /* s_agc */
      fit_data(i,10);               /* s_tide */
      fit_data(i,11);               /* o_tide */
      fit_data(i,12);               /* w_fnoc */
      fit_data(i,13);               /* w_smmr */
      fit_data(i,14);               /* d_fnoc */
      fit_data(i,15);               /* iono */
      fit_data(i,16);               /* dh_swh */
      fit_data(i,17);               /* dh_fm */
      fit_data(i,18);               /* att */
    }

/*
  Write out points...
*/

  for (k=latstart; k<j; k++)
    {
      if (fwrite((char *)&frout[k], 1, REC_LEN, stdout) != REC_LEN)
        {
          geo_error(3,argv[0]);
          exit(3);
        }

    }

}

/*
------------------------------------------------------------
  Subroutine fit_time
  Written by:
    Michael Caruso
    Woods Hole Oceanographic Institution
    Woods Hole, MA

  Purpose:
    This subprogram fits a spline to the time variable in a
    GEOSAT GDR.
------------------------------------------------------------
*/

int fit_time(i)
     int i;
{
```

```
double yout;
ii = 0;
for (j=seg_beg[i]; j<seg_beg[i]+seg_len[i]; j++)
  {

    x[ii] = frsin[j].lat*MICRO;
    y[ii] = (float)(frsin[j].utc - frsin[seg_beg[i]].utc)
      + (float)frsin[j].utcm*MICRO;
    ii++;
  }
x0 = x[0];


j = latstart;
while ((ascorb && (lat[j] < x0)) || (!ascorb && (lat[j] > x0)))
  {
    j++;
  }
while ((lat[j] > minvrl) && (lat[j] < maxval))
  {
    if ((ascorb && (lat[j] > frsin[seg_beg[i]].lat*MICRO) &&
         (lat[j] < frsin[seg_beg[i]+seg_len[i]-1].lat*MICRO)) ||
        (!ascorb && (lat[j] < frsin[seg_beg[i]].lat*MICRO) &&
         (lat[j] > frsin[seg_beg[i]+seg_len[i]-1].lat*MICRO)))
      {
        ii = 0;
        if (ascorb)
          {
            while(x[ii] < lat[j])
              ii++;
          }
        else
          {
            while(x[ii] > lat[j])
              ii++;
          }

        linear((double)x[ii], (double)y[ii], (double)x[ii-1],
               (double)y[ii-1], (double)lat[j], &yout);

        frout[j].utc = (int)yout + frsin[seg_beg[i]].utc;
        frout[j].utcm = (int)((yout - (int)yout)/MICRO);
      }
    else
      {
        frout[j].lat = (int)(lat[j]/MICRO);
        frout[j].utc = 0;
        frout[j].utcm = 0;
      }
    j++;
  }

return;
```

```
}
/*
--------------------------------------------------
  Subroutine fit_data
  Written by:
    Michael Caruso
    Woods Hole Oceanographic Institution
    Woods Hole, MA

  Purpose:
    This subprogram fits a spline to the data segment
    specified.
--------------------------------------------------
*/

int fit_data(i, var)
     int i, var;
{

  double yout;
  int iyout;
  ii = 0;
  for (j=seg_beg[i]; j<seg_beg[i]+seg_len[i]; j++)
    {

      x[ii] = frsin[j].lat*MICRO;
      switch (var)
        {
        case 0:
          y[ii] = (float)frsin[j].lon;
          break;
        case 1:
          y[ii] = (float)frsin[j].m_h;
          break;
        case 2:
          y[ii] = (float)frsin[j].orb;
          break;
        case 3:
          y[ii] = (float)frsin[j].s_h;
          break;
        case 4:
          y[ii] = (float)frsin[j].geoid;
          break;
        case 5:
          y[ii] = (float)frsin[j].swh;
          break;
        case 6:
          y[ii] = (float)frsin[j].s_swh;
          break;
        case 7:
          y[ii] = (float)frsin[j].s_nght;
          break;
        case 8:
          y[ii] = (float)frsin[j].agc;
```

```
          break;
        case 9:
          y[ii] = (float)frsin[j].s_agc;
          break;
        case 10:
          y[ii] = (float)frsin[j].s_tide;
          break;
        case 11:
          y[ii] = (float)frsin[j].o_tide;
          break;
        case 12:
          y[ii] = (float)frsin[j].w_fnoc;
          break;
        case 13:
          y[ii] = (float)frsin[j].w_smmr;
          break;
        case 14:
          y[ii] = (float)frsin[j].d_fnoc;
          break;
        case 15:
          y[ii] = (float)frsin[j].iono;
          break;
        case 16:
          y[ii] = (float)frsin[j].dh_swh;
          break;
        case 17:
          y[ii] = (float)frsin[j].dh_fm;
          break;
        case 18:
          y[ii] = (float)frsin[j].att;
          break;
        default:
          return(1);
        }

      ii++;
    }
  x0 = x[0];

  j = latstart;
  while ((ascorb && (lat[j] < x0)) || (!ascorb && (lat[j] > x0)))
    {
      j++;
    }
  while ((lat[j] > minval) && (lat[j] < maxval))
    {
      if ((ascorb && (lat[j] > frsin[seg_beg[i]].lat*MICRO) &&
           (lat[j] < frsin[seg_beg[i]+seg_len[i]-1].lat*MICRO)) ||
          (!ascorb && (lat[j] < frsin[seg_beg[i]].lat*MICRO) &&
           (lat[j] > frsin[seg_beg[i]+seg_len[i]-1].lat*MICRO)))
        {
          ii = 0;
          if (ascorb)
            {
```

```
            while(x[ii] < lat[j])
              ii++;
          }
        else
          {
            while(x[ii] > lat[j])
              ii++;
          }

        linear((double)x[ii], (double)y[ii], (double)x[ii-1],
               (double)y[ii-1], (double)lat[j], &yout);

        iyout = nint(yout);
        set_data(j, iyout, var);
      }
    else
      {
        set_data(j, BAD, var);
      }
    j++;
  }
  natcubspline(x, y, ii, lat[j], &yout, 2);

  return(0);
}

/*
------------------------------------------------
  Subroutine set_data

  Written by:
    Michael Caruso
    Woods Hole Oceanographic Institution
    Woods Hole, MA

  Purpose:
    This subroutine simply puts the data into the
    correct element.
-------------------------------------------------
*/
int set_data(point, data, var)
     int point;
     int data;
     int var;
{
  switch (var)
    {
    case 0:
      frout[point].lon = data;
      break;
    case 1:
      frout[point].m_h = data;
      break;
    case 2:
```

```
      frout[point].orb = data;
      break;
    case 3:
      frout[point].s_h = data;
      break;
    case 4:
      frout[point].geoid = data;
      break;
    case 5:
      frout[point].swh = data;
      break;
    case 6:
      frout[point].s_swh = data;
      break;
    case 7:
      frout[point].s_nght = data;
      break;
    case 8:
      frout[point].agc = data;
      break;
    case 9:
      frout[point].s_agc = data;
      break;
    case 10:
      frout[point].s_tide = data;
      break;
    case 11:
      frout[point].o_tide = data;
      break;
    case 12:
      frout[point].w_fnoc = data;
      break;
    case 13:
      frout[point].w_smmr = data;
      break;
    case 14:
      frout[point].d_fnoc = data;
      break;
    case 15:
      frout[point].iono = data;
      break;
    case 16:
      frout[point].dh_swh = data;
      break;
    case 17:
      frout[point].dh_fm = data;
      break;
    case 18:
      frout[point].att = data;
      break;
    default:
      return(1);
    }
}
```

```
/*
----------------------------------------
  Subroutine lon_to_lat

  Written by:
    Michael Caruso
    Woods Hole Oceanographic Insititution
    Woods Hole, MA

  Purpose:
    This subroutine converts the minval and
    maxval when given in longitude to latitude.
-------------------------------------------
*/
int
  lon_to_lat()

{
  int i;
  float tmpval;
  double tmptime;

  /* Determine orbit number */

  tmptime = frsin[0].utc + frsin[0].utcm*MICRO;
  geo_cyc_orb(tmptime, &cyc, &orb);

  rs = RE + frsin[0].orb;
  rs3 = rs*rs*rs;
  cosinc = cos(INCL);
  prec = -1.5*J2*sqrt(GM/rs)*RE*RE*cosinc/rs3;
  rot = prec - (M_PI_2/SD);
  dthdt = M2PI/PERIOD;

  /*
    Loop until we find min_lat and
    max_lat.
    */

  lon0 = frsin[0].lon*1.0e-6*RAD;
  time = 0.0;


  if (ascorb)
    {

      i = 0;
      while (lon0 > maxval*RAD)
        {
          i++;
          lon0 = frsin[i].lon*1.0e-6*RAD;
        }
```

```
    while (lon0 < maxval*RAD)
      {
        time -= timestep;
        theta = dthdt*time;
        sinth = sin(theta);
        lat0 = (frsin[i].lat*1.0e-6)*RAD + asin(sin(INCL)*sin(theta));
        tmp = cosinc*sinth/cos(lat0);
        tmp = (tmp>1.0) ? 1.0 : tmp;
        tmp = (tmp<-1.0) ? -1.0 : tmp;
        lon0 = (asin(tmp) + rot*time) + frsin[i].lon*1.0e-6*RAD;
      }

    lon0 = frsin[points].lon*1.0e-6*RAD;
    tmpval = lat0*DEG;
    time = 0.0;
    i = points;
    while (lon0 < minval*RAD)
      {
        i--;
        lon0 = frsin[i].lon*1.0e-6*RAD;
      }
    while (lon0 > minval*RAD)
      {
        time += timestep;
        theta = dthdt*time;
        sinth = sin(theta);
        lat0 = (frsin[i].lat*1.0e-6)*RAD + asin(sin(INCL)*sin(theta));
        tmp = cosinc*sinth/cos(lat0);
        tmp = (tmp>1.0) ? 1.0 : tmp;
        tmp = (tmp<-1.0) ? -1.0 : tmp;
        lon0 = (asin(tmp) + rot*time) + frsin[i].lon*1.0e-6*RAD;
      }
    minval = tmpval;
    maxval = lat0*DEG;
  }
else
  {

    lon0 = frsin[0].lon*1.0e-6*RAD;
    time = 0.0;
    i = 0;
    while (lon0 > maxval*RAD)
      {
        i++;
        lon0 = frsin[i].lon*1.0e-6*RAD;
      }

    while (lon0 < maxval*RAD)
      {
        time -= timestep;
        theta = dthdt*time;
        sinth = sin(theta);
        lat0 = (frsin[i].lat*1.0e-6)*RAD + asin(sin(INCL)*sin(theta));
        tmp = cosinc*sinth/cos(lat0);
```

```
        tmp = (tmp>1.0) ? 1.0 : tmp;
        tmp = (tmp<-1.0) ? -1.0 : tmp;
        lon0 = (asin(tmp) + rot*time) + frsin[i].lon*1.0e-6*RAD;
      }

    maxval = lat0*DEG;
    lon0 = frsin[points].lon*1.0e-6*RAD;
    time = 0.0;
    i = points;
    while (lon0 < minval*RAD)
      {
        i--;
        lon0 = frsin[i].lon*1.0e-6*RAD;
      }
    while (lon0 > minval*RAD)
      {
        time += timestep;
        theta = dthdt*time;
        sinth = sin(theta);
        lat0 = (frsin[i].lat*1.0e-6)*RAD + asin(sin(INCL)*sin(theta));

        tmp = cosinc*sinth/cos(lat0);
        tmp = (tmp>1.0) ? 1.0 : tmp;
        tmp = (tmp<-1.0) ? -1.0 : tmp;
        lon0 = (asin(tmp) + rot*time) + frsin[i].lon*1.0e-6*RAD;
      }
    minval = lat0*DEG;
  }

}
```

Program g_print.c

```
/*
 @(#)g_print.c 1.3 6/14/90

 Written by:
 -----------
 Pierre Flament
 University of Hawaii
 Honolulu, HI  96822

 Modifications:
 --------------
 Michael Caruso
 Woods Hole Oceanographic Institution
 Woods Hole, MA

 Added Coments and cleaned up some of code.

 Purpose:
 --------
 Decodes Geosat GDR record and prints in a
 lengthy format.

 Method:
 -------
 Reads Geosat GDR from standard input and converts
 each element to ascii and prints each element with
 an identifier to standard output.

 Usage:
 ------
 g_print < c???.a???

 Input:
 ------
 c???.a???      Raw Geosat GDRs from standard input.

 Output:
 -------
 Standard Output                Formatted output.

 Note:
 -----
 The flags are ordered as follows:

               FEDCBA9876543210

               where 0 is the zeroth flag bit
               and F is the fifteenth flag bit as
               listed in the Geosat Altimeter GDR User
```

```
          Handbook.

  Subroutines Required:
  -------------------
  None.

*/
# include <stdio.h>
# include "geos.h"

/* Labels for output...  */


char *xlab[]={  "utc ",
                "utcm",
                "lat ",
                "lon ",
                "orb "};

char *ylab[]={  "m_h        ",
                "s_h        ",
                "geoid      ",
                "h[1]       ",
                "h[2]       ",
                "h[3]       ",
                "h[4]       ",
                "h[5]       ",
                "h[6]       ",
                "h[7]       ",
                "h[8]       ",
                "h[9]       ",
                "h[10]      ",
                "swh        ",
                "s_swh      ",
                "s_naught   ",
                "agc        ",
                "s_agc      "};

char *zlab[]={  "h_off   ",
                "sol_tide",
                "oc_tide ",
                "wet_fnoc",
                "wet_smmr",
                "dry_fnoc",
                "iono_gps",
                "dh_swh  ",
                "dh_fm   ",
                "att     "};


short int       i,j;
long int        record = 0;
```

```
struct          flags *f;

main()

{

  while(fread((char*)&fr,1,REC_LEN,stdin)==REC_LEN)
   {
     printf("\f");

     printf("Record Number:\t%10ld\n",++record);
     printf("%s :\t%10ld\t",xlab[0],fr.utc);
     printf("%s :\t%10ld\n",xlab[1],fr.utcm);
     printf("%s :\t%10ld\t",xlab[2],fr.lat);
     printf("%s :\t%10ld\n",xlab[3],fr.lon);
     printf("%s :\t%10ld\n",xlab[4],fr.orb);

     printf("\n");

     printf("%s :\t%6d\t",ylab[0],fr.m_h);
     printf("%s :\t%6d\n",ylab[1],fr.s_h);
     printf("%s :\t%6d\n",ylab[2],fr.geoid);
     for(i=0; i<10; i++)
       {
         if(i%2)
           printf("%s :\t%6d\n",ylab[i+3],fr.h[i]);
         else
           printf("%s :\t%6d\t",ylab[i+3],fr.h[i]);
       }
     printf("%s :\t%6d\t",ylab[13],fr.swh);
     printf("%s :\t%6d\n",ylab[14],fr.s_swh);
     printf("%s :\t%6d\n",ylab[15],fr.s_nght);
     printf("%s :\t%6d\t",ylab[16],fr.agc);
     printf("%s :\t%6d\n",ylab[17],fr.s_agc);

     f = &(fr.fl);
     printf("\nflags (0-15 right to left):\t%d%d%d%d%d%d%d%d%d%d%d%d%d%d%d%d\n\n",
            f->junk15,f->junk14,f->s,f->t,f->junk11,f->junk10,f->junk09,
            f->junk08,f->junk07,f->a6,f->a5,f->a4,f->h,f->r,f->d,f->w);

     printf("%s :\t%6d\n",zlab[0],fr.h_off);
     printf("%s :\t%6d\t",zlab[1],fr.s_tide);
     printf("%s :\t%6d\n",zlab[2],fr.o_tide);
     printf("%s :\t%6d\t",zlab[3],fr.w_fnoc);
     printf("%s :\t%6d\t",zlab[4],fr.w_smmr);
     printf("%s :\t%6d\n",zlab[5],fr.d_fnoc);
     printf("%s :\t%6d\n",zlab[6],fr.iono);
     printf("%s :\t%6d\n",zlab[7],fr.dh_swh);
     printf("%s :\t%6d\n",zlab[8],fr.dh_fm);
     printf("%s :\t%6d\n",zlab[9],fr.att);

   }
}
```

Program g_region.c

```
/*

 @(#)g_region.c        1.5 12/19/89

 Program  g_region.c

  Written by:
  -----------
  Michael Caruso
  Woods Hole Oceanographic Institution
  Woods Hole, MA

  Modifications:
  --------------
  12-15-89  MC Changed call to geo_cyc_orb so that time is
            passed and not a GDR.

  Purpose:
  --------
  Decodes GEOSAT data and separates raw data into separate orbits.  Each
  orbit is defined as beginning at the northernmost point of a track.
  Each orbit is further separated into an ascending section and a descending
  section.  Orbits are then written out to separate files of the form:

             cmmm.annn or cmmm.nnnd
      where
             mmm             is the cycle number,
             nnn             is the orbit number for that cycle,
             a               signifies ascending portion,
             d               signifies descending portion.

      The data is written out in the same form as it was read in.  This is
      consecutive records of 78 bytes each.

 Usage:
 ------
 The program reads the minimum and maximum latitudes and longitudes
 from the command line and reads the data from standard input.  To
 use the program to extract data from tape (/dev/rmt8, 6250bpi, input
 block size 16380) HP format from NODC, from 10N to 30N and 280E to
 300E:

 dd if=/dev/rmt8 ibs=16380 files=34 | g_region 1 10 30 280 300

 The first number on the argument line specifies whether the box
 should be bounded by a latitude line(1) or a longitude line(2).
 Note that longitudes are all east of Greenwich and if the box selected
 spans 360E, add 360 degrees to right edge of box, ie 350 365.
```

```
  Input:
  ------
  Stdin         Raw Geosat GDRs

  Output:
  -------
  c???.????     Geosat data within region separated in ascending and
                descending orbits.

  Subroutines and Subprograms Required:
  -------------------------------------
  geo_cyc_orb   Returns the cycle number and orbit number
                for a given time.
  geo_which     returns an array of 1's and 0's for each cycle
                within the desired box.
  geo_error     prints error messages to standard error.

  References:
  -----------

  Bugs:
  -----
        Assumes input data contains complete orbits.


*/

#include <stdio.h>
#include <sys/file.h>
#include <math.h>
#include "geos.h"

#define NUMARG  5
#define DIRARG  1
#define MNLTARG 2
#define MXLTARG 3
#define MNLNARG 4
#define MXLNARG 5


main(argc, argv)
     int argc;
     char *argv[];
{

  struct frame fr2;

  unsigned char a[ORB_PER_CYC],
                d[ORB_PER_CYC],
                c[ORB_PER_CYC]; /* arrays of orbits within box          */

  char          str[80];        /* string for output file name          */

  int           i,j;            /* Counters                             */
```

```
short int     dir;            /* Direction of lat/lon boundary        */
short int     orbit_num_tot;  /* the total number of orbits           */
int           cycle_num;      /* the number of cycles since           */
int           orbit_num;      /* orbit number within cycle 0-244      */
short int     isopen = FALSE; /* check to see if file is already open */
short int     asc;            /* flag for ascending or descending     */

long int      llcmp,
              llmin, llmax;   /* lat/lon boundary                     */
long int      lslope1,
              lslope2;        /* "Slope" of orbit                     */


double        min_lat,
              max_lat,
              min_lon,
              max_lon;        /* input lon-lat box                    */
double        time;           /* time variable                        */


FILE *fdout;                  /* output file descriptor               */


/*
  Read command line arguments.
  */

if (argc >= NUMARG + 1)
  {
    sscanf(argv[DIRARG], "%hd", &dir);
    sscanf(argv[MNLTARG],"%lf",&min_lat);
    sscanf(argv[MXLTARG],"%lf",&max_lat);
    sscanf(argv[MNLNARG],"%lf",&min_lon);
    sscanf(argv[MXLNARG],"%lf",&max_lon);
  }
else
  {
    fprintf(stderr,
    "Usage: %s dir min_lat max_lat min_lon max_lon [orb#]\n", argv[0]);
    exit(1);
  }

/* determine orbits to remove. */

geo_which(min_lat, max_lat, min_lon, max_lon, a, d);

/*
  Mask out orbit numbers if given on command line...
  */
if (argc > NUMARG+1)
  {
    for (i=6; i<argc; i++)
```

```
      {
        sscanf(argv[i], "%d", &j);
        c[j] = TRUE;
      }
    for (i=0; i<ORB_PER_CYC; i++)
      {
        a[i] = a[i] && c[i];
        d[i] = d[i] && c[i];
      }
  }

/*    Set llmin, llmax...     */

if (dir == 1)
  {
    llmin = (int) (min_lat*1.0e06);
    llmax = (int) (max_lat*1.0e06);
  }
else
  {
    llmin = (int) (min_lon*1.0e06);
    llmax = (int) (max_lon*1.0e06);
  }


/* read initial lat and long coordinates */

if(fread((char *)&fr,1,REC_LEN,stdin) != REC_LEN)
  {
    geo_error(2, argv[0]);
    exit(2);
  }

if(fread((char *)&fr2,1,REC_LEN,stdin) != REC_LEN)
  {
    geo_error(2, argv[0]);
    exit(2);
  }
/* Determine name of first orbit */

time = fr.utc + fr.utcm * 1.0e-6;
lslope1 = fr2.lat - fr.lat;

geo_cyc_orb(time, &cycle_num, &orbit_num);

/*
  Check to see if first orbit is ascending or
  descending...
  */

if ( lslope1 > 0 )
  {
    sprintf(str,"c%.3d.a%.3d",cycle_num,orbit_num);
    asc = TRUE;
```

```
  }
else if ( lslope1 < 0)
  {
    sprintf(str,"c%.3d.d%.3d",cycle_num,orbit_num);
    asc = FALSE;
  }
else
  {
    if (fr.lat < 0)
      {
        sprintf(str,"c%.3d.a%.3d",cycle_num,orbit_num);
        asc=TRUE;
      }
    else
      {
        sprintf(str,"c%.3d.d%.3d",cycle_num,orbit_num);
        asc=FALSE;
      }
  }

/* Check to see if point is an orbit we want and greater
   than the minimum latitude and smaller than the
   maximum latitude. If so, write to the output file. If
   the output file is not open, open it and mark it as
   being open.         */

llcmp = (dir == 1) ? fr.lat : fr.lon;

if(((asc && a[orbit_num]) || (!asc && d[orbit_num])) && (llcmp > llmin)
   && (llcmp < llmax))
  {
    if (isopen == 0)
      {
        fdout = fopen(str,"a");
        isopen = 1;
      }
    if(fwrite((char *)&fr,1,REC_LEN,fdout) != REC_LEN)
      {
        geo_error(3, argv[0]);
        exit(3);
      }
  }

/*
 Check second point...
 */

llcmp = (dir == 1) ? fr2.lat : fr2.lon;

if(((asc && a[orbit_num]) || (!asc && d[orbit_num])) && (llcmp > llmin)
   && (llcmp < llmax))
  {
    if (isopen == 0)
      {
```

```
        fdout = fopen(str,"a");
        isopen = 1;
      }
    if(fwrite((char *)&fr2,1,REC_LEN,fdout) != REC_LEN)
      {
        geo_error(3, argv[0]);
        exit(3);
      }
  }



/* Read in rest of geosat data.  We keep three points active
   to monitor when an orbit changes from ascending to descending.
   This was done because of the incomplete data at high latitudes.
   */

fr = fr2;

while(fread((char *)&fr2,1,REC_LEN,stdin) == REC_LEN)
  {

    lslope2 = fr2.lat - fr.lat;

    if ((lslope1 > 0 && lslope2 <= 0) || (lslope1 < 0 && lslope2 >= 0))
      {
        /* Determine name of next orbit */

        time = fr2.utc + fr2.utcm * 1.0e-6;

        geo_cyc_orb(time, &cycle_num, &orbit_num);

        if ( lslope2 > 0 )
          {
            sprintf(str,"c%.3d.a%.3d",cycle_num,orbit_num);
            asc=TRUE;
          }
        else if ( lslope2 < 0 )
          {
            sprintf(str,"c%.3d.d%.3d",cycle_num,orbit_num);
            asc=FALSE;
          }
        else
          {
            if (fr2.lat < 0 )
              {
                sprintf(str,"c%.3d.a%.3d",cycle_num,orbit_num);
                asc=TRUE;
              }
            else
              {
                sprintf(str,"c%.3d.d%.3d",cycle_num,orbit_num);
                asc=FALSE;
```

```
            }
          }

        fclose(fdout);          /* Close previous file */
        isopen = 0;
      }
    llcmp = (dir == 1) ? fr2.lat : fr2.lon;

    if(((asc && a[orbit_num]) || (!asc && d[orbit_num])) && (llcmp > llmin)
       && (llcmp < llmax))
      {
        if (isopen == 0)
          {
            fdout = fopen(str,"a");
            isopen = 1;
          }
        if(fwrite((char *)&fr2,1,REC_LEN,fdout) != REC_LEN)
          {
            geo_error(3, argv[0]);
            exit(3);
          }
      }

    fr = fr2;
    lslope1 = lslope2;

  }

}
```

Program g_repeat.c

```
/*
  @(#)g_repeat.c        1.3 12/18/89

  Program g_repeat.c

  Written by:
  --------------
  Michael Caruso
  Woods Hole Oceanograhic Institution
  Woods Hole, MA


  Purpose:
  --------
  This program will perform a repeat track analysis
  of geosat GDR's.

  Method:
  -------
  1.    Read in all cycles
  2.    Calculate the mean sea surface height (m_h)
  3.    Subtract mean from each cycle.
  4.    Calculate quadratic regression and subtract from
        each cycle.
  5.    Calculate a second regression weighted by the
        inverse of the variance of the first regression.
  6.    Subtract new fit from each profile to obtain
        final heights.
  7.    Calculate mean and variance.
  8.    Print results.

  Usage:
  ------
  Each GEOSAT GDR is read from a separate file.

  g_repeat c???.a002 > data.text

  will perform a repeat track analysis from the cleaned and splined
  GDR's in all available cycle for track a002.

  Input:
  ------
  c???.a002     All cycles for the speciifed track cleaned
                and splined.

  Output:
  -------
  Stdout:       Data file containing the latitude, the
```

```
                longitude, the mean height(m), the rms height
                variance and the number of valid points for each
                location of a given orbit.

  c???.a002_r   Data file containing the latitude, the
                longitude and the residual height.
  Subroutines Required:
  --------------------
  cr_mat_float         creates a floating point matrix.
  cr_mat_double        creates a double precision matrix.
  gauss_elim           solves linear system of equations.

  References:
  -----------

*/

# include <math.h>
# include <stdio.h>
# include "geos.h"

#define MICRO           1.e-6   /* conversion from micro-deg to deg             */
#define MILLI           1.e-2   /* conversion from centimeter to meter          */
#define MAXPOINTS       2915    /* Max point -70 to +70  degrees latitude       */

/*
  Global Variables...
*/

float **h;              /* Original Heights. */
float **h_tmp;          /* Temporary Heights. */

float *mean;            /* Mean of original heights (h) */
float *mean2;           /* Mean of original heights - quad orbit corr.*/
float *mean_lon;        /* Mean of Longitudes */
float *lat;             /* Latitudes of first orbit. */
                        /* We don't find mean_lat because the
                           latitudes are fixed in a previous
                           program such as g_spline
                           */

float *var;             /* Variance of original heights */
float *var2;

int *count;             /* Count for mean and var*/
int *count2;            /* Count for mean2 and var2*/

double **a, *b;         /* Used for quadratic fit*/
double *xans;           /* Used for quadratic fit */
float **quad0;          /* Keep values of quadratic fit for printing */
float **quad1;          /* Keep values of quadratic fit for printing */

int *cyc;               /* Keep track of good and bad cycles. */
int maxcyc;             /* Maximum number of good points at each
```

```
                                latitude grid point
                                */

int     i;              /* Counter */
int     ipoint;         /* Counter for points read in for each cycle.
                           Note: program assumes each cycle has been
                           regridded to a common grid and has the same
                           number of points.
                           */


main (argc,argv)
     int argc;
     char *argv[];
{

/*
  Declare non-integer subroutines:
*/

  float **cr_mat_float();
  double **cr_mat_double();

/*
  I/O file descriptors
*/

  FILE *gfile, *ofile;

/*
  Various counters.
*/

  int iarg;
  int ipointold;
  int fill = FALSE;

/*
  Temporary variables for calculations prior to printing.
*/
  float htmp, vtmp;
  float fit0, fit1;

/*
  Misc. variables.
*/
  int err;              /* Returned error message.      */
  char str[80];         /* String for filenames etc. */


/*
  Create arrays described above...
*/
  h             = cr_mat_float(argc, MAXPOINTS);
```

```
h_tmp          = cr_mat_float(argc, MAXPOINTS);

mean           = (float *)calloc(MAXPOINTS, sizeof(float));
mean2          = (float *)calloc(MAXPOINTS, sizeof(float));
mean_lon       = (float *)calloc(MAXPOINTS, sizeof(float));
lat            = (float *)calloc(MAXPOINTS, sizeof(float));

var            = (float *)calloc(MAXPOINTS, sizeof(float));
var2           = (float *)calloc(MAXPOINTS, sizeof(float));

count          = (int *)calloc(MAXPOINTS, sizeof(int));
count2         = (int *)calloc(MAXPOINTS, sizeof(int));

cyc            = (int *)calloc(argc, sizeof(int));

a              = cr_mat_double(3,3);
b              = (double *)calloc(3, sizeof(double));
xans           = (double *)calloc(3, sizeof(double));
quad0          = cr_mat_float(argc,3);
quad1          = cr_mat_float(argc,3);

/*
  Check count2 and quad1 to see if they were allocated space.  If so,
  assume that all other arrays were allocated ok.
  */

  if ((quad1 == NULL) || (count2 == NULL))
    {
      fprintf(stderr,"%s: Unable to allocate enough storage space.\n",
              argv[0]);
      exit(1);
    }

/*
  Check to see if program is given arguments...
*/
  if (argc == 1)
    {
      fprintf(stderr,"Usage: %s c???.a000 > fileout.text\n",argv[0]);
      exit(1);
    }


  /*
    Read in all GDRs and calculate mean and variance...
    iarg        is the cycle number to read in.
    ipoint      is the along track point
    */

  for(iarg=0; iarg<argc-1; iarg++)
    {
      /*
        Open each input file...
        */
```

```
    if ((gfile = fopen(argv[iarg+1],"r")) == NULL)
      {
        fprintf(stderr,"%s: Unable to open file %s. Continuing...\n",
                argv[0], argv[iarg+1]);
        cyc[iarg] = BAD;
        continue;  /* If there is no file, try the next file. */
      }

    ipoint = 0;

    while (fread((char*)&fr,1,REC_LEN,gfile)==REC_LEN)
      {
        if (!fill)    /* Fill latitude from first good cycle. */
          {
            lat[ipoint] = fr.lat;
          }
        else                    /* Check against first cycle */
          {                     /* to make sure points line up */
            if((abs((int)lat[ipoint]-fr.lat) > 1000) && (fr.lat != 0))
              {
                fprintf(stderr,
                        "%s: Repeat tracks out of sync.  Offending file: %s\n",
                        argv[0], argv[iarg+1]);
                exit(1);
              }
          }

        /*
          Store Heights...
          */

        h[iarg][ipoint]   = fr.m_h;

        /*
          Set up to find mean and variances...
          */

        if ((fr.m_h != BAD) && (fr.lon !=BAD))
          {
            mean[ipoint] += fr.m_h;
            var[ipoint]   += fr.m_h*fr.m_h;
            count[ipoint] += 1;
            mean_lon[ipoint] += fr.lon;
          }
        ipoint++;
      }

/*
  If a cycle has no points, mark that cycle as BAD and
  print error message.
*/
    if(ipoint == 0)
      {
        cyc[iarg] = BAD;
```

```
          ipoint = ipointold;
          fprintf(stderr,"%s: Bad file %s, no data found\n",argv[0], argv[iarg+1]);
        }
      else
        {
          ipointold = ipoint;
          fill = TRUE;
        }

    }

/*
  If no points were read, exit program...
*/
  if ((ipoint == 0) && (ipointold == 0))
    {
      fprintf(stderr,"%s: No points read, unable to perform analysis.\n", argv[0]);
      exit(1);
    }
  ipoint--;

/*
  Find mean and variance of raw data.  Also find the mean lon
  at each point and determine the maximum number of cycles.
  mean, var, mean_lon and maxcyc.
*/

  calc_mean1();

/*
  Calculate quadratic regression of difference and subtract
  from each cycle...
*/

  fit_quad(argc, 0);


/*
  Find mean and var of h_tmp.
  mean2 and var.
*/
  calc_mean2();

/*
  Zero mean2, var2 and count2 and
  Calculate weighted regression...
*/


  zero_2();
  fit_quad(argc, 1);

/*
  Find mean and var of h_tmp after weighted
```

```
  regresssion - mean2 and var.
*/

  calc_mean2();

/*
  Write out residuals for each good cycle.
  Concatenate "_r" to the end of the file name.  This was
  done instead of substitution since the user may call the
  program with subdirectories - c000/c000.a002c, in which
  case a prefix would change the directory name; or if the
  input file does not end with an additional character -
  c000.a002, substituting the last character would affect
  the file name.

  Output file format:

  lat   lon     residual        fit0    fit1

  Where the residual is h - hmean, and fit0 is
  the resulting fit of the first quadratic and fit1
  is the fit of the second quadratic.
*/

      for (iarg=0; iarg<argc-1; iarg++)
        {
          if(cyc[iarg] != BAD)
            {
              strcpy(str,argv[iarg+1]);
              strcat(str,"_r");
              if ((ofile = fopen(str, "w")) == NULL)
                {
                  fprintf(stderr,"%s: Unable to open file %s. Continuing...\n",
                          argv[0], str);
                  continue;     /* Try next file.      */
                }

              for (i=0; i<ipoint; i++)
                {
                  if((h_tmp[iarg][i] != BAD) && (mean2[i] != BAD))
                    {
                      htmp = (h_tmp[iarg][i]-mean2[i])*MILLI;
                    }
                  else
                    {
                      htmp = BAD*MILLI;
                    }
                  fit0 = (quad0[iarg][0] + (quad0[iarg][1]+quad0[iarg][2]*lat[i])
                    *lat[i])*MILLI;
                  fit1 = (quad1[iarg][0] + (quad1[iarg][1]+quad1[iarg][2]*lat[i])
                    *lat[i])*MILLI;
                  fprintf(ofile,"%4d\t%8.4f\t%8.4f\t%8.4f\t%8.4f\t%8.4f\n", i,
                          lat[i]*MICRO, mean_lon[i]*MICRO, htmp, fit0, fit1);
                }
```

```
            fclose(ofile);
          }

      }
/*
  Write out statistics to standard output.
  The output is:

  i     lat     mean_lon        mean2   var     var2  count2

  Where i is the sequential point number, lat is the latitude
  at that point, mean_lon is the mean_lon, mean2 is the mean height
  with orbit error etc removed, var is the variance, var2 is the sum
  of squares and count2 is the number of cycles that went into the
  statistics...
*/
  for (i=0; i<ipoint; i++)
    {
      if (mean2[i] != BAD)
        {
          vtmp = sqrt(var[i])*MILLI;
        }
      else
        {
          vtmp = BAD*MILLI;
        }

      fprintf(stdout,"%4d\t%8.4f\t%8.4f\t%8.4f\t%8.4f\t%12.4f\t%3d\n",
              i, lat[i]*MICRO, mean_lon[i]*MICRO,  mean2[i]*MILLI,
              vtmp, var2[i], count2[i]);
    }
  fclose(str);

}

/*-----------------------------------------------------------------
  Subprogram calc_mean1()

  Written by: Michael Caruso
              Woods Hole Oceanographic Institution

  Purpose:  This subprogram is used with g_repeat
            to calculate the means of input data.
-------------------------------------------------------------------*/
calc_mean1()
{
  maxcyc = 0;

  for (i=0; i<ipoint; i++)
    {
      if (count[i] > 1)         /* Check for at least two good points. */
        {
          mean[i] /= count[i];
          mean_lon[i] /= count[i];
```

```
          var[i] = var[i]/count[i] - mean[i]*mean[i];
          maxcyc = (count[i] > maxcyc) ? count[i] : maxcyc;
        }
      else
        {
          mean[i] = BAD;
          var[i] = BAD;
        }
    }
}

/*-----------------------------------------------------------------------
  Subprogram calc_mean2()

  Written by: Michael Caruso
              Woods Hole Oceanographic Institution

  Purpose:  This subprogram is used with g_repeat
            to calculate the means of temp data.
-----------------------------------------------------------------------*/
calc_mean2()
{

  for (i=0; i<ipoint; i++)
    {
      if (count2[i] > 1)        /* Check for at least two good points */
        {
          mean2[i] /= count2[i];
          var[i] = var2[i]/count2[i] - mean2[i]*mean2[i];
        }
      else
        {
          mean2[i] = BAD;
          var[i] = BAD;
        }
    }
}

/*-----------------------------------------------------------------------
  Subprogram fit_quad()

  Written by: Michael Caruso
              Woods Hole Oceanographic Institution

  Purpose:  This subprogram is used with g_repeat
            to fit a quadratic to an arc.

  Method: fit a quadratic in a least squares sense..

        num     x       xx
  a =   x       xx      xxx
        xx      xxx     xxxx

        y    - z
```

```
  b =   xy  - xz
        xxy - xxz

---------------------------------------------------------------------*/
fit_quad(numcyc, pass)
     int numcyc;
     int pass;
{
  int iarg;
  double x, y, z, xx, xy, xz, xxx, xxy, xxz, xxxx;
  double t, t2, t3, t4;
  int err;
  int num;

  for (iarg=0; iarg<numcyc-1; iarg++)
    {
      if(cyc[iarg] != BAD)
        {
          /*
            Declare dummy arrays for procedure...
            And set to zero.
            */
          num = 0;
          x = y = z = xx = xy = xz = xxx = xxy = xxz = xxxx = 0.0;

          for (i=0; i<ipoint; i++)
            {
              if (h[iarg][i] != BAD)
                {
                  if (count[i] >= maxcyc/2)
                    {
                      if (!pass)
                        {
                          t     = lat[i];
                          t2    = t*t;
                          t3    = t2*t;
                          t4    = t3*t;
                          x     += t;
                          y     += h[iarg][i];
                          z     += mean[i];
                          xx    += t2;
                          xy    += t*h[iarg][i];
                          xz    += t*mean[i];
                          xxx   += t3;
                          xxy   += t2*h[iarg][i];
                          xxz   += t2*mean[i];
                          xxxx  += t4;
                          num++;
                        }
                      else
                        {
                          if (var[i] != BAD)
                            {
```

```
                        t       = lat[i];
                        t2      = t*t;
                        t3      = t2*t;
                        t4      = t3*t;
                        x       += t/var[i];
                        y       += h[iarg][i]/var[i];
                        z       += mean[i]/var[i];
                        xx      += t2/var[i];
                        xy      += t*h[iarg][i]/var[i];
                        xz      += t*mean[i]/var[i];
                        xxx     += t3/var[i];
                        xxy     += t2*h[iarg][i]/var[i];
                        xxz     += t2*mean[i]/var[i];
                        xxxx    += t4/var[i];
                        num++;
                      }

                    }
                  }
                }
              }
/*
  If we have less than three points, label bad cycle
  */
          if (num < 3)
            {
              cyc[iarg] = BAD;
              continue;
            }

/*
  Set up matrices and solve x*xans=b...
*/
          a[0][0] = num;
          a[0][1] = a[1][0] = x;
          a[0][2] = a[1][1] = a[2][0] = xx;
          a[1][2] = a[2][1] = xxx;
          a[2][2] = xxxx;
          b[0] = y-z;
          b[1] = xy-xz;
          b[2] = xxy-xxz;

          err = gauss_elim(a, 3, 3, b, xans);

  /*
    Save fit parameters for later printing...
    */

          for (i=0; i<3; i++)
            {
              if(pass == 0)
                {
                  quad0[iarg][i] = xans[i];
                }
```

```
            else
              {
                quad1[iarg][i] = xans[i];
              }
          }

  /*
    Subtract regression from each cycle...
    Put result into h_tmp and keep track of
    data for mean and variance calculation - calc_mean2.
    */

        for(i=0; i<ipoint; i++)
          {
            if(h[iarg][i] != BAD)
              {
                h_tmp[iarg][i] = h[iarg][i]-xans[0]-
                                 (xans[1]+xans[2]*lat[i])*lat[i];
                mean2[i] += h_tmp[iarg][i];
                var2[i] += h_tmp[iarg][i]*h_tmp[iarg][i];
                count2[i] += 1;
              }
            else
              {
                h_tmp[iarg][i] = BAD;
              }
          }
      }
    }
}
/*------------------------------------------------------------------------
  Subprogram zero_2()

  Written by: Michael Caruso
              Woods Hole Oceanographic Institution

  Purpose:  This subprogram is used with g_repeat
            to zero mean2 var2 and count2
--------------------------------------------------------------------------*/
zero_2()
{
  for (i=0; i<ipoint; i++)
    {
      mean2[i] = var2[i] = 0.0;
      count2[i] = 0;
    }
}
```

Program g_repeats.c

```
/*
 @(#)g_repeats.c       1.2 4/25/90

 Written by:
 --------------
 Michael Caruso
 Woods Hole Oceanograhic Institution
 Woods Hole, MA


 Purpose:
 --------
 This program will perform a repeat track analysis
 of geosat GDR's.

 Method:
 -------
 1.    Read in all cycles
 2.    Calculate the mean sea surface height (m_h)
 3.    Subtract mean from each cycle.
 4.    Calculate sine regression and subtract from
       each cycle.
 5.    Calculate a second regression weighted by the
       inverse of the variance of the first regression.
 6.    Subtract new fit from each profile to obtain
       final heights.
 7.    Calculate mean and variance.
 8.    Print results.

 Usage:
 ------
 Each GEOSAT GDR is read from a separate file.

 g_repeat c???.a002 > data.text

 will perform a repeat track analysis from the cleaned and splined
 GDR's in all available cycle for track a002.

 Input:
 ------
 c???.a002      All cycles for the specified track cleaned
                and splined.

 Output:
 -------
 Stdout:        Data file containing the latitude, the
                longitude, the mean height(m), the rms height
                variance and the number of valid points for each
                location of a given orbit.
```

```
  c???.a002_r   Data file containing the latitude, the
                longitude and the residual height.

  Subroutines Required:
  --------------------
  cr_mat_float          creates a floating point matrix.
  cr_mat_double         creates a double precision matrix.
  gauss_elim            solves linear system of equations.

  References:
  -----------

*/

# include <math.h>
# include <stdio.h>
# include "geos.h"

#define MICRO           1.e-6   /* conversion from micro-deg to deg          */
#define MILLI           1.e-2   /* conversion from centimeter to meter       */
#define MAXPOINTS       2915    /* Max point -70 to +70  degrees latitude    */

#define DEG_TO_RAD      M_PI/180.0      /* Converstion to radians            */
#define OM              2.0*M_PI/PERIOD /* Orbital Omega                     */
#define M_2PI           2.0*M_PI        /* 2 * PI                            */
/*
  Global Variables...
*/

float **h;              /* Original Heights. */
float **h_tmp;          /* Temporary Heights. */
double **times;         /* Array of times for each GDR  */

float *mean;            /* Mean of original heights (h) */
float *mean2;           /* Mean of original heights - quad orbit corr.*/
float *mean_lon;        /* Mean of Longitudes */
float *lat;             /* Latitudes of first orbit. */
                        /* We don't find mean_lat because the
                           latitudes are fixed in a previous
                           program such as g_spline
                           */

float *var;             /* Variance of original heights */
float *var2;

int *count;             /* Count for mean and var*/
int *count2;            /* Count for mean2 and var2*/

double **a, *b;         /* Used for quadratic fit*/
double *xans;           /* Used for quadratic fit */
float **quad0;          /* Keep values of quadratic fit for printing */
float **quad1;          /* Keep values of quadratic fit for printing */
```

```
int *cyc;                   /* Keep track of good and bad cycles. */
int maxcyc;                 /* Maximum number of good points at each
                               latitude grid point
                               */

int     i;                  /* Counter */
int     ipoint;             /* Counter for points read in for each cycle.
                               Note: program assumes each cycle has been
                               regridded to a common grid and has the same
                               number of points.
                               */


main (argc,argv)
     int argc;
     char *argv[];
{

/*
  Declare non-integer subroutines:
*/

  float **cr_mat_float();
  double **cr_mat_double();

/*
  I/O file descriptors
*/

  FILE *gfile, *ofile;

/*
  Various counters.
*/

  int iarg;
  int ipointold;
  int fill = FALSE;

/*
  Temporary variables for calculations prior to printing.
*/
  float htmp, vtmp;
  float fit0, fit1;
  float t;

/*
  Misc. variables.
*/
  int err;                  /* Returned error message.      */
  char str[80];             /* String for filenames etc. */


/*
```

```
  Create arrays described above...
*/
  h             = cr_mat_float(argc, MAXPOINTS);
  h_tmp         = cr_mat_float(argc, MAXPOINTS);
  times         = cr_mat_double(argc, MAXPOINTS);

  mean          = (float *)calloc(MAXPOINTS, sizeof(float));
  mean2         = (float *)calloc(MAXPOINTS, sizeof(float));
  mean_lon      = (float *)calloc(MAXPOINTS, sizeof(float));
  lat           = (float *)calloc(MAXPOINTS, sizeof(float));

  var           = (float *)calloc(MAXPOINTS, sizeof(float));
  var2          = (float *)calloc(MAXPOINTS, sizeof(float));

  count         = (int *)calloc(MAXPOINTS, sizeof(int));
  count2        = (int *)calloc(MAXPOINTS, sizeof(int));

  cyc           = (int *)calloc(argc, sizeof(int));

  a             = cr_mat_double(3,3);
  b             = (double *)calloc(3, sizeof(double));
  xans          = (double *)calloc(3, sizeof(double));
  quad0         = cr_mat_float(argc,3);
  quad1         = cr_mat_float(argc,3);

/*
  Check count2 and quad1 to see if they were allocated space.  If so,
  assume that all other arrays were allocated ok.
  */

  if ((quad1 == NULL) || (count2 == NULL))
    {
      fprintf(stderr,"%s: Unable to allocate enough storage space.\n",
              argv[0]);
      exit(1);
    }

/*
  Check to see if program is given arguments...
*/
  if (argc == 1)
    {
      fprintf(stderr,"Usage: %s c???.a000 > fileout.text\n",argv[0]);
      exit(1);
    }


  /*
    Read in all GDRs and calculate mean and variance...
    iarg        is the cycle number to read in.
    ipoint      is the along track point
    */

  for(iarg=0; iarg<argc-1; iarg++)
```

```
{
  /*
    Open each input file...
    */
  if ((gfile = fopen(argv[iarg+1],"r")) == NULL)
    {
      fprintf(stderr,"%s: Unable to open file %s. Continuing...\n",
              argv[0], argv[iarg+1]);
      cyc[iarg] = BAD;
      continue;  /* If there is no file, try the next file. */
    }

  ipoint = 0;

  while (fread((char*)&fr,1,REC_LEN,gfile)==REC_LEN)
    {
      if (!fill)    /* Fill latitude from first good cycle. */
        {
          lat[ipoint] = fr.lat;
        }
      else                  /* Check against first cycle */
        {                   /* to make sure points line up */
          if((abs((int)lat[ipoint]-fr.lat) > 1000) && (fr.lat != 0))
            {
              fprintf(stderr,
                "%s: Repeat tracks out of sync.  Offending file: %s\n",
                argv[0], argv[iarg+1]);
              exit(1);
            }
        }

      /*
        Store Heights and times...
        Note: Subtract time zero to keep times small.
        */

      h[iarg][ipoint]     = fr.m_h;
      times[iarg][ipoint] = (fr.utc+fr.utcm*MICRO) - TIME_ZERO;


      /*
        Set up to find mean and variances...
        */

      if ((fr.m_h != BAD) && (fr.lon !=BAD))
        {
          mean[ipoint] += fr.m_h;
          var[ipoint]    += fr.m_h*fr.m_h;
          count[ipoint] += 1;
          mean_lon[ipoint] += fr.lon;
        }
      ipoint++;
    }
```

```
/*
  If a cycle has no points, mark that cycle as BAD and
  print error message.
*/
      if(ipoint == 0)
        {
          cyc[iarg] = BAD;
          ipoint = ipointold;
          fprintf(stderr,"%s: Bad file %s, no data found\n",argv[0],
            argv[iarg+1]);
        }
      else
        {
          ipointold = ipoint;
          fill = TRUE;
        }

    }

/*
  If no points were read, exit program...
*/
  if ((ipoint == 0) && (ipointold == 0))
    {
      fprintf(stderr,"%s: No points read, unable to perform analysis.\n",
        argv[0]);
      exit(1);
    }
  ipoint--;

/*
  Find mean and variance of raw data.  Also find the mean lon
  at each point and determine the maximum number of cycles.
  mean, var, mean_lon and maxcyc.
*/

  calc_mean1();

/*
  Calculate quadratic regression of difference and subtract
  from each cycle...
*/

  fit_sin(argc, 0);


/*
  Find mean and var of h_tmp.
  mean2 and var.
*/
  calc_mean2();

/*
  Zero mean2, var2 and count2 and
```

```
  Calculate weighted regression...
*/


  zero_2();
  fit_sin(argc, 1);

/*
  Find mean and var of h_tmp after weighted
  regresssion - mean2 and var.
*/

  calc_mean2();

/*
  Write out residuals for each good cycle.
  Concatenate "_r" to the end of the file name.  This was
  done instead of substitution since the user may call the
  program with subdirectories - c000/c000.a002c, in which
  case a prefix would change the directory name; or if the
  input file does not end with an additional character -
  c000.a002, substituting the last character would affect
  the file name.

  Output file format:

  lat   lon    residual       fit0    fit1

  Where the residual is h - hmean, and fit0 is
  the resulting fit of the first quadratic and fit1
  is the fit of the second quadratic.
*/

      for (iarg=0; iarg<argc-1; iarg++)
        {
          if(cyc[iarg] != BAD)
            {
              strcpy(str,argv[iarg+1]);
              strcat(str,"_r");
              if ((ofile = fopen(str, "w")) == NULL)
                {
                  fprintf(stderr,"%s: Unable to open file %s. Continuing...\n",
                          argv[0], str);
                  continue;     /* Try next file.       */
                }

              for (i=0; i<ipoint; i++)
                {
                  if((h_tmp[iarg][i] != BAD) && (mean2[i] != BAD))
                    {
                      htmp = (h_tmp[iarg][i]-mean2[i])*MILLI;
                      t = remainder(OM*times[iarg][i], M_2PI);
                      fit0 = (quad0[iarg][0] + quad0[iarg][1]*cos(t) +
                              quad0[iarg][2]*sin(t))*MILLI;
```

```
                    fit1 = (quad1[iarg][0] + quad1[iarg][1]*cos(t) +
                            quad1[iarg][2]*sin(t))*MILLI;
                  }
                else
                  {
                    htmp = BAD*MILLI;
                    fit0 = fit1 = BAD;
                  }
                fprintf(ofile,"%4d\t%8.4f\t%8.4f\t%8.4f\t%8.4f\t%8.4f\n", i,
                        lat[i]*MICRO, mean_lon[i]*MICRO, htmp, fit0, fit1);
              }
            fclose(ofile);
          }

      }
/*
  Write out statistics to standard output.
  The output is:

  i     lat     mean_lon        mean2   var     var2  count2

  Where i is the sequential point number, lat is the latitude
  at that point, mean_lon is the mean_lon, mean2 is the mean height
  with orbit error etc removed, var is the variance, var2 is the sum
  of squares and count2 is the number of cycles that went into the
  statistics...
*/
  for (i=0; i<ipoint; i++)
    {
      if (mean2[i] != BAD)
        {
          vtmp = sqrt(var[i])*MILLI;
        }
      else
        {
          vtmp = BAD*MILLI;
        }

      fprintf(stdout,"%4d\t%8.4f\t%8.4f\t%8.4f\t%8.4f\t%12.4f\t%3d\n",
              i, lat[i]*MICRO, mean_lon[i]*MICRO,  mean2[i]*MILLI,
              vtmp, var2[i], count2[i]);
    }
  fclose(str);

}

/*--------------------------------------------------------------------
  Subprogram calc_mean1()

  Written by: Michael Caruso
              Woods Hole Oceanographic Institution

  Purpose:  This subprogram is used with g_repeat
            to calculate the means of input data.
```

```
-----------------------------------------------------------------*/
calc_mean1()
{
  maxcyc = 0;

  for (i=0; i<ipoint; i++)
    {
      if (count[i] > 1)          /* Check for at least two good points. */
        {
          mean[i] /= count[i];
          mean_lon[i] /= count[i];
          var[i] = var[i]/count[i] - mean[i]*mean[i];
          maxcyc = (count[i] > maxcyc) ? count[i] : maxcyc;
        }
      else
        {
          mean[i] = BAD;
          var[i] = BAD;
        }
    }
}

/*-----------------------------------------------------------------
  Subprogram calc_mean2()

  Written by: Michael Caruso
              Woods Hole Oceanographic Institution

  Purpose:  This subprogram is used with g_repeat
            to calculate the means of temp data.
-----------------------------------------------------------------*/
calc_mean2()
{

  for (i=0; i<ipoint; i++)
    {
      if (count2[i] > 1)         /* Check for at least two good points */
        {
          mean2[i] /= count2[i];
          var[i] = var2[i]/count2[i] - mean2[i]*mean2[i];
        }
      else
        {
          mean2[i] = BAD;
          var[i] = BAD;
        }
    }
}

/*-----------------------------------------------------------------
  Subprogram fit_sin()

  Written by: Michael Caruso
              Woods Hole Oceanographic Institution
```

```
  Purpose:  This subprogram is used with g_repeat
            to fit a sin to an arc.

  Method: fit a sin in a least squares sense..

        num      x       xx
  a =   x        xxx     xxxx
        xx       xxxx    xxxxx

        y   - z
  b =   xy  - xz
        xxy - xxz

-------------------------------------------------------------------*/
fit_sin(numcyc, pass)
     int numcyc;
     int pass;
{

  double x, y, z, xx, xy, xz, xxx, xxy, xxz, xxxx, xxxxx;
  double t;              /* Variable to fit to */
  double sin_t, cos_t;   /* Sin(t), Cos(t) */

  int iarg;
  int err;
  int num;

  for (iarg=0; iarg<numcyc-1; iarg++)
    {
      if(cyc[iarg] != BAD)
        {
          /*
            Declare dummy arrays for procedure...
            And set to zero.
            */
          num = 0;
          x = y = z = xx = xy = xz = xxx = xxy = xxz = xxxx = xxxxx = 0.0;

          for (i=0; i<ipoint; i++)
            {
              if (h[iarg][i] != BAD)
                {
                  if (count[i] >= maxcyc/2)
                    {
                      if (!pass)
                        {
                          t              = remainder(OM*times[iarg][i], M_2PI);
                          cos_t          = cos(t);
                          sin_t          = sin(t);
                          x              += cos_t;
                          y              += h[iarg][i];
                          z              += mean[i];
                          xx             += sin_t;
```

```
                    xy          += cos_t*h[iarg][i];
                    xz          += cos_t*mean[i];
                    xxx         += cos_t*cos_t;
                    xxy         += sin_t*h[iarg][i];
                    xxz         += sin_t*mean[i];
                    xxxx        += cos_t*sin_t;
                    xxxxx       += sin_t*sin_t;
                    num++;
                  }
                else
                  {
                    if (var[i] != BAD)
                      {
                        t                 = remainder(OM*times[iarg][i],
                                                      M_2PI);
                        cos_t             = cos(t);
                        sin_t             = sin(t);
                        x                 += cos_t/var[i];
                        y                 += h[iarg][i]/var[i];
                        z                 += mean[i]/var[i];
                        xx                += sin_t/var[i];
                        xy                += cos_t*h[iarg][i]/var[i];
                        xz                += cos_t*mean[i]/var[i];
                        xxx               += cos_t*cos_t/var[i];
                        xxy               += sin_t*h[iarg][i]/var[i];
                        xxz               += sin_t*mean[i]/var[i];
                        xxxx              += cos_t*sin_t/var[i];
                        xxxxx             += sin_t*sin_t/var[i];
                        num++;
                      }

                  }
              }
          }
      }
/*
  If we have less than three points, label bad cycle
  */
        if (num < 3)
          {
            cyc[iarg] = BAD;
            continue;
          }

/*
  Set up matrices and solve x*xans=b...
*/
        a[0][0] = num;
        a[0][1] = a[1][0] = x;
        a[0][2] = a[2][0] = xx;
        a[1][1] = xxx;
        a[1][2] = a[2][1] = xxxx;
        a[2][2] = xxxxx;
        b[0] = y-z;
```

```
        b[1] = xy-xz;
        b[2] = xxy-xxz;

        err = gauss_elim(a, 3, 3, b, xans);

    /*
      Save fit parameters for later printing...
      */

        for (i=0; i<3; i++)
          {
            if(pass == 0)
              {
                quad0[iarg][i] = xans[i];
              }
            else
              {
                quad1[iarg][i] = xans[i];
              }
          }

    /*
      Subtract regression from each cycle...
      Put result into h_tmp and keep track of
      data for mean and variance calculation - calc_mean2.
      */

        for(i=0; i<ipoint; i++)
          {
            if(h[iarg][i] != BAD)
              {
                t = remainder(OM*times[iarg][i], M_2PI);
                h_tmp[iarg][i] = h[iarg][i]-xans[0]-xans[1]*cos(t)-
                                 xans[2]*sin(t);
                mean2[i] += h_tmp[iarg][i];
                var2[i] += h_tmp[iarg][i]*h_tmp[iarg][i];
                count2[i] += 1;
              }
            else
              {
                h_tmp[iarg][i] = BAD;
              }
          }
      }
    }
}
/*------------------------------------------------------------------
  Subprogram zero_2()

  Written by: Michael Caruso
              Woods Hole Oceanographic Institution

  Purpose:  This subprogram is used with g_repeat
            to zero mean2 var2 and count2
```

```
---------------------------------------------------------------------*/
zero_2()
{
  for (i=0; i<ipoint; i++)
    {
      mean2[i] = var2[i] = 0.0;
      count2[i] = 0;
    }
}
```

Program g_seporb.c

```
/*
  @(#)g_seporb.c        1.4 6/14/90

  Written by:
  -----------
  Michael Caruso
  Woods Hole Oceanographic Institution
  Woods Hole, MA

  Modifications:
  --------------
  12-15-89  MC Changed call to geo_cyc_orb so that time is
            passed and not a GDR.

  Purpose:
  --------
  Read raw GEOSAT GDR and splits data into separate orbits.  Each
  orbit is defined as beginning at the northernmost point of a track.
  Each orbit is further separated into an ascending section and a descending
  section.  Orbits are then written out to separate files of the form:

               cmmm.annn or cmmm.nnnd
        where
               mmm              is the cycle number,
               nnn              is the orbit number for that cycle,
               a                signifies ascending portion,
               d                signifies descending portion.

  The data is written out in the same form as it was read in.  This is
  consecutive records of 78 bytes each.  The last point of an ascending
  or descending orbit is the most northern or most southern point of
  that orbit.

  Method:
  -------
  Reads raw GEOSAT GDR from standard input.  Calculates correct filename
  using convention shown above.  Tests to see when slope of lat/lon track
  changes sign which indicates change from ascending to descending or
  descending to ascending part of orbit.

  Usage:
  ------
  The data is read from standard input such as direct from the NODC
  data tapes:

  dd if=/dev/rmt8 ibs=16380 files=34 | g_seporb

  This would separate all the files into the correct orbits.
  NOTE: input file must have at least two points.
```

```
  Input:
  -----
  Stdin         Raw Geosat GDRs.

  Output:
  -------
  c???.????             Separated GEOSAT data.

  Subroutines Required:
  ---------------------
  geo_cyc_orb   Returns the cycle number and orbit number
                for a given time.
  geo_error     Prints error messages.

*/

#include <stdio.h>
#include <sys/file.h>
#include <math.h>
#include "geos.h"

main(argc, argv)
     int argc;
     char *argv[];
{

  struct frame fr2;

  char str[80];

  short int orbit_num_tot;
  int cycle_num;
  int orbit_num;
  long int lslope1, lslope2;


  double time;

  FILE *fdout;


  /* read initial data record... */

  if(fread((char *)&fr, 1, REC_LEN, stdin) != REC_LEN)
    {
      geo_error(3, argv[0]);
      exit(1);
    }

/* read second data record... */

  if(fread((char *)&fr2, 1, REC_LEN, stdin) != REC_LEN)
    {
```

```
      geo_error(3, argv[0]);
      exit(1);
    }

  /* Determine name of first orbit */

  time = fr.utc + fr.utcm*1.0e-6;
  lslope1 = fr2.lat - fr.lat;

  geo_cyc_orb(time, &cycle_num, &orbit_num);

  if ( lslope1 > 0 )    /* If lslope1 > 0 ascending orbit else descending */
    sprintf(str,"c%.3d.a%.3d",cycle_num,orbit_num);
  else if ( lslope1 < 0)
    sprintf(str,"c%.3d.d%.3d",cycle_num,orbit_num);
  else
    {
      if (fr.lat < 0)
        {
          sprintf(str,"c%.3d.a%.3d",cycle_num,orbit_num);
        }
      else
        {
          sprintf(str,"c%.3d.d%.3d",cycle_num,orbit_num);
        }
    }
  fdout = fopen(str,"a");       /* open a new file or append an old */


/* Write out first two records to opened file... */

  if(fwrite((char *)&fr, 1, REC_LEN, fdout) != REC_LEN)
    {
      geo_error(3, argv[0]);
      exit(3);
    }

  if(fwrite((char *)&fr2, 1, REC_LEN, fdout) != REC_LEN)
    {
      geo_error(3, argv[0]);
      exit(3);
    }
  /* Read in rest of geosat data */

  fr = fr2;

  while(fread((char *)&fr2, 1, REC_LEN, stdin) == REC_LEN)
    {
      lslope2 = fr2.lat - fr.lat;

      if ((lslope1 > 0 && lslope2 <= 0) || (lslope1 < 0 && lslope2 >= 0))
        {
          /* Determine name of next orbit */
```

```
        time = fr2.utc + fr2.utcm * 1.0e-6;

        geo_cyc_orb(time, &cycle_num, &orbit_num);

        if ( lslope2 > 0 )
          sprintf(str,"c%.3d.a%.3d",cycle_num,orbit_num);
        else if ( lslope2 < 0)
          sprintf(str,"c%.3d.d%.3d",cycle_num,orbit_num);
        else
          {
            if (fr.lat < 0)
              {
                sprintf(str,"c%.3d.a%.3d",cycle_num,orbit_num);
              }
            else
              {
                sprintf(str,"c%.3d.d%.3d",cycle_num,orbit_num);
              }
          }
        fclose(fdout);                  /* Close previous file */
        fdout = fopen(str,"a");         /* Open new or append file*/
      }

    if(fwrite((char *)&fr2, 1, REC_LEN, fdout) != REC_LEN)
      {
        geo_error(3, argv[0]);
        exit(3);
      }

    fr      = fr2;
    lslope1 = lslope2;

  }

}
```

Program g_spike.c

```
/*
  @(#)g_spike.c 1.4 11/14/89

  Program g_spike.c

  Written by:
  --------------
  Michael Caruso
  Woods Hole Oceanograhic Institution
  Woods Hole, MA


  Purpose:
  --------
  This program will remove spikes from data by a series
  of quadratic fits.

  Method:
  -------
  1.    Break data into continuous segments
  2.    Fit each segment with a quadratic
  3.    If fit is within outlier, keep point
        else, remove two worst points and
        fit segment with another quadratic.
  4.    If fit is within outlier, keep point
        else, calculate a linear fit.
  5.    If linear fit is within outlier, keep
        point.  Else reject point as bad
  6.    Delete points that do not have enough
        neighbors and segments that do not have
        enough points.

  Usage:
  ------
  The GEOSAT GDR is read from standard input.

  cat c000.a002 | g_spike deltmax neighbors outlier > c000.a002s

  will remove the spike records from the GDR in c000.a002.


  Input:
  ------
  Stdin         Clean GEOSAT GDRs
  deltmax       Seconds that define gap in record
  neighbors     Number of neighbors required for spline.
  outliers      Centimeters for rejection of point after
                spline fit.
```

```
  Output:
  -------
  Stdout:       GEOSAT GDRs with spikes removed.

  Subroutines Required:
  ---------------------
  gauss_elim    Solves linear equation Ax=b by Gaussian Elimination.

  Subprograms Required:
  --------------------
  comp_fit_quad Computes quadratic least squares fit.
  comp_fit_lin  Computes linear least squares fit.
  find_bad_pts  Finds two worst points in segment.
  write_gdr     Writes a single GDR to stdout or prints error.

  References:
  -----------

*/
# include <math.h>
# include <stdio.h>
# include "geos.h"

#define NUMARGS         3
#define DARG            1
#define NARG            2
#define OARG            3
#define MICRO           1e-6            /* Conv for utcm                  */
#define BADFIT          -99999.0        /* Return if least sq malformed */

#define SWAP(u,v) {(xswap)=(u);(u)=(v);(v)=(xswap);}

double **a, *b;                 /* Arrays for quad fit */
double **a1, *b1;               /* Arrays for linear fit */
double *xans, *xans1;           /* Results for quad and linear fit */
struct frame frs[3000];         /* GDRs */
short int kstart, kstop;         * Start and stop for each fit */
double max0, max1;                 Two worst points to remove */
int imax0, imax1;               /*   if first fit is bad      */
double *h_fit;                  /* Temp array of fits for find_bad_pts */
double t;                       /* Temp time variable */

int num;                        /* Number of points in fit segment */
double xswap;                   /* Temp swap variable              */


main (argc,argv)
     int argc;
     char *argv[];
{


/*
  Declare subroutines and subprograms...
```

```
*/
  double        **cr_mat_double();
  double        comp_fit_quad();
  double        comp_fit_lin();
  int           find_bad_pts();
  int           write_gdr();

  int err;                      /* error returned by gauss_elim */
  int neighbors;

  double time1, time2;          /* Read in times for gap determination */
  double h_new;                 /* Calculated height */


  float deltmax;                /* Maximum time for gap */
  float outlier;                /* Maximum offset for rejected point */



  short int seg_len[1000];      /* defines each segment */
  short int seg_beg[1000];      /*                      */

  short int points=0;           /* Counters     */
  short int segments = 0;       /*              */
  short int i, j, k, ii;        /*              */


/*
  Check arguments...
*/

  if ((argc != NUMARGS+1) && (argc != 1))
    {
      fprintf(stderr,"Usage: %s [deltmax deighbors outlier] < filein > filout\n",
        argv[0]);
      exit(1);
    }
  if (argc == NUMARGS+1)
    {
      sscanf(argv[DARG],"%f", &deltmax);
      sscanf(argv[NARG],"%d", &neighbors);
      sscanf(argv[OARG],"%f", &outlier);
    }
  else
    {
      deltmax   = 3.3;
      neighbors = 13;
      outlier   = 50.0;
    }

  /*
    set up matrices for least squares...
    */
  a     = cr_mat_double(3,3);
```

```
a1    = cr_mat_double(2,2);
b     = (double *)calloc(3, sizeof(double));
b1    = (double *)calloc(2, sizeof(double));
xans  = (double *)calloc(3, sizeof(double));
xans1 = (double *)calloc(2, sizeof(double));

h_fit = (double *)calloc(neighbors, sizeof(double));


/*
  Read in all GDRs...
  */
while (fread((char*)&frs[points],1,REC_LEN,stdin)==REC_LEN) points++;
points--;


/*
  1. Break into data segments...
  */
seg_len[0] = 1;
seg_beg[0] = 0;
time2 = frs[0].utc + frs[0].utcm*MICRO;
for (i=0; i<points; i++)
  {

    time1 = time2;
    time2 = frs[i+1].utc + frs[i+1].utcm*MICRO;

    if ((time2 - time1) <= deltmax)
      {
        seg_len[segments]++;
      }
    else
      {
        segments++;
        seg_beg[segments] = i + 1;
        seg_len[segments] = 1;
      }
  }

if (seg_len[0] == 0)  exit(1);

/*
  2. For each segment...
  */
for (i=0; i<=segments; i++)
  {
    /*
      2a. Ignore segment if there are too few
      points...
      */
    if (seg_len[i] < neighbors)      continue;
    /*
      2b. Fit a quadratic through each point...
```

```
        */
      else
        {
          for (j=seg_beg[i]; j<seg_beg[i]+seg_len[i]; j++)
            {
              kstart = j - neighbors/2;
              kstop  = kstart + neighbors;
              imax0 = imax1 = -1;

              h_new = comp_fit_quad(j);

/*
  3. If fit is tolerable write out point...
*/
              if((fabs(h_new-frs[j].m_h) < outlier)
                 && (h_new != BADFIT))
                {
                  write_gdr(j, argv[0]);
                }
/*
  3a. Else remove two worst points and recompute fit...
      If fit is malformed, remove first and last point and recompute fit...
*/
              else
                {
                  if (h_new == BADFIT)
                    {
                      imax0 = 0;
                      imax1 = num - 1;
                    }
                  else
                    find_bad_pts();

                  /*
                    Recompute fit...
                    */

                  h_new = comp_fit_quad(j);

/*
  4. If fit is tolerable write out point...
  */

                  if((fabs(h_new-frs[j].m_h) < outlier)
                     && (h_new != BADFIT))
                    {
                      write_gdr(j, argv[0]);
                    }
                  else
                    {

                      h_new = comp_fit_lin(j);
                      t = frs[j].utc + frs[j].utcm*MICRO;
```

```
/*
  5. If fit is tolerable write out point...
  */
                    if((fabs(h_new-frs[j].m_h) < outlier)
                       && (h_new != BADFIT))
                      {
                        write_gdr(j, argv[0]);
                      }
                    else
                      {
                        fprintf(stderr,"Rejected point -> \tTime: \t%lf \n",t);
                        fprintf(stderr,"\t\t\tOriginal h: \t%d\n", frs[j].m_h);
                        fprintf(stderr,"\t\t\tComputed h: \t%7.1lf\n", h_new);
                      }
                  }

              }

          }
        }

    }
}


/*--------------------------------------------------------------
  subprogram comp_fit_quad.

  Written by:  Michael Caruso
               Woods Hole Oceanographic Institution

  Purpose:  This subprogram is used with g_spline to
            compute a quadratic fit.
---------------------------------------------------------------*/

double
  comp_fit_quad(j)
int j;
{
  double x, y, xx, xy, xxx, xxy, xxxx;
  double h_new;
  double t, t2, t3, t4;
  int k;
  int err;

  num = 0;
  x = y = xx = xy = xxx = xxy = xxxx = 0.0;


  for (k= kstart; k< kstop; k++)
    {
      if ((k != (kstart+imax0)) && (k != (kstart+imax1))
          && (k != j))
        {
```

```
            t  = frs[k].utc + frs[k].utcm*MICRO;
            t2 = t*t;
            t3 = t2*t;
            t4 = t3*t;
            x += t;
            y += frs[k].m_h;
            xx += t2;
            xy += t*frs[k].m_h;
            xxx += t3;
            xxy += t2*frs[k].m_h;
            xxxx += t4;
            num ++;
          }

        }

    a[0][0]         = num;
    a[0][1]         = a[1][0] = x;
    a[0][2]         = a[1][1] = a[2][0] = xx;
    a[1][2]         = a[2][1] = xxx;
    a[2][2]         = xxxx;
    b[0]            = y;
    b[1]            = xy;
    b[2]    = xxy;

    err = gauss_elim(a, 3, 3, b, xans);

    t = frs[j].utc + frs[j].utcm*MICRO;
    h_new = xans[0] + xans[1]*t + xans[2]*t*t;

    if (err == 0)
      return(h_new);
    else
      return(BADFIT);

  }

/*------------------------------------------------------------------
  Subprogram comp_fit_lin

  Written by:  Michael Caruso
               Woods Hole Oceanographic Institution

  Purpose:      This subprogram is used with g_spline to
                compute a linear least squares fit.
-------------------------------------------------------------------*/
double
  comp_fit_lin(j, prog)
int j;
char *prog;
{
  double x, y, xx, xy;
  double h_new, t, t2, t3, t4;
  int err;
```

```
  int k;

  x = y = xx = xy = 0.0;
  num = 0;

  for (k= kstart; k< kstop; k++)
    {
      if((k != (kstart+imax0)) && (k != (kstart+imax1))
         && (k != j))
        {
          t  = frs[k].utc + frs[k].utcm*MICRO;
          t2 = t*t;
          x += t;
          y += frs[k].m_h;
          xx += t2;
          xy += t*frs[k].m_h;
          num++;
        }
    }
  a1[0][0]      = num;
  a1[0][1]      = a1[1][0] = x;
  a1[1][1]      = xx;

  b1[0] = y;
  b1[1] = xy;

  err = gauss_elim(a1, 2, 2, b1, xans1);

  t = frs[j].utc + frs[j].utcm*MICRO;
  h_new = xans1[0] + xans1[1]*t;
  if (err == 0)
    return (h_new);
  else
    return (BADFIT);

}
/*-----------------------------------------------------------------
  Subprogram find_bad_pts

  Written by:   Michael Caruso
                Woods Hole Oceanographic Institution

  Purpose:
                This subprogram finds the two worst points
                in a series for program g_spike.
-------------------------------------------------------------------*/
find_bad_pts()
{
  short int ii;

  for (ii=0; ii<num; ii++)
    {
      t = frs[kstart+ii].utc + frs[kstart+ii].utcm*MICRO;
      h_fit[ii] = frs[kstart+ii].m_h - (xans[0]+xans[1]*t + xans[2]*t*t);
```

```
    }
  if (fabs(h_fit[0]) > fabs(h_fit[1]))
    {
      max0 = fabs(h_fit[0]);
      max1 = fabs(h_fit[1]);
      imax0 = 0;
      imax1 = 1;
    }
  else
    {
      max0 = fabs(h_fit[1]);
      max1 = fabs(h_fit[0]);
      imax0 = 1;
      imax1 = 0;
    }
  for (ii=2; ii<num; ii++)
    {
      if(fabs(h_fit[ii]) > max1)
        {
          max1 = fabs(h_fit[ii]);
          imax1 = ii;
        }
      if (max1 > max0)
        {
          SWAP(max1,max0);
          SWAP(imax1,imax0);
        }

    }
}
/*-------------------------------------------------------------
  Subprogram write_gdr

  Written by: Michael Caruso
              Woods Hole Oceanographic Institution

  Purpose:      This subprogram writes the selected GDR to
                standard output.

--------------------------------------------------------------*/
write_gdr(j, prog)
     int j;
     char *prog;
{
  if (fwrite((char *)&frs[j], 1, REC_LEN, stdout) != REC_LEN)
    {
      geo_error(3,prog);
      exit(3);
    }
}
```

Program g_spline.c

```
/*
 @(#)g_spline.c        1.3 12/15/89

 Program g_spline.c

 Written by:
 --------------
 Michael Caruso
 Woods Hole Oceanograhic Institution
 Woods Hole, MA

 Modifications:
 --------------
 12-15-89  MC Changed call to geo_cyc_orb so that time is
              passed and not a GDR.

 Purpose:
 --------
 This program will spline all geosat GDR data except the
 10 per second heights againts the latitude value.

 Method:
 -------
 1.    Break data into continuous segments
 2.    Fit each segment with a spline
 3.    Spline each data value.
 4.    Write out all data between min and max
 Usage:
 ------
 The GEOSAT GDR is read from standard input.

 cat c000.a002 | g_spline dir min max deltmax timestep> c000.a002s

 will spline the records from the GDR in c000.a002 between min
 and max latitude if dir is 1 and between min and max longitudes
 if dir is 2.


 Input:
 ------
 Stdin          Clean GEOSAT GDRs
 dir            Determines boundaries of spline:
                1       - Use latitude
                2       - Use longitude
 min            Minimum lat or lon.
 max            Maximum lat or lon.
 deltmax        Maximum gap in seconds for a contiguous
                segment.
 timestep       Interval between spline points.
```

```
  Output:
  -------
  Stdout:       Splined GEOSAT GDRs.  Values in gaps
                between valid segments are set to all
                zeros.

  Subroutines Required:
  ---------------------
  geo_cyc_orb   Returns the cycle number and orbit number
                for a given time.
  natcubspline  Computes a spline

  References:
  -----------

*/
# include <math.h>
# include <stdio.h>
# include "geos.h"

#define NUMARGS         5
#define DIARG           1
#define MIARG           2
#define MAARG           3
#define DEARG           4
#define TIARG           5
#define MICRO           1e-6    /* Conv for utcm                          */
#define MAXPOINTS       2915    /* +/-  degrees latitude                  */
/*
  Define some globals to share with fit subprograms...
*/

int  ii, j, k;
int latstart, latstop;
struct frame frsin[MAXPOINTS];  /* frame MAXPOINTS/2 is defined at the equator */
struct frame frout[MAXPOINTS];
float           timestep;
short int points=0;
float *x;
float *y2;
float *lat;
float *y;

float x0, minval, maxval;

/*
  Variables for orbit analysis.
*/
int orb, cyc;
double rs, rs3, cosinc, prec, rot, dthdt;
double time, theta, sinth, lat0, lon0, tmp;

short int seg_len[1000];
```

```
short int seg_beg[1000];

unsigned char ascorb;

main (argc,argv)
     int argc;
     char *argv[];
{

  int fit_time();
  int fit_data();


  double        time1, time2;   /* time variable used to determine gap  */

  float deltmax;

  short int i;

  short int dir;
  short int segments = 0;


  static char SccsId[] = "@(#)g_spline.c        1.4\t6/23/89";

  x = (float *)calloc(MAXPOINTS, sizeof(float));
  y2 = (float *)calloc(MAXPOINTS, sizeof(float));
  y = (float *)calloc(MAXPOINTS, sizeof(float));
  lat = (float *)calloc(MAXPOINTS, sizeof(float));


  if ((argc != NUMARGS+1) && (argc != 1))
    {
      fprintf(stderr,"Usage: %s [dir min max deltmax timestep] < filein > fileout\n",
            argv[0]);
      exit(1);
    }
  if (argc == NUMARGS+1)
    {
      sscanf(argv[DIARG],"%hd", &dir);
      sscanf(argv[MIARG],"%f", &minval);
      sscanf(argv[MAARG],"%f", &maxval);
      sscanf(argv[DEARG],"%f", &deltmax);
      sscanf(argv[TIARG],"%f", &timestep);
    }
  else
    {
      fprintf(stderr,"Usage: %s [dir min max deltmax timestep] < filein > fileout\n",
         argv[0]);
      exit(1);
    }


  /*
```

```
  Read in all GDRs...
  */
 while (fread((char*)&frsin[points],1,REC_LEN,stdin)==REC_LEN) points++;
 points--;

 if (points <= 0)
   {
     fprintf(stderr,"%s: No points to spline\n\n",argv[0]);
     exit(0);
   }

/*
 Determine if orbit is ascending or descending...
*/
 if((frsin[0].lat < frsin[1].lat) && (frsin[1].lat < frsin[points].lat))
   {
     ascorb = TRUE;
   }
 else if((frsin[0].lat > frsin[1].lat) && (frsin[1].lat > frsin[points].lat))
   {
     ascorb = FALSE;
   }
 else
   {
     fprintf(stderr,"%s: Unable to determine if orbit is ascending or ");
     fprintf(stderr,"descending\n", argv[0]);
     exit(1);
   }
/*
 Set up for direction.  If dir != 1, then we
 find the latitude that corresponds to minlon
 and maxlon.
*/
 if (dir != 1)
   {
     lon_to_lat();
   }

 /*
   Fill latitude array and find starting index...
   */
 for (i=0; i< MAXPOINTS; i++)
   {
     if (ascorb)
       {
         lat[i] = DEG*asin(sin((i-MAXPOINTS/2)*timestep*M2PI/PERIOD)*sin(INC));
         if (minval > lat[i])  latstart = i+1;
       }
     else
       {
         lat[i] = DEG*asin(sin(((MAXPOINTS/2)-i)*timestep*M2PI/PERIOD)*sin(INC));
         if (maxval < lat[i])  latstart = i+1;
       }
```

```
  }

/*
  1. Break into data segments...
  */
seg_len[0] = 1;
seg_beg[0] = 0;
time2 = frsin[0].utc + frsin[0].utcm*MICRO;
for (i=0; i<points; i++)
  {

    time1 = time2;
    time2 = frsin[i+1].utc + frsin[i+1].utcm*MICRO;

    if ((time2 - time1) <= deltmax)
      {
        seg_len[segments]++;
      }
    else
      {
        segments++;
        seg_beg[segments] = i + 1;
        seg_len[segments] = 1;
      }
  }

if (seg_len[0] == 0)  exit(1);

/*
  2. Initialize data and fit a spline to each segment...
  */

for(i=latstart; i<MAXPOINTS; i++)
  {
    frout[i].lat = (int)(lat[i]/MICRO);
    frout[i].lon = BAD;
    frout[i].m_h = BAD;
  }

for (i=0; i<=segments; i++)
  {
    fit_time(i);              /* initialize spline */
    fit_data(i,0);            /* lon */
    fit_data(i,1);            /* h */
    fit_data(i,2);            /* orb */
    fit_data(i,3);            /* s_h */
    fit_data(i,4);            /* geoid */
    fit_data(i,5);            /* swh */
    fit_data(i,6);            /* s_swh */
    fit_data(i,7);            /* s_nght */
    fit_data(i,8);            /* agc */
    fit_data(i,9);            /* s_agc */
    fit_data(i,10);           /* s_tide */
    fit_data(i,11);           /* o_tide */
```

```
      fit_data(i,12);            /* w_fnoc */
      fit_data(i,13);            /* w_smmr */
      fit_data(i,14);            /* d_fnoc */
      fit_data(i,15);            /* iono */
      fit_data(i,16);            /* dh_swh */
      fit_data(i,17);            /* dh_fm */
      fit_data(i,18);            /* att */
    }

/*
  Write out points...
*/

  for (k=latstart; k<j; k++)
    {
      if (fwrite((char *)&frout[k], 1, REC_LEN, stdout) != REC_LEN)
        {
          geo_error(3,argv[0]);
          exit(3);
        }

    }

}

/*
------------------------------------------------------------
  Subroutine fit_time
  Written by:
    Michael Caruso
    Woods Hole Oceanographic Institution
    Woods Hole, MA

  Purpose:
    This subprogram fits a spline to the time variable in a
    GEOSAT GDR.
------------------------------------------------------------
*/

int fit_time(i)
     int i;
{

  float yout;
  ii = 0;
  for (j=seg_beg[i]; j<seg_beg[i]+seg_len[i]; j++)
    {

      x[ii] = frsin[j].lat*MICRO;
      y[ii] = (float)(frsin[j].utc - frsin[seg_beg[i]].utc)
        + (float)frsin[j].utcm*MICRO;
      ii++;
    }
  x0 = x[0];
```

```
/*
  spline(x-1, y-1, ii, 1.e30, 1.e30, y2-1);
*/

  natcubspline(x, y, ii, x0, &yout, 0);

  j = latstart;
  while ((ascorb && (lat[j] < x0)) || (!ascorb && (lat[j] > x0)))
    {
      j++;
    }
  while ((lat[j] > minval) && (lat[j] < maxval))
    {
      if ((ascorb && (lat[j] > frsin[seg_beg[i]].lat*MICRO) &&
           (lat[j] < frsin[seg_beg[i]+seg_len[i]-1].lat*MICRO)) ||
          (!ascorb && (lat[j] < frsin[seg_beg[i]].lat*MICRO) &&
           (lat[j] > frsin[seg_beg[i]+seg_len[i]-1].lat*MICRO)))
        {
/*
          splint(x-1, y-1, y2-1, ii, lat[j], &yout);
*/
          natcubspline(x, y, ii, lat[j], &yout, 1);

          frout[j].utc = (int)yout + frsin[seg_beg[i]].utc;
          frout[j].utcm = (int)((yout - (int)yout)/MICRO);
        }
      else
        {
          frout[j].lat = (int)(lat[j]/MICRO);
          frout[j].utc = 0;
          frout[j].utcm = 0;
        }
      j++;
    }
  natcubspline(x, y, ii, lat[j], &yout, 2);

  return;
}
/*
-----------------------------------------------------
  Subroutine fit_data
  Written by:
    Michael Caruso
    Woods Hole Oceanographic Institution
    Woods Hole, MA

  Purpose:
    This subprogram fits a spline to the data segment
    specified.
-----------------------------------------------------
*/

int fit_data(i, var)
```

```
    int i, var;
{
  float yout;
  int iyout;
  ii = 0;
  for (j=seg_beg[i]; j<seg_beg[i]+seg_len[i]; j++)
    {
      x[ii] = frsin[j].lat*MICRO;
      switch (var)
        {
        case 0:
          y[ii] = (float)frsin[j].lon;
          break;
        case 1:
          y[ii] = (float)frsin[j].m_h;
          break;
        case 2:
          y[ii] = (float)frsin[j].orb;
          break;
        case 3:
          y[ii] = (float)frsin[j].s_h;
          break;
        case 4:
          y[ii] = (float)frsin[j].geoid;
          break;
        case 5:
          y[ii] = (float)frsin[j].swh;
          break;
        case 6:
          y[ii] = (float)frsin[j].s_swh;
          break;
        case 7:
          y[ii] = (float)frsin[j].s_nght;
          break;
        case 8:
          y[ii] = (float)frsin[j].agc;
          break;
        case 9:
          y[ii] = (float)frsin[j].s_agc;
          break;
        case 10:
          y[ii] = (float)frsin[j].s_tide;
          break;
        case 11:
          y[ii] = (float)frsin[j].o_tide;
          break;
        case 12:
          y[ii] = (float)frsin[j].w_fnoc;
          break;
        case 13:
          y[ii] = (float)frsin[j].w_smmr;
          break;
```

```
        case 14:
          y[ii] = (float)frsin[j].d_fnoc;
          break;
        case 15:
          y[ii] = (float)frsin[j].iono;
          break;
        case 16:
          y[ii] = (float)frsin[j].dh_swh;
          break;
        case 17:
          y[ii] = (float)frsin[j].dh_fm;
          break;
        case 18:
          y[ii] = (float)frsin[j].att;
          break;
        default:
          return(1);
        }

      ii++;
    }
  x0 = x[0];
/*
  spline(x-1, y-1, ii, 1.e30, 1.e30, y2-1);
*/
  natcubspline(x, y, ii, x0, &yout, 0);

  j = latstart;
  while ((ascorb && (lat[j] < x0)) || (!ascorb && (lat[j] > x0)))
    {
      j++;
    }
  while ((lat[j] > minval) && (lat[j] < maxval))
    {
      if ((ascorb && (lat[j] > frsin[seg_beg[i]].lat*MICRO) &&
           (lat[j] < frsin[seg_beg[i]+seg_len[i]-1].lat*MICRO)) ||
          (!ascorb && (lat[j] < frsin[seg_beg[i]].lat*MICRO) &&
           (lat[j] > frsin[seg_beg[i]+seg_len[i]-1].lat*MICRO)))
        {
/*
          splint(x-1, y-1, y2-1, ii, lat[j], &yout);
*/
          natcubspline(x, y, ii, lat[j], &yout, 1);

          iyout = nint(yout);
          set_data(j, iyout, var);
        }
      else
        {
          set_data(j, BAD, var);
        }
      j++;
    }
  natcubspline(x, y, ii, lat[j], &yout, 2);
```

```
  return(0);
}

/*
------------------------------------------------
  Subroutine set_data

  Written by:
    Michael Caruso
    Woods Hole Oceanographic Institution
    Woods Hole, MA

  Purpose:
    This subroutine simply puts the data into the
    correct element.
------------------------------------------------
*/
int set_data(point, data, var)
     int point;
     int data;
     int var;
{
  switch (var)
    {
    case 0:
      frout[point].lon = data;
      break;
    case 1:
      frout[point].m_h = data;
      break;
    case 2:
      frout[point].orb = data;
      break;
    case 3:
      frout[point].s_h = data;
      break;
    case 4:
      frout[point].geoid = data;
      break;
    case 5:
      frout[point].swh = data;
      break;
    case 6:
      frout[point].s_swh = data;
      break;
    case 7:
      frout[point].s_nght = data;
      break;
    case 8:
      frout[point].agc = data;
      break;
    case 9:
      frout[point].s_agc = data;
```

```
      break;
    case 10:
      frout[point].s_tide = data;
      break;
    case 11:
      frout[point].o_tide = data;
      break;
    case 12:
      frout[point].w_fnoc = data;
      break;
    case 13:
      frout[point].w_smmr = data;
      break;
    case 14:
      frout[point].d_fnoc = data;
      break;
    case 15:
      frout[point].iono = data;
      break;
    case 16:
      frout[point].dh_swh = data;
      break;
    case 17:
      frout[point].dh_fm = data;
      break;
    case 18:
      frout[point].att = data;
      break;
    default:
      return(1);
    }
}

/*
--------------------------------------------
  Subroutine lon_to_lat

  Written by:
    Michael Caruso
    Woods Hole Oceanographic Insititution
    Woods Hole, MA

  Purpose:
    This subroutine converts the minval and
    maxval when given in longitude to latitude.
-----------------------------------------------
*/
int
  lon_to_lat()

{
  int i;
  float tmpval;
  double tmptime;
```

```
/* Determine orbit number */
tmptime = frsin[0].utc + frsin[0].utcm*MICRO;
geo_cyc_orb(tmptime, &cyc, &orb);

rs = RE + frsin[0].orb;
rs3 = rs*rs*rs;
cosinc = cos(INCL);
prec = -1.5*J2*sqrt(GM/rs)*RE*RE*cosinc/rs3;
rot = prec - (M_PI_2/SD);
dthdt = M2PI/PERIOD;

/*
  Loop until we find min_lat and
  max_lat.
  */

lon0 = frsin[0].lon*1.0e-6*RAD;
time = 0.0;


if (ascorb)
  {

    i = 0;
    while (lon0 > maxval*RAD)
      {
        i++;
        lon0 = frsin[i].lon*1.0e-6*RAD;
      }

    while (lon0 < maxval*RAD)
      {
        time -= timestep;
        theta = dthdt*time;
        sinth = sin(theta);
        lat0 = (frsin[i].lat*1.0e-6)*RAD + asin(sin(INCL)*sin(theta));
        tmp = cosinc*sinth/cos(lat0);
        tmp = (tmp>1.0) ? 1.0 : tmp;
        tmp = (tmp<-1.0) ? -1.0 : tmp;
        lon0 = (asin(tmp) + rot*time) + frsin[i].lon*1.0e-6*RAD;
      }

    lon0 = frsin[points].lon*1.0e-6*RAD;
    tmpval = lat0*DEG;
    time = 0.0;
    i = points;
    while (lon0 < minval*RAD)
      {
        i--;
        lon0 = frsin[i].lon*1.0e-6*RAD;
      }
    while (lon0 > minval*RAD)
      {
```

```
      time += timestep;
      theta = dthdt*time;
      sinth = sin(theta);
      lat0 = (frsin[i].lat*1.0e-6)*RAD + asin(sin(INCL)*sin(theta));
      tmp = cosinc*sinth/cos(lat0);
      tmp = (tmp>1.0) ? 1.0 : tmp;
      tmp = (tmp<-1.0) ? -1.0 : tmp;
      lon0 = (asin(tmp) + rot*time) + frsin[i].lon*1.0e-6*RAD;
    }
  minval = tmpval;
  maxval = lat0*DEG;
 }
else
 {

  lon0 = frsin[0].lon*1.0e-6*RAD;
  time = 0.0;
  i = 0;
  while (lon0 > maxval*RAD)
    {
      i++;
      lon0 = frsin[i].lon*1.0e-6*RAD;
    }

  while (lon0 < maxval*RAD)
    {
      time -= timestep;
      theta = dthdt*time;
      sinth = sin(theta);
      lat0 = (frsin[i].lat*1.0e-6)*RAD + asin(sin(INCL)*sin(theta));
      tmp = cosinc*sinth/cos(lat0);

      tmp = (tmp>1.0) ? 1.0 : tmp;
      tmp = (tmp<-1.0) ? -1.0 : tmp;
      lon0 = (asin(tmp) + rot*time) + frsin[i].lon*1.0e-6*RAD;
    }

  maxval = lat0*DEG;
  lon0 = frsin[points].lon*1.0e-6*RAD;
  time = 0.0;
  i = points;
  while (lon0 < minval*RAD)
    {
      i--;
      lon0 = frsin[i].lon*1.0e-6*RAD;
    }
  while (lon0 > minval*RAD)
    {
      time += timestep;
      theta = dthdt*time;
      sinth = sin(theta);
      lat0 = (frsin[i].lat*1.0e-6)*RAD + asin(sin(INCL)*sin(theta));

      tmp = cosinc*sinth/cos(lat0);
```

```
        tmp = (tmp>1.0) ? 1.0 : tmp;
        tmp = (tmp<-1.0) ? -1.0 : tmp;
        lon0 = (asin(tmp) + rot*time) + frsin[i].lon*1.0e-6*RAD;
      }
    minval = lat0*DEG;
  }

}
```

Program g_uncompress.c

```
/*
 @(#)g_uncompress.c    1.2 6/14/90

 Written by:
 -----------
 Pierre Flament
 Oceanography Department
 University of Hawaii
 Honolulu, HI

 Modifications:
 --------------
 Mike Caruso
 Woods Hole Oceanographic Institution
 Woods Hole, MA

 Purpose:
 --------

 Uncompress compressed geosat data from 18 bytes/frame to
 standard NOAA data frame; leave 0 for items lost
 in compression.


 item  parameter              units    range          type

 1     TIME since start a000  ms       0 to 1.4/e9    long int (4)
 2     HEIGHT                 cm       0 to 32766     short int (2)
 3     CYCLE number                    0...           char (1)
 4     LATITUDE+90 deg.       10-4deg  0 to 18e5      unsigned int (3)
 5     LONGITUDE              10-4deg  0 to 36e5      unsigned int (3)
 6     SIGMA HEIGHT           cm       0 to 255       unsigned char (1)
 7     SWH                    5cm      0 to 255       unsigned char (1)
 8     So                     .1db     0 to 255       unsigned char (1)
 9     FLAGS                                          char (1)
 10    OCEAN TIDE             cm       -128 to 128    char (1)

 Method:
 -------

 Usage:
 ------
 g_uncompress < file.18b > file.gdr

 Input:
 ------
 18-byte data record

 Output:
```

```
  -------
  Pseudo Geosat GDR


*/
# include <stdio.h>

# define PERIOD 6037.551518571
# define START_TIME 58406188.43 /* equator xing orbit c000.a000 */
# define BAD 32767

/* this is the standard geosat frame */

struct  in_frame {
        long int utc,utcm,lat,lon,orb;
        short int m_h,s_h,geoid,h[10],swh,s_swh,s_nght,agc,s_agc;
        char fl[2];
        short int h_off,s_tide,o_tide,w_fnoc,w_smmr,d_fnoc,iono,dh_swh,dh_fm,att;
        };

struct in_frame in;

/* this is the compressed frame. Order is important since compiler
forces short int on even word boundaries */

struct  out_frame {
        long int utc;
        short int m_h;
        char cycle_n;
        char lat[3],lon[3];
        unsigned char s_h,swh,s_nght;
        char fl;
        char o_tide;
        };

struct out_frame out;

char junk[4];

double time,cycle=244*PERIOD;


int i,j;

main()

{
while(fread((char*)&out,1,18,stdin)==18)
        {
        time=out.utc/1000.+START_TIME+out.cycle_n*cycle;
        /* implicit cast to long int */
        in.utc=(long int)time;
        in.utcm=(long int)((time-(long int)time)*1e6);
```

```
        junk[1]=out.lat[0];
        junk[2]=out.lat[1];
        junk[3]=out.lat[2];

        in.lat=(*(long int*)&junk[0]-900000)*100;

        junk[1]=out.lon[0];
        junk[2]=out.lon[1];
        junk[3]=out.lon[2];

        in.lon=(*(long int*)&junk[0])*100;

        in.m_h=out.m_h;

        in.s_h=(out.s_h==255?BAD:out.s_h);
        in.swh=(out.swh==255?BAD:out.swh*5);
        in.s_nght=(out.s_nght==255?BAD:out.s_nght*10);
        in.fl[1]=out.fl;
        in.s_h=(out.s_h==255?BAD:out.s_h);
        in.o_tide=(out.o_tide==127?BAD:out.o_tide*10);
        fwrite((char*)&in,1,78,stdout);
        }
}
```

Program g_which.c

```
/*
 @(#)g_which.c 1.4 11/14/89

 Program  g_which.c

  Written by:
  -----------
  Michael Caruso
  Woods Hole Oceanographic Institution
  Woods Hole, MA

  Modifications:
  --------------
  Original concept by P. Flament
  Woods Hole Oceanographic Institution
  Woods Hole, MA

  Purpose:
  --------
  Reads minimum latitude and longitude, maximum latitude and
  longitude and returns the orbit numbers within that box.

  Usage:
  ------
  The program reads the minimum and maximum latitudes and longitudes
  from the command line.  If only two arguments are given, they are
  taken to be a lat/lon point and the nearest ascending and descending
  tracks are found.

  g_which 30 45 280 300

  or

  g_which 30 280

  Input:
  ------
  None

  Output:
  -------
  Orbit numbers suitable for use in a chain of pipes:

  cat c0/c000.'g_which 30 45 380 300' | g_ext 1 L

 Subroutines required:
 ---------------------
      geo_which       returns an array of 1's and 0's for each cycle
                      within the desired box.
```

```
  References:
  -----------

  Bugs:
  -----
        Assumes input data contains complete orbits.


*/

#include <stdio.h>
#include <sys/file.h>
#include <math.h>
#include "geos.h"

#define NUMARG          4
#define NUMARG2         2
#define MNLTARG         1
#define MXLTARG         2
#define MNLNARG         3
#define MXLNARG         4
#define MNLNARG2        2

char *cm="";    /* single , for print statement */

main(argc, argv)
     int argc;
     char *argv[];
{


  unsigned char a[ORB_PER_CYC],
              d[ORB_PER_CYC]; /* arrays of orbits within box          */

  int i;        /* Counter */

  double        min_lat,
                max_lat,
                min_lon,
                max_lon;        /* input lon-lat box                    */


  /*
    Read command line arguments.
    */

  if (argc == NUMARG + 1)
    {
      sscanf(argv[MNLTARG],"%lf",&min_lat);
      sscanf(argv[MXLTARG],"%lf",&max_lat);
      sscanf(argv[MNLNARG],"%lf",&min_lon);
      sscanf(argv[MXLNARG],"%lf",&max_lon);
```

```
    }
  else if (argc == NUMARG2 +1)
    {
      sscanf(argv[MNLTARG],"%lf",&min_lat);
      sscanf(argv[MNLNARG2],"%lf",&min_lon);
      max_lat = min_lat;
      max_lon = min_lon;
    }
  else
    {
      fprintf(stderr,"Usage: %s min_lat max_lat min_lon max_lon\n", argv[0]);
      fprintf(stderr," Or\n");
      fprintf(stderr,"       %s lat lon\n", argv[0]);
      exit(1);
    }

  /* determine orbits to remove. */

  geo_which(min_lat, max_lat, min_lon, max_lon, a, d);

  fprintf(stdout,"{");
  for (i=0; i<ORB_PER_CYC; i++)
    {
      if(a[i]) {fprintf(stdout,"%sa%03d",cm,i);cm=",";}
      if(d[i]) {fprintf(stdout,"%sd%03d",cm,i);cm=",";}
    }
  fprintf(stdout,"}");

}
```

Program s_ext r

```
/*
  @(#)s_ext.c   1.1 12/14/89

Program s_ext.c

Written by:
-------------
Mimi Baker
Pierre Flament
Oceanography Department
University of Hawaii
Honolulu, HI

5 October 1989

Modifications:
--------------
14 October 1989 MJC changed include ssmi.h to
geos.h for future compatibility.

Purpose:
--------------
To extract user specified data from an SSMI record

Method:
--------------
Reads raw SSMI data from standard input and applies
corrections and conversions.  Reads user desired output variables
from command line arguments.  Writes output on standard output in
ASCII format.

Usage:
--------------
The SSMI data are read from standard input and output variables are
read from  the command line.

cat s000.a002 | s_ext t l L > file.asc

will extract the time, latitude and longitude for each point
in the file s000.a002.

Input:
--------------
Stdin           Raw SSMI data, except for the times which are in
                GEOSAT time.

Output
--------------
Stdout          Extracted ASCII format
```

```
Subroutines required:
---------------------
None

References:
------------
None

*/

# include <math.h>
# include <stdio.h>
# include <string.h>
# include "geos.h"

# define MXP    9                 /* max number of parameters */
# define NCH    4                 /* number of SSMI channels:
                                         0 and 3 are long int
                                         1 and 2 are short int */

# define PRINT(X) printf(form[col[i]],X)

char* val[NCH+5]={"t","l","L","fl","ws","vp","cl","rn","cws"};

/* in which
   t    (seconds)         time since START_TIME
   l    (degrees)         latitude
   L    (degrees)         east longitude
   fl   (-)               flag indicating data characteristics
                          =0, over ocean
                          =1, no orbit altitude information
                          =2, over land
                          =3, over sea ice
   ws   (meters/second) wind speed (NOTE: if = BAD, no wind due to rain)
   vp   (kg/m*m)          columnar water vapor (NOTE: if = BAD, no water
                          vapor due to rain)
   cl   (kg/m*m)          columnar cloud water
   rn   (mm/hr)           rain rate
   cws  (meters)          SSMI correction for water vapor


*/


/* formats for printing output fields           */

char* form[NCH+5]={"%10.1lf\t","%6.2lf\t","%6.2lf\t","%d\t",
                   "%6.2lf\t","%6.2lf\t","%8.4lf\t","%7.3lf\t",
                   "%8.4lf\t"};


int i,j,col[MXP];
```

```
double ws(),vp(),cl(),rn(),cws();
double lat,lon;


main (argc,argv)
int argc;
char *argv[];

{
  for(j=0;j<MXP;j++)
    col[i] = -1;

  argc--;
  argv++;

  if (argc==0)
    {
      fprintf(stderr,"sext: argument error\n");
      exit(1);
    }


  /* find which channels should be processed
     i: argument/column index
     j: channel number
     col[i]: channel number corresponding to column i
     */

  for (i=0;i<argc;i++)
    for (j=0;j<NCH+6;j++)
      if(!strcmp(argv[i],val[j])) {col[i]=j;break;}

  while (fread((char*)&frssmi,1,12,stdin)==12)
    {
      lat = frssmi.lat*1.0e-02;
      lon = frssmi.lon*1.0e-02 + 180.;

      if (frssmi.flag!=0) continue;

      for (i=0;i<argc;i++)
        if (col[i]==0)
          PRINT(frssmi.utc-START_TIME);
        else if (col[i]==1)
          PRINT(lat);
        else if (col[i]==2)
          PRINT(lon);
        else if (col[i]==3)
          PRINT((int)frssmi.flag);
        else if (col[i]==4)
          PRINT(ws());
        else if (col[i]==5)
          PRINT(vp());
        else if (col[i]==6)
```

```
            PRINT(cl());
          else if (col[i]==7)
            PRINT(rn());
          else if (col[i]==8)
            PRINT(cws());
        printf("\n");
    }
}

double ws()
/* computes wind speed */

{
        double wind;

        wind = 0.2*(frssmi.win - 30.);
        if (wind == 45.0) wind = BAD;
        return(wind);
}

double vp()
/* computes water vapor */

{
        double vapor;

        vapor = 0.04*(frssmi.vap - 5.0);
        if (fabs(vapor - 10.) < 1.e-3 )
            vapor = BAD;
        else
            vapor = vapor*10.;
        return(vapor);
}

double cl()
/* computes cloud water */

{
        double cloud;

        cloud = 0.5*(frssmi.cld -32)*1.0e-02;
        return(cloud);
}

double rn()
/* computes rain rate */

{
        double rain;

        rain = 0.193*cl()*1.0e02 -0.48;
        if (rain < 0.0)
            rain = 0.0;
        return(rain);
```

```
}

double cws()
/* computes GEOSAT correction for water vapor

References:
------------

P.A. Phoebus and J.D. Hawkins, 'The impact of water vapor
attenuation on the interpretation of altimeter-derived ocean
topography in the Northeast Pacific', submitted to JGR, special
GEOSAT issue, June 1989

B.D. Tapley and J.B. Lundberg, 'The SEASAT altimeter wet
tropospheric range correction', JGR 87 pp. 3213-3220, 1982

*/

{
        double wettrop;

        if (vp() == BAD)
            wettrop = BAD;
        else
            wettrop = -0.00636*vp();
        return(wettrop);
}
```

Program s_region.c

```
/*

  @(#)s_region.c        1.2 12/14/89

  Program  s_region.c

   Written by:
   -----------
   Michael Caruso
   Woods Hole Oceanographic Institution
   Woods Hole, MA

   Modifications:
   --------------
   Mimi Baker
   Oceanography Department
   University of Hawaii
   Honolulu, HI

   modified g_region to s_region to process SSMI data
   4 October 1989

   MC 14 Oct 1989, changed include file ssmi.h to geos.h
   for future compatibility.

   MB 28 Nov 1989, changed calls to geo_cyc_orb to reflect
   changes in geo_cyc_orb.

   Purpose:
   --------
   Decodes SSMI data and separates raw data into separate orbits.  Each
   orbit is defined as beginning at the northernmost point of a track.
   Each orbit is further separated into an ascending section and a descending
   section.  Orbits are then written out to separate files of the form:

                smmm.annn or smmm.nnnd
        where
                mmm             is the cycle number,
                nnn             is the orbit number for that cycle,
                a               signifies ascending portion,
                d               signifies descending portion.

        The data is written out in the same form as it was read in.  This is
        consecutive records of 12 bytes each.

  Usage:
  ------
  The program reads the minimum and maximum latitudes and longitudes
  from the command line and reads the data from standard input.  To use
```

```
  the program to extract data from the tape (/dev/rmt8, 6250bpi, input
  block size 14400) issued by Wentz, from 10N to 30N and 280E to 300E:

  dd if=/dev/rmt8 ibs=14400 | s_region 1 10 30 280 300

  The first number on the argument line specifies whether the box
  should be bounded by a latitude line(1) or a longitude line(2).
  Note that longitudes are all east of Greenwich and if the box selected
  spans 360E, add 360 degrees to right edge of box, ie 350 365.

  Input:
  ------
  Stdin          Raw SSMI data

  Output:
  -------
  s???.????      SSMI data within region separated in ascending and
                 descending orbits, with times consistent with GEOSAT
                 times.

  Subroutines required:
  ---------------------
        geo_which       returns an array of 1's and 0's for each cycle
                        within the desired box.
        geo_error       prints error messages to standard error.
        geo_cyc_orb     return orbit and cycle number

  References:
  -----------

  Bugs:
  -----
        Assumes input data contains complete orbits.


*/

#include <stdio.h>
#include <sys/file.h>
#include <math.h>
#include "geos.h"

#define NUMARG  5
#define DIRARG  1
#define MNLTARG 2
#define MXLTARG 3
#define MNLNARG 4
#define MXLNARG 5
#define SEC_YR2 2.*365.*24.*3600.  /* 2 times seconds per year */


main(argc, argv)
     int argc;
     char *argv[];
```

```
{
  struct frames frssmi2;

  unsigned char a[ORB_PER_CYC],
                d[ORB_PER_CYC]; /* arrays of orbits within box         */

  char          str[80];        /* string for output file name         */

  short int     dir;            /* Direction of lat/lon boundary       */
  short int     orbit_num_tot;  /* the total number of orbits          */
  int           cycle_num;      /* the number of cycles since          */
  int           orbit_num;      /* orbit number within cycle 0-244     */
  short int     isopen = FALSE; /* check to see if file is already open */
  short int     asc;            /* flag for ascending or descending    */

  long int      llcmp,
                llmin, llmax;   /* lat/lon boundary                    */
  long int      lslope1,
                lslope2;        /* "Slope" of orbit                    */


  double        min_lat,
                max_lat,
                min_lon,
                max_lon;        /* input lon-lat box                   */
  double        time;           /* time variable                       */


  FILE *fdout;                  /* output file descriptor              */


  /*
    Read command line arguments.
    */

  if (argc == NUMARG + 1)
    {
      sscanf(argv[DIRARG], "%hd", &dir);
      sscanf(argv[MNLTARG],"%lf",&min_lat);
      sscanf(argv[MXLTARG],"%lf",&max_lat);
      sscanf(argv[MNLNARG],"%lf",&min_lon);
      sscanf(argv[MXLNARG],"%lf",&max_lo`;
    }
  else
    {
      fprintf(stderr,"Usage: %s dir min_lat max_lat min_lon max_lon\n", argv[0]);
      exit(1);
    }

  /* determine orbits to remove. */

  geo_which(min_lat, max_lat, min_lon, max_lon, a, d);
```

```
/*    Set llmin, llmax...     */

if (dir == 1)
  {
    llmin = (int) (min_lat*1.0e02);
    llmax = (int) (max_lat*1.0e02);
  }
else
  {
    llmin = (int) ((min_lon - 18000)*1.0e02);
    llmax = (int) ((max_lon - 18000)*1.0e02);
  }


/* read initial lat and long coordinates */

if(fread((char *)&frssmi,1,SSMIREC,stdin) != SSMIREC)
  {
    geo_error(2, argv[0]);
    exit(2);
  }
frssmi.utc = frssmi.utc + SEC_YR2;

if(fread((char *)&frssmi2,1,SSMIREC,stdin) != SSMIREC)
  {
    geo_error(2, argv[0]);
    exit(2);
  }
frssmi2.utc = frssmi2.utc + SEC_YR2;

/* Determine name of first orbit */

time = frssmi.utc;
lslope1 = frssmi2.lat - frssmi.lat;
/*
  orbit_num_tot = (int)floor((time-TIME_ZERO)/PERIOD);
  cycle_num     = orbit_num_tot / ORB_PER_CYC;
  orbit_num     = orbit_num_tot % ORB_PER_CYC;
*/

geo_cyc_orb(time, &cycle_num, &orbit_num);

/*
  Check to see if first orbit is ascending or
  descending...
  */

if ( lslope1 > 0 )
  {
    sprintf(str,"s%.3d.a%.3d",cycle_num,orbit_num);
    asc = TRUE;
  }
else if ( lslope1 < 0)
```

```
  {
    sprintf(str,"s%.3d.d%.3d",cycle_num,orbit_num);
    asc = FALSE;
  }
else
  {
    if (frssmi.lat < 0)
      {
        sprintf(str,"s%.3d.a%.3d",cycle_num,orbit_num);
        asc=TRUE;
      }
    else
      {
        sprintf(str,"s%.3d.d%.3d",cycle_num,orbit_num);
        asc=FALSE;
      }
  }

/* Check to see if point is an orbit we want and greater
   than the minimum latitude and smaller than the
   maximum latitude. If so, write to the output file. If
   the output file is not open, open it and mark it as
   being open.        */

llcmp = (dir == 1) ? frssmi.lat : frssmi.lon;

if(((asc && a[orbit_num]) || (!asc && d[orbit_num])) && (llcmp > llmin)
   && (llcmp < llmax))
  {
    if (isopen == 0)
      {
        fdout = fopen(str,"a");
        isopen = 1;
      }
    if(fwrite((char *)&frssmi,1,SSMIREC,fdout) != SSMIREC)
      {
        geo_error(3, argv[0]);
        exit(3);
      }
  }

/*
 Check second point...
 */

llcmp = (dir == 1) ? frssmi2.lat : frssmi2.lon;

if(((asc && a[orbit_num]) || (!asc && d[orbit_num])) && (llcmp > llmin)
   && (llcmp < llmax))
  {
    if (isopen == 0)
      {
        fdout = fopen(str,"a");
        isopen = 1;
```

```
        }
      if(fwrite((char *)&frssmi2,1,SSMIREC,fdout) != SSMIREC)
        {
          geo_error(3, argv[0]);
          exit(3);
        }
    }



  /* Read in rest of geosat data.  We keep three points active
     to monitor when an orbit changes from ascending to descending.
     This was done because of the incomplete data at high latitudes.
     */

  frssmi = frssmi2;

  while(fread((char *)&frssmi2,1,SSMIREC,stdin) == SSMIREC)
    {

      frssmi2.utc = frssmi2.utc + SEC_YR2;
      lslope2 = frssmi2.lat - frssmi.lat;

      if ((lslope1 > 0 && lslope2 <= 0) || (lslope1 < 0 && lslope2 >= 0))
        {
          /* Determine name of next orbit */

          time = frssmi2.utc;
/*
          orbit_num_tot = (int)floor((time-TIME_ZERO)/PERIOD);
          cycle_num     = orbit_num_tot / ORB_PER_CYC;
          orbit_num     = orbit_num_tot % ORB_PER_CYC;
 */

          geo_cyc_orb(time, &cycle_num, &orbit_num);

          if ( lslope2 > 0 )
            {
              sprintf(str,"s%.3d.a%.3d",cycle_num,orbit_num);
              asc=TRUE;
            }
          else if ( lslope2 < 0 )
            {
              sprintf(str,"s%.3d.d%.3d",cycle_num,orbit_num);
              asc=FALSE;
            }
          else
            {
              if (frssmi2.lat < 0 )
                {
                  sprintf(str,"s%.3d.a%.3d",cycle_num,orbit_num);
                  asc=TRUE;
                }
```

```
              else
                {
                  sprintf(str,"s%.3d.d%.3d",cycle_num,orbit_num);
                  asc=FALSE;
                }
            }

          fclose(fdout);          /* Close previous file */
          isopen = 0;
        }
      llcmp = (dir == 1) ? frssmi2.lat : frssmi2.lon;

      if(((asc && a[orbit_num]) || (!asc && d[orbit_num])) && (llcmp > llmin)
         && (llcmp < llmax))
        {
          if (isopen == 0)
            {
              fdout = fopen(str,"a");
              isopen = 1;
            }
          if(fwrite((char *)&frssmi2,1,SSMIREC,fdout) != SSMIREC)
            {
              geo_error(3, argv[0]);
              exit(3);
            }
        }

      frssmi = frssmi2;
      lslope1 = lslope2;

    }

}
```

Subroutine geo_cyc_orb.c

```
/*
  @(#)geo_cyc_orb.c     1.3 6/14/90

  Written by:
  -----------
  Michael Caruso
  Woods Hole Oceanographic Institution.
  Woods Hole, MA

  Modified by:
  ------------
  Mimi Baker
  Oceanography Department
  University of Hawaii
  Honolulu, HI

  28 November 1989, to make subroutine frame independent
  and a function of time only.

  Purpose:
  --------
  This subroutine calculates the orbit number of
  a given GEOSAT GDR or SSMI data.

  Usage:
  ------
    geo_cyc_orb(time, cyc, orb)
          double time           time of record
          int *cyc;             the cycle number
          int *orb;             the orbit number

  Returns:
  --------
    -1          on error.

  Reference:
    None.

*/

#include "geos.h"
#include <math.h>

int geo_cyc_orb(time, cyc, orb)
     double time;
     int *cyc, *orb;

{
  double orbit;
```

```
  int orbit_num_tot;

  if ((time - TIME_ZERO) < 0 )
    return(-1);

  orbit = (time-TIME_ZERO)/PERIOD;
  orbit_num_tot = (int)floor(orbit);

  if((orbit - (int)orbit) > 0.99)
    orbit_num_tot += 1;

  *cyc     = orbit_num_tot / ORB_PER_CYC;
  *orb     = orbit_num_tot % ORB_PER_CYC;

  return;

}
```

Subroutine geo_error.c

```
/*
  @(#)geo_error.c       1.2 6/14/90

  Written by:
  -----------
  Michael Caruso
  Woods Hole Oceanographic Institution
  Woods Hole, MA
  October 1988

  Purpose:
  --------
  This program prints an error message to standard error
  along with the name of the program that generated the
  error.

  Method:
  ------

  Usage:
  ------
    geo_err(num, progname)
       int      num             Error number to print.
       char     *progname       Program generating error.

  Input:
  ------
    num         Error number to print.
    *progname   Pointer to name of program generating error.

  Output:
  -------
    Error message to standard error.

  Returns:
  --------
    None.

  Subroutines Required:
  --------------------
    None.

*/

#include <stdio.h>
#define NUMMSG          4
```

```
/*
        define array of error messages...
*/

char *mesg[] = {"Unrecoverable error\n",
                  "c???.[ad]???\n",
                  "Error reading file\n",
                "Error writing file\n"};

geo_error(num, prog)
        long num;
        char *prog;
{

  if (num < NUMMSG)
    fprintf(stderr,"%s: %s", prog, mesg[num]);
  else
    fprintf(stderr,"%s: %s", prog, mesg[0]);

}
```

Subroutine geo_mask.c

```
/*
 @(#)geo_mask.c        1.3 6/14/90

 Written by:
 -----------
 Pierre Flament
 Oceanography Department
 University of Hawaii
 Honolulu, HI

 Modifications:
 --------------
 Mike Caruso
 Woods Hole Oceanographic Institution
 Woods Hole, MA

 Purpose:
 --------
 Reads in environment variable GMASK if available
 and converts to integers.  Any character other
 than a 0 or a 1 is ignored.

 Method:
 -------
 Checks to see if the user has set an environment
 variable GMASK.  If so, the program returns it.

 Input:
 ------
 None

 Output:
 -------
 msk
 valid

 Returns:
 --------
      msk      an integer with bits set to correspond
               to GMASK values of 1
      valid    an integer with bits set to correspond
               to GMASK values of 0 and 1

*/
#define NULL 0
geo_mask(msk,valid)

short int *msk,*valid;
```

```
{
  char * getenv();

  char* st="1-------0-------00";  /* default msk; - can be any char */
  int i;

  if (getenv("GMASK")!=NULL) st=getenv("GMASK");

  for (i=0 ; *(st+i)!='\0' && i<16 ; i++)
    {
      if (*(st+i)=='1') *valid += 1<<i;
      if (*(st+i)=='1' || *(st+i)=='0') *msk += 1<<i;
    }
}
```

Subroutine geo_which.c

```
/*
  @(#)geo_which.c       1.3 6/14/90

  Written by:
  -----------
  Pierre Flament
  Woods Hole Oceanographic Institution
  Woods Hole, MA

  Modified by:
  ------------
  Michael Caruso
  Woods Hole Oceanographic Institution
  Woods Hole, MA

  Purpose:
  --------
  This procedure determines which orbit numbers cross a given
  area.

  Method:
  -------
  Find the times when ascending and descending parts of orbit 000
  cross min_lat and max_lat.  Find the corresponding longitudes for
  that orbit.  Then repeatedly shift the orbit by INCR, the ground
  spacing between successive orbits, and flag those that cross the
  box.

  Only a first order sinusoidal approximation of the orbit ground path
  is used.

  Usage:
  ------
  geo_which(min_lat, max_lat, min_lon, max_lon, a, d)
        int             min_lat, max_lat
        int             min_lon, max_lon
        unsigned char   *a, *d

  Input:
  ------
    min_lat             minimum latitude of box.
    max_lat             maximum latitude of box.
    min_lon             minimum eastward longitude of box.
    max_lon             maximum eastward longitude of box.

  Output:
  -------
    *a                  array of 244 elements.  Array has
                        1 in location i if ascending orbit
```

```
                        i crosses box, 0 otherwise.
  *d                    Same as *a, except for descending
                        orbits.

 Returns:
 --------
  None.

 Subroutines Required:
 ---------------------
  double   fold(x)    folds angle x, in radians, to the range 0-2*M_PI.

 Reference:
  None.

*/

#include "geos.h"
#include <stdio.h>
#include <math.h>

# define SWAP(X1,X2) {x=X1;X1=X2;X2=x;}

# define NODE    (START_LON*RAD)        /* the longitude Xing of c000.a000 */
# define OM      (2*M_PI/PERIOD)        /* orbital omega */
# define DELT    (2*M_PI/244.)          /* the increment between adjacent orbits
*/
# define INCR    (2*M_PI*17./244.)      /* the increment between successive orbits
*/
# define REP     (OM*17./244.)          /* 1/17 of the repeat cycle */
# define SINCL   (sin(INCL))            /* the sin of inclination */
# define COSCL   (cos(INCL))            /* the cos of inclination */
# define EPSLAT  (5.e-2)                /* tolerance at +- INCL */

double
  min_time_a,   /* time orbit a000 crosses min_lat               */
  min_time_d,   /* time orbit d000 crosses min_lat               */
  max_time_a,   /* time orbit a000 crosses max_lat               */
  max_time_d,   /* time orbit d000 crosses max_lat               */
  min_lon_a,    /* longitude where asc orbit crosses min_lat     */
  min_lon_d,    /* longitude where dec orbit crosses min_lat     */
  max_lon_a,    /* longitude where asc orbit crosses max_lat     */
  max_lon_d,    /* longitude where dec orbit crosses max_lat     */
  del_lon,      /* longitude width of the box                    */
  x;            /* dummy variable                                */

int
  i, s;


geo_which(min_lat, max_lat, min_lon, max_lon, a, d)
     double min_lat, max_lat, min_lon, max_lon;
     unsigned char *a, *d;
```

```
{
  double fold();

  /* check order of min,max_lat, SWAP if necessary, convert to RAD */

  if (min_lat > max_lat) SWAP(min_lat,max_lat);
  min_lat *= RAD;
  max_lat *= RAD;

  /* check to see if min,max_lat exceed satellite inclination.*/

  min_lat = (min_lat>M_PI-INCL?M_PI-INCL-EPSLAT:min_lat);
  min_lat = (min_lat<INCL-M_PI?INCL-M_PI+EPSLAT:min_lat);

  max_lat = (max_lat>M_PI-INCL?M_PI-INCL-EPSLAT:max_lat);
  max_lat = (max_lat<INCL-M_PI?INCL-M_PI+EPSLAT:max_lat);

  /* convert lon to RAD and check to see if the calling program has
    selected an individual point */

  min_lon *= RAD;
  max_lon *= RAD;

  if (min_lon == max_lon )
    {
      min_lon = min_lon-DELT/2;
      max_lon = max_lon+DELT/2;
    }

  /* min,max_lon are always given in the right order.  Fold them
    so that they are always in the range 0 to 4*M_PI and
    max_lon>min_lon */

  while (min_lon > max_lon) max_lon += 2*M_PI;
  del_lon = max_lon - min_lon;
  min_lon = fold(min_lon);
  del_lon = fold(del_lon);
  max_lon = min_lon + del_lon;

  /* find times at which orbit a000 crosses min_lat and max_lat;
        the origin of time is at the ascending node */

  min_time_a = asin(sin(min_lat)/SINCL)/OM;
  max_time_a = asin(sin(max_lat)/SINCL)/OM;

  /* find times at which orbit d000 crosses min_lat and max_lat;
     d000 is BEFORE a000; the origin of time is at the ascending node */

  min_time_d = -PERIOD/2 - min_time_a;
  max_time_d = -PERIOD/2 - max_time_a;

  /* find corresponding longitudes; here min_lon_? correspond to min_time_?,
        and do not necessarily mean a minimum longitude */
```

```
min_lon_a = fold(NODE+atan2(COSCL*sin(OM*min_time_a),cos(OM*min_time_a))-
            min_time_a*REP);
min_lon_d = fold(NODE+atan2(COSCL*sin(OM*min_time_d),cos(OM*min_time_d))-
            min_time_d*REP);
max_lon_a = fold(NODE+atan2(COSCL*sin(OM*max_time_a),cos(OM*max_time_a))-
            max_time_a*REP);
max_lon_d = fold(NODE+atan2(COSCL*sin(OM*max_time_d),cos(OM*max_time_d))-
            max_time_d*REP);

/* check if orbit crosses the given box, then shift the orbit by INCR.
      A given orbit crosses the box if the top right and bottom left
      corners fall on opposite sides of an ascending orbit, or if the
      top left and bottom right corners fall on the opposite sides of
      a descending orbit. This can be expressed by the conditions

              (min_lon-min_lon_a)*(max_lon-max_lon_a)<=0

              (max_lon-min_lon_d)*(min_lon-max_lon_d)<=0

      given the definition of min_lon_a, etc...
              */

for(i=0;i<244;i++)
  {

/* special care must be taken when the orbit spans 360; in that
      case, max_lon_d and min_lon_a were folded too much and 2*M_PI
      must first be added to them */

  if(min_lon_a<max_lon_a) min_lon_a += 2*M_PI;
  if(min_lon_d>max_lon_d) max_lon_d += 2*M_PI;

  a[i]=d[i]=0;

/* test if the orbit crosses the box and the box shifted by +- 2*M_PI */

  for(s = -1 ; s <= 1 ; s++)
      {
          if( (min_lon+s*2*M_PI-min_lon_a)*(max_lon+s*2*M_PI-max_lon_a)<=0)
              a[i]=1;
          if( (max_lon+s*2*M_PI-min_lon_d)*(min_lon+s*2*M_PI-max_lon_d)<=0 )
              d[i]=1;
      }

    min_lon_a = fold(min_lon_a-INCR);
    min_lon_d = fold(min_lon_d-INCR);
    max_lon_a = fold(max_lon_a-INCR);
    max_lon_d = fold(max_lon_d-INCR);
  }

}
/*
```

```
  Function fold

  Written by:
    Pierre Flament
    Woods Hole Oceanographic Institution
    Woods Hole, MA

  Modifications:
    None.

  Purpose:
    To fold an angle to the range 0-2*M_PI by removing or adding an integer
    number of 2*M_PI.

  Method:
    None.

  Usage:
    r = fold(x)
    double r, x

  Input:
    x           angle to be folded, in radians

  Output:
    None.

  Returns:
    r           folded angle corresponding to x.

  Subroutines Required:
    remainder(x,y) double x,y; which returns a number in the range -y/2 to
    y/2 which differs from x by an integer number times y, as defined in the
    reference.

  Reference:
    ANSI/IEEE Std 754-1985

*/
double fold(x)
     double x;
{
  double r;
  r=remainder(x,2*M_PI);
  return(r<0?r+2*M_PI:r);
}
```

C Shell Listings

This appendix contains listings and descriptions of shell scripts used in this report. Experienced shell programmers may wish to modify the scripts for complicated analysis. For novice shell programmers, a description of each of the scripts is given along with necessary modifications.

C.1 Repeat Analysis

This shows how to use the basic programs to clean and correct raw GEOSAT GDRs and perform the repeat analysis described in section 6. This particular shell script uses the C-Shell instead of the Bourne Shell [4] since the C-Shell provides the command *foreach*. With a few minor modifications, these scripts could perform an analysis on an entire region instead of one orbit. The first script assumes that each file is in a directory named with the cycle number and the filename follows the convention given in section 4.

```
#csh
# shell for generating single orbit repeat
# track analysis
mkdir means
foreach i (c???)
echo $i
#
cat $i/$i."$1" | g_clean1 | g_correct | g_clean2 >! tmp
(cat tmp | g_spike | g_spline 1 22 48 3.3 0.97992165 >
$i/$i."$1"c)
end
#
echo Performing repeat analysis.
g_repeat c*/c*."$1"c > means/mean."$1"c
```

This script is called with the orbit number desired:

```
gs_repeat a002
```

First a directory is created to hold the means. Then a loop is created using all the cycle subdirectories. The desired orbit *a002* is then cleaned, corrected and splined for all available cycles. When all cycles are processed, the repeat analysis is performed and the mean/geoid is placed in the *means* subdirectory.

The second script is similar to the first except that it assumes that each file is in the current directory.

```
#csh
# shell for generating single orbit repeat
# track analysis
foreach i (c???.$1)
echo $i
#
```

[4]There are many good shell programming books available to describe the similarities and differences between the C-Shell and the Bourne Shell.

```
    cat $i | g_clean1 | g_correct | g_clean2 >! tmp
    (cat tmp | g_spike | g_spline 1 22 48 3.3 0.97992165 >
"$i"c)
    end
    #
    echo Performing repeat analysis.
    g_repeat "$i"c > mean."$1"
```

C.2 Data Extraction

Sometimes it is useful to view cleaned and corrected data. The following scripts show how this may be done.

```
    #
    # shell script for extracting clean sea surface heights
    #
    cat $1 | g_clean1 | g_correct | g_clean2 | g_spike | g_ext 1
L ha > "$1"h
    #
```

This script would remove bad data and extract the latitude, the longitude and the sea surface height.

```
    #
    # shell script for extracting clean significant wave heights
    #
    cat $1 | g_clean1 | g_correct | g_clean2 | g_spike | g_ext 1
L w sw > "$1"w
    #
```

This script would extract the significant wave height and the standard deviation of the significant wave height.

C.3 Imaging

This script shows how to create a variability image using the output from the repeat script shown above. This script uses the UNIX command *cut* to select the necessary data from the input files. In this example, the latitude, longitude and the variability columns are extracted and piped to *g_image*. The arguments to g_image are the minimum and maximum latitude, the minimum and maximum longitude and the number of rows and columns in the output image. This image is in SDPS floating point format and needs to be remapped to a bitmap image. This is done using the program *sdps_ftb* which converts floating point images to byte images. The arguments to *sdps_ftb* specify that the bitmap should be scaled from 0.0 to 0.5 and any values less than 0.0 should be set to 254 and values greater than 0.5 should be set to 255.

```
#
# shell for generating equirectangular image
# of variability.
#
cat mean.a* | cut -f2,3,7 | g_image 22 48 284 316 416 512
> vara.sdpsf
cat vara.sdpsf | sdps_ftb -mxlh 0.0 0.5 254 255 > vara.sdps
```

The following script is only a variation of the previous script. It merely demonstrates how to create an image of the mean sea surface height of all the descending tracks in a region. Since there were no arguments given to *sdps_ftb*, the minimum and maximum are found and used for scaling the output bitmap image.

```
#
# shell for generating equirectangular image
# of mean sea surface height.
#
cat mean.a* | cut -f2,3,6 | g_image 22 48 284 316 416 512
> meand.sdpsf
cat meand.sdpsf | sdps_ftb  > meand.sdps
```

DOCUMENT LIBRARY

January 17, 1990

Distribution List for Technical Report Exchange

Attn: Stella Sanchez-Wade
Documents Section
Scripps Institution of Oceanography
Library, Mail Code C-075C
La Jolla, CA 92093

Hancock Library of Biology &
Oceanography
Alan Hancock Laboratory
University of Southern California
University Park
Los Angeles, CA 90089-0371

Gifts & Exchanges
Library
Bedford Institute of Oceanography
P.O. Box 1006
Dartmouth, NS, B2Y 4A2, CANADA

Office of the International
Ice Patrol
c/o Coast Guard R & D Center
Avery Point
Groton, CT 06340

NOAA/EDIS Miami Library Center
4301 Rickenbacker Causeway
Miami, FL 33149

Library
Skidaway Institute of Oceanography
P.O. Box 13687
Savannah, GA 31416

Institute of Geophysics
University of Hawaii
Library Room 252
2525 Correa Road
Honolulu, HI 96822

Marine Resources Information Center
Building E38-320
MIT
Cambridge, MA 02139

Library
Lamont-Doherty Geological
Observatory
Colombia University
Palisades, NY 10964

Library
Serials Department
Oregon State University
Corvallis, OR 97331

Pell Marine Science Library
University of Rhode Island
Narragansett Bay Campus
Narragansett, RI 02882

Working Collection
Texas A&M University
Dept. of Oceanography
College Station, TX 77843

Library
Virginia Institute of Marine Science
Gloucester Point, VA 23062

Fisheries-Oceanography Library
151 Oceanography Teaching Bldg.
University of Washington
Seattle, WA 98195

Library
R.S.M.A.S.
University of Miami
4600 Rickenbacker Causeway
Miami, FL 33149

Maury Oceanographic Library
Naval Oceanographic Office
Bay St. Louis
NSTL, MS 39522-5001

Marine Sciences Collection
Mayaguez Campus Library
University of Puerto Rico
Mayagues, Puerto Rico 00708

Library
Institute of Oceanographic Sciences
Deacon Laboratory
Wormley, Godalming
Surrey GU8 5UB
UNITED KINGDOM

The Librarian
CSIRO Marine Laboratories
G.P.O. Box 1538
Hobart, Tasmania
AUSTRALIA 7001

Library
Proudman Oceanographic Laboratory
Bidston Observatory
Birkenhead
Merseyside L43 7 RA
UNITED KINGDOM

50272-101

REPORT DOCUMENTATION PAGE	1. REPORT NO. WHOI-90-45	2.	3. Recipient's Accession No.

4. Title and Subtitle	5. Report Date
Altimeter Processing Tools for Analyzing Mesoscale Ocean Features	September, 1990
	6.
7. Author(s) Michael J. Caruso, Ziv Sirkes, Pierre J. Flament, and M.K. Baker	8. Performing Organization Rept. No. WHOI 90-45
9. Performing Organization Name and Address The Woods Hole Oceanographic Institution Woods Hole, Massachusetts 02543	10. Project/Task/Work Unit No.
	11. Contract(C) or Grant(G) No. (C) N00014-86-K-0751 (G)
12. Sponsoring Organization Name and Address Funding was provided by the Office of Naval Research.	13. Type of Report & Period Covered Technical Report
	14.

15. Supplementary Notes

This report should be cited as: Woods Hole Oceanog. Inst. Tech. Rept., WHOI-90-45.

16. Abstract (Limit: 200 words)

Satellite altimeters provide many opportunities for oceanographers to supplement their research with a valuable new data set. The recent GEOSAT exact repeat mission is the first of several altimetry missions proposed during the next decade. To utilize this new data, a software package was developed at the Woods Hole Oceanographic Institution and the University of Hawaii to facilitate the extraction of useful information from the NODC distributed GEOSAT data tapes. This software package was written with portability and modularity in mind. It should be possible to use this package with little or no modifications on data from future altimeters. The code was critten in C and tested on Sun workstations and is oriented toward UNIX operating systems. However, since standard code was used, the programs should port easily to other computer systems. The modularity of the code should enable users to create additional programs. Additional programs designed to handle collocated water vapor corrections are also included for comparison.

17. Document Analysis a. Descriptors

1. GEOSAT
2. altimeter
3. software

b. Identifiers/Open-Ended Terms

c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report)	21. No. of Pages
Approved for publication; distribution unlimited.	UNCLASSIFIED	209
	20. Security Class (This Page)	22. Price

(See ANSI-Z39.18) *See Instructions on Reverse* OPTIONAL FORM 272 (4-77) (Formerly NTIS-35) Department of Commerce