SOUTHAMPTON OCEANOGRAPHY CENTRE

REPORT

No. 2

Optical Plankton Counter SeaSoar data collected on *Polarstern* Cruise ANT-XIII/2 04 Dec 1995 - 24 Jan 1996

R T Pollard[1], M J Griffiths[2], T J P Gwilliam[3] & J F Read[1]

1996

[1]George Deacon Division for Ocean Processes
[2]James Rennell Division for Ocean Circulation
[3]Ocean Technology Division
Southampton Oceanography Centre
European Way
Southampton SO14 3ZH
UK

DOCUMENT DATA SHEET

AUTHOR

POLLARD, R T, GRIFFITHS, M J, GWILLIAM, T J P & READ, J F

PUBLICATION DATE

1996

TITLE

Optical Plankton Counter SeaSoar data collected on *Polarstern* Cruise ANT-XIII/2, 04 Dec 1995 - 24 Jan 1996.

REFERENCE

Southampton Oceanography Centre, Report No. 2, 43pp. & figs.

ABSTRACT

Four SeaSoar surveys were carried out from FS *Polarstern* during the ANT XIII/2 cruise, to look at upper ocean dynamics in areas of high biological productivity. As well as the standard CTD (conductivity, temperature, depth) measurements, the SeaSoar was fitted with a fluorimeter and Optical Plankton Counter, the latter being used for the first time. This report describes in some detail the methods used to reduce the OPC data and presents the data contour plots and profiles, section by section, without interpretation. The data are presented using the manufacturer's default conversion tables from particle size to equivalent spherical diameter. More accurate calibration awaits analysis of zooplankton samples.

KEYWORDS

CRUISE XIII/2 1995, CTD OBSERVATIONS, DATA PROCESSING, FLUORIMETER, OPC-1T, OPTICAL PLANKTON COUNTER, *POLARSTERN*, SEASOAR, ZOOPLANKTON

ISSUING ORGANISATION

Southampton Oceanography Centre
European Way
Southampton SO14 3ZH
UK

Director: Professor John Shepherd

Copies of this report are available from: **The National Oceanographic Library, SOC** *PRICE:* £24.00
Tel: 01703 596116
Fax: 01703 596115

This page intentionally left blank

Contents

Optical Plankton Counter SeaSoar data collected on *Polarstern* Cruise ANT-XIII/2, 4 Dec 1995 - 24 Jan 1996

R T Pollard, M J Griffiths, T J P Gwilliam and J F Read

Abstract

Four SeaSoar surveys were carried out from FS *Polarstern* during the ANT XIII/2 cruise, to look at upper ocean dynamics in areas of high biological productivity. As well as the standard CTD (conductivity, temperature, depth) measurements, the SeaSoar was fitted with a fluorimeter and Optical Plankton Counter (OPC), the latter being used for the first time. This report describes in some detail the methods used to reduce the OPC data and presents the data contour plots and profiles, section by section, without interpretation. The data are presented using the manufacturer's default conversion tables from particle size to equivalent spherical diameter. More accurate calibration awaits analysis of zooplankton samples.

Introduction

ANT XIII/2 was the first cruise on which the SeaSoar was fitted with an Optical Plankton Counter (OPC). A team from the Southampton Oceanography Centre (SOC) participated in this cruise by kind invitation of Dr V Smetacek of the Alfred Wegener Institut für Polar- und Meeresforschung (AWI). This report describes our first experiences, the programs and execs developed to process the data and presents the data collected but as yet uncalibrated against net samples.

The OPC-1T optical plankton counter was developed at the Bedford Institute of Oceanography (Herman, 1992) and is manufactured by Focal Technologies Inc. The instrument used on SeaSoar (serial number TOW026) was purchased in 1995. It was fitted with an acrylic flow insert to reduce the tunnel cross-section to 0.001 m^2 for towed use and was mounted beneath the SeaSoar in place of the usual torpedo shaped weight. No problems were experienced, but it appeared that the extra drag somewhat limited the maximum depth that could be reached with acceptable cable strains. Profiling on AntXIII/2 was usually down to 350 - 360 m but had to be limited to 300 m on occasions when the cable termination was beginning to fail. The cable connection out of the rear of the OPC was vulnerable to crushing during deployment and recovery when the SeaSoar was being lifted out of or placed into its stand, also when the SeaSoar and stand were being moved around the deck. However, no failures occurred.

The OPC estimates the equivalent spherical diameter (ESD) of particles that break the 640 nm LED light beam. ESDs between 250μ and 3mm can be resolved. However, for calibration and interpretation it is essential to collect net samples of zooplankton and possibly to use a laboratory version of the OPC to investigate how it responds to the shapes and translucence of the zooplankton sizes and species observed. On AntXIII/2, Corinna Dubischar of the AWI was using a second OPC mounted on a 5-sample multinet to obtain calibration samples during station work. These will be analysed and used in due course to calibrate the data introduced here. However, in this report all data are "calibrated" only by using the reference calibration table given in the OPC handbook.

The SeaSoar, OPC and associated computing were installed on the *Polarstern* in Bremerhaven prior to the cruise. Details of operations are given in the associated SeaSoar data report (Griffiths *et al.*, 1996), where a diary of deployments is also given. A summary only is presented here. Details of the meteorological conditions may be found in the Underway Data Report Read *et al.*, 1996) and Acoustic Doppler Current Profiler Data are presented in the ADCP Data Report (Allen *et al.*, 1996)

There were three main deployments of the SeaSoar/OPC system on the cruise (Tables 1 and 2, Figures 1-3), a long transect on passage from Cape Town to Antarctica, a Course Scale Survey (CSS) and a Fine Scale Survey (FSS). The long transect, spanning 15.5° of latitude from 41°47'S, 12°46'E to 57°19'S, 2°06'W on passage to the German Antarctic base at Neumayer, was split into Runs 2 and 3 (Table 1, Fig. 1)) by a break for an attempted mooring recovery near 50°S. On return from Neumayer, part of the transect was repeated as Run 6.1 from 54°S to the mooring position, and Run 6 continued with the Course Scale Survey of 6 north-south legs (Table 2, Fig. 2) 75 km apart to choose an area for a fine-scale survey. After three days pause for station work, the Fine-Scale Survey consisting of 11 north-south legs 13 km apart was undertaken in the northeast corner of the CSS area (Table 2, Fig. 3). The significant difference between the two surveys was that the 75 km separation of legs on the CSS was too large to resolve the small to meso-scale eddies (tens of km diameter) that are most likely to be responsible for plankton patchiness. Indeed, the plots show major, barely mappable changes from leg to leg of the CSS.

Data analysis

The OPC data stream

Logging of OPC data is significantly different from previous physical PSTAR data streams, in that the number of data values per second is highly variable. Every time a particle enters the OPC light beam, its size is recorded, given as an integer between 0 and 4095 related by a nominal calibration table to the ESD. The specification of the OPC is that it can sample up to 200 counts per second, but in practice it may saturate at rates over 100 counts per second. When towed under the SeaSoar at 4 m s^{-1}, the acrylic insert is used to reduce the counts per second.

On Ant XIII/2 counts varied typically between 0-10 per second below 100 m up to nearly 100 per second in the surface layer. Thus the sampling rate was high, on average 50 counts per second. In addition to the size of each particle, the OPC normally logs two other variables, a pulse from the pc's internal clock twice per second, and light attenuation also twice per second. The former provides a crude time base. The latter is monitored by the OPC so that it can constantly adjust the intensity of shadow that it recognises as a valid particle passing through the beam. Each of the three variables (count, clock and attenuance) is stored in 12 bits of a two-byte output word, the remaining 4 bits specifying which variable follows.

The time base provided by the internal clock is inadequate for SeaSoar operations, because the pc clock can and did drift by a second or more every 1 to 4 hours, yet time must be used to correlate the OPC data to the SeaSoar data, where the SeaSoar is profiling through the water column at rates up to 1-2 m s^{-1}. For this reason, the OPC suppliers were requested to make provision in the software for logging an additional time variable, an RS-

232 serial WEMPE clock stream in the NMEA-0183 format provided on *Polarstern*. This clock was also interfaced to the SeaSoar data stream, so providing the master time base for all data logged on the cruise. The clock data were inserted in the OPC data stream once per second as a 27 character ASCII string starting and ending with an ID serial byte 'C0' (hex).

A second clock option, to match the RVS clock format, was also provided by the manufacturers, but has not yet been used.

File transfer from the OPC pc to PSTAR computer

OPC data are normally logged to the pc hard disc in a continuous stream, which can only be read back later when data logging ceases. To avoid this, PC NFS software was added to the pc and data were logged over the ethernet to a hard disc partition on the PSTAR computer. This caused loss of about 20 minutes of data on one occasion when the emergency power switch in the PSTAR laboratory was accidentally knocked, but otherwise was entirely satisfactory. To avoid excessively long files, the switches to stop and start data logging on the pc were toggled once every 4 hours by a watchkeeper and the times noted to the second. For convenience, this was done one hour into each watch at the same time 4-hourly SeaSoar processing started. This procedure automatically started a new raw data file each time, updating the extension to the file name .Dnn from .D00 through .D01, .D02, etc. (For file naming conventions, see Appendix 1).

Two programs were written to read the raw data into PSTAR, POPCIN and a minor variant, POPAIN. A listing of POPCIN is given in Appendix 1. The major problems in these programs were (a) to resync when bad data are found in the data stream and (b) to create a time base. To establish sync for the WEMPE clock used on *Polarstern*, the program searched for two bytes 29 bytes apart and both containing 'C0'. The 27 byte string inbetween could then reliably be taken to be a clock string. Data of type 1 (target size) for POPCIN or type 2 (light attenuance) for POPAIN were then read into a buffer until a second clock string was found. A time base was then created by arbitrarily assuming that the targets (or light attenuance values) were evenly spaced in time between the two clock times. The times created are thus correct to a fraction of a second.

POPCIN and POPAIN will need to be modified for used with the RVS clock when used on an RVS ship.

These programs also contain code to use the pc's internal clock, which increments by 1 every 0.5 seconds and resets to zero when it reaches 4095. This code was tested during the trial Run 1 of SeaSoar when the WEMPE clock was not being logged. (This occurred because it was found that the WEMPE clock was not logged when a mouse was fitted to the pc.) However, these data were not processed further, so the code for the internal clock has not been exhaustively tested. If the internal clock has to be used, it should be possible to establish a time base accurate to a second by accurately noting the times when logging to file is started and stopped, then using pcalib to stretch the time base as necessary.

Further information on POPCIN and POPAIN and on running them in OPCEXEC0 is given in Appendix 1.

Further processing and data reduction

The 4-hour files created by POPCIN were large, typically containing 300,000 data cycles, hence the reason that only two variables (time and count) were stored, with the attenuance data (twice per second) read into a separate file. Nevertheless, these are the master files, in which no information is lost. Further processing, described here, separates the data into size classes and loses information in so doing. The size classes and averaging developed on ANT XIII/2 are cruise specific, and will probably need modification on future cruises. In particular, in oligotrophic areas, more careful averaging may be needed to separate the data into size classes over usefully small vertical intervals.

To proceed further, the OPC data must be merged (on time) with the SeaSoar CTD data to provide pressure and with navigation data to provide distance run (distrun). They can then be binned into pressure intervals (POPCAV) or gridded by binning on both pressure and distrun (PGRIDO). In both cases, the distrun variable is needed to calculate the volume of water that has flowed through the OPC in the binning interval, and hence to normalise the number of counts (total or in each size class) to counts/m^3.

The 4-hourly files were therefore appended over 12 (or more usually, with overlap, 16) hours to match the 12-hourly SeaSoar processing and then pressure and distrun from the SeaSoar-plus-navigation file were merged onto the OPC file to create a file containing 4 variables and usually over a million data cycles. The slightly tedious part of this procedure was to truncate (relatively few) data cycles from the ends of the appended OPC file to ensure that its times lay within the times of the SeaSoar file. This was easy if logging to the 4-hourly OPC files had been stopped one or two seconds before the end of the 4-hourly SeaSoar processing period and restarted a second after the start of the next period, but it would have been unreasonable to expect watchkeepers to achieve this always. After appending and truncating, further processing was done within an exec, OPCEXEC1 (see Appendix 1).

PGRIDO

PGRIDO was the first analysis program developed to provide a quick look at the OPC data split into a few size classes. The size classes chosen, 7-86, 87-686 and greater than 686 OPC units, corresponded to 250-1000 microns, 1-3mm and greater than 3mm using the default OPC ESD lookup table. These size classes are hard coded into PGRIDO and may need changing or generalising in future. PGRIDO is a relatively minor variant of PGRIDS, the SeaSoar gridding program. Both bin the data into particular depth and distrun ranges. However, where PGRIDS finishes by averaging the N data values in a particular bin, N is itself the output value for PGRIDO, divided by the volume of water that flowed through the OPC for counts in that bin. Of course, in addition to the total count N over all particles of whatever size, PGRIDO also calculates N_i (i = 1, 2, 3) for the three size classes chosen.

POPCAV

Given the typically large counts per second, it was later realized that data from a single profile could be binned over quite small depth ranges with better resolution of size classes than PGRIDO. Therefore POPCAV was written as a minor variant of PAVRGE (in the same relationship as PGRIDO to PGRIDS). As before, the normalized count Ń rather than the average is the output for each averaging interval. The size classes chosen were

again hard-coded into the program, and 5 were chosen corresponding to 250-350μ, 350-500μ, 500-800μ, 800-1000μ and greater than 1000μ (1 mm).

POPCAV was applied to all the 12-hour files of OPC counts (plus time, pressure and distrun), binning on pressure into 2 dbar intervals. Given mean up and down rates for the SeaSoar of only 0.5 m/s, there were on average say 4 seconds of data per bin, so several hundred counts. Reducing these to 5 size classes thus reduced the file size by an order of magnitude or more. Thus a more versatile program, with more size classes, would still reduce the data greatly in a productive region. A significant advantage of using POPCAV is that the file of attenuance values can be merged on time onto the averaged file, and PGRIDS can then be used for gridding and contouring in the usual way. This has been done for the data presented in this report.

PLOTYX

One further program was developed to display OPC data, but can be applied to any data set. Where the standard PSTAR program PLOTXY can plot several y-variables against a single x-variable, PLOTYX plots several x-variables against a single y-variable. However, PLOTYX is more related to PLOTPR than PLOTXY in that it plots profiles of each x-variable, offsetting the profiles by one x-axis tick mark after each profile. Thus PLOTYX is a version of PLOTGR which can display offset profiles of several variables (in different colours) simultaneously. Thus profiles of several size classes can be simultaneously displayed. Equally, however, it can be used to plot SeaSoar CTD data in profile form just as CTD casts are plotted, but with many (small) multiple profiles per page.

Data presentation

Three track plots are presented first, for the initial long run (Runs 2 and 3), the Course Scale Survey (Run 6) and the Fine Scale Survey (Run 8).

All the remaining plots are matched on two facing pages. Two contour plots on the lefthand page show total particle density and attenuance. The small tick marks along the x-axis are the positions at which the data were gridded. The profile plot at the bottom of the lefthand page presents the profiles of total particle density at those grid points. On the righthand page are shown five sets of profiles, where the total counts have been separated into the five size classes shown. Note the change of scale between the top three plots and the bottom two. Note that the x-axis scale for the profile plots refers to the "first profile", which may be absent! The zero line for an individual profile may reliably be determined from the x-axis tick mark that is just to the left of the base of the profile, as the values are always close to zero (less than one tick interval) at the maximum depth reached.

Runs 2 and 3 are presented over 4 double pages, 4° of latitude per page. Run 6.1 is presented twice, first at 4° on the page to match Run 3 which it repeated, second at 2.7° per page to match the remainder of Run 6. The 7 legs of Run 6 are presented one per double page, all on the same latitude scale, spanning 2.7° from 49.4°S to 52.1°S. The 11 legs of Run 8 are presented one per double page, all on the same latitude scale, spanning 1.1° from 49.7°S to 50.8°S.

Acknowledgements

Much of the success in logging and processing of SeaSoar and OPC data during this cruise was due to the excellent pre-cruise advice and support received from the Ocean Technology Division and Research Vessel Services at the Southampton Oceanographic Centre. We would like to thank in particular Ed Cooper for his detailed reconnaissance, and John Smithers, Martin Beney and Alan Taylor for their help at SOC preparing equipment, and hard work setting up hardware on the *Polarstern* in a wintery Bremerhaven, in November 1995. Our participation was partially supported by the MOD under Joint Grant funding for FAME (Quantifying the structure of Fronts And Mesoscale Eddies).

References

Allen J. T., H. Fischer, M. J. Griffiths, R. T. Pollard, J. F. Read and V. H. Strass (1996) *Acoustic Doppler Current Profiler data collected on Polarstern Cruise ANT XIII/2 4 December 1995 - 24 January 1996.* Southampton Oceanography Centre, Internal Document No. 8, 46 pp.

Griffiths M. J., J. T. Allen, T. J. P. Gwilliam, A. G. Naveira, R. T. Pollard and J. F. Read (1996) *SeaSoar operations and data collected on Polarstern Cruise ANT XIII/2 4 Dec 1995 - 24 Jan 1996.* Southampton Oceanography Centre, Report No. 1, 86 pp.

Herman A. W. (1992) Design and calibration of a new optical plankton counter capable of sizing small zooplankton. *Deep-Sea Research*, **39**, 395-415.

Read J. F., M. J. Griffiths and R. T. Pollard (1996) *Polarstern ANT XIII/2 4 December 1995 - 24 January 1996 Underway Data Report, comprising navigation, thermosalinograph, bathymetric and meteorological data.* Southampton Oceanography Centre, Internal Document No. 9, 49 p

Appendix 1 - Computing

Naming conventions

All OPC processing was done in directory /users/pstar/data/opc, which occupied a separate partition. Data names were all 8 characters long followed the convention ddcccnnn, where dd identifies the data type, ccc is the cruise identifier (set to 'ps1' - our first cruise on *Polarstern*), and nnn is a number incrementing from 001. The file names, and (usually) data names were thus:

opa13r08.d24 for the pc raw data files. Here 'op' signifies OPC, 'a13' is equivalent to 'ps1' but was created before 'ps1' was chosen for the cruise name, 'r08' signifies 'run 8' and 'd24' is the file extension created by the OPC pc.

o4ps1nnn PSTAR file for 4-hourly segments of raw OPC counts. Run 2 - 001-018; Run 3 - 019-031; Run 6 - 032-069; Run 8 - 071-094.

oaps1nnn PSTAR file for 4-hourly segments of raw attenuance. File numbers nnn match those for 'o4' files.

opps1nnn Appended 'o4' file 12 or 16 hours long. Run 2 - 001-005; Run 3 - 006-010; Run 6 - 011-022; Run 8 - 023-030.

opps1nnn.nav Same as 'op' files but with pressure and distrun merged on. Data name still opps1nnn.

atps1nnn Appended 'oa' file 12 or 16 hours long. File numbers nnn match those for 'op' files.

atps1nnn.nav As for opps1nnn, but for attenuation.

opps1nnn.av Created from opps1nnn.nav files by POPCAV with 2 dbar pressure intervals.

opps1rnn These files, one for each run (r02, r03, r06 and r08) were created by appending the '.av' files for each run, deleting overlaps, then merging in attenuance values from appended 'oa' files, one for each run. The data name matches the file name. These are the master files for the cruise covering total zooplanton counts, attenuance and 5 size classes. If further or different size classes are required, processing must be restarted from the 'opps1nnn.nav' files.

ogps1rnn Gridded files derived from opps1rnn by using PGRIDS with 8 dbar and 6 km bins.

run6.block and run8.block were created from ogps1r06 and 08 by copying out each of the survey lines, sorting on latitude, regridding by horizontal linear

interpolation onto even intervals of latitude about 0.05° apart, then re-appending to form a 3-dimensional file stored in order pressure, latitude, longitude; i.e. section 6.1, section 6.2, etc.

run6.map and run8.map were created from the '.block' files by using pinvxy to reorder the x, y and z variables so that they are in order latitude, longitude, pressure;; i.e. map at 5 dbar, map at 13 dbar, map at 21 dbar, etc.

Programs developed for OPC processing

POPCIN and POPAIN

These programs are most often run within OPCEXEC0, where the detailed input requirements can be inferred. Each program asks for the name of the pc input file ('opa13 is hard coded and will need to be generalised on a subsequent cruise) and prints the ASCII header of that file so that the user can check if it is the correct one. The start time given by the pc's internal clock is the primary guide, so it is advisable to reset the pc clock from time to time so that it is accurate within a minute or two. If the start time matches the time when logging was started (read and written down from an accurate ship's clock), answer 'y' to the 'continue?' question, and enter the type of clock (WEMPE or internal), the output file name, the data name, and the accurate start time. On completion, the program lists the number of times that sync of the time base has been lost. This can be a guide to cable problems.

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C *****POPCIN
C
      PROGRAM POPCIN
      IMPLICIT REAL*8 (A-H,O-Z,a-h,o-z)
C
C......read in data from sequential file
C......into PSTAR file
C
#include "datadf.h"
#include "psio.h"
C
      LOGICAL NOMOD
      CHARACTER*1 ANS,yes,yes2
      CHARACTER*8 PROG
      CHARACTER*80 filename
      integer plen
      byte b(1600)
      character*1 bb(1600),c(1650)
      character*6 yemoda,homise
      integer iii(1650)
      character*1600 bchar
      DIMENSION count(2000), time(2000)
      equivalence (b,bchar),(b,bb)
      data yes/'Y'/,yes2/'y'/
C
      DATA PROG/'POPCIN01'/
C
C......Set DD to blanks
      DATA INSTMT/'OPC          '/,DATNAM/'opa13nnn'/
      DATA PLATYP/'SHIP     '/,PLTNUM/'         '/,PLTNAM/'
'/
      DATA FLDNAM/IFLDXX*'         '/,FLDUNT/IFLDXX*'         '/
      data absent/ifldxx*-999./
      data prefil/'         '/,postfl/'         '/
      data recint/2*'         '/
C
C......initilise
```

```
C
      call proghd(prog)
C
C......indisk must be in range 10 to 15, but I think output file
C......will be 10, so avoid it. This is not robust code, but will
C......do for now.
      indisk=13
      nblen=1600
      magic=magvar
      nrows=0
      nplane=0
      icent=1900
      alat=0.
      along=0.
      depthi=0.
      depthw=4000.
C......clock sync word is set to OK
      itsync=0
C......set iclker to keep track of clock sync problems
      iclker=0
C      set VERSON to '  ', VERSON will correct it
      vers='  '
C......open and check input file
C
      filename='/users/pstar/data/opc/opal3'
      ilen=plen(filename,80)
      write(ioitt,501)
 501  format(' Job will get input from ',
     & '~pstar/data/opc/opa13r01.d00'/
     & ' Enter r01.d00 or similar')
      read(initt,'(a7)')filename(ilen+1:ilen+7)
C     write(ioitt,*)filename
      open (unit=indisk,file=filename,access='DIRECT',
     & form='UNFORMATTED',RECL=1600,
     & status='OLD',err=600)
C     write(ioitt,*)filename
C......set record pointer
      irec=1
C......read first record and check it is OK with user
      read(indisk,REC=irec,ERR=700,IOSTAT=IOST)bb
 4    continue
      irec=irec+1
      do 5 i=1,1600
      iii(i)=b(i)
      if(iii(i).lt.0)iii(i)=iii(i)+256
      c(i)=bb(i)
 5    continue
C     write(ioitt,'(165(10I4/))')(iii(i),i=1,1650)
      write(ioitt,504)bchar(1:24)
 504  format('First record read in is:'/
     & a24/'Is this correct? (y/n)')
      ans=yes
      read(initt,'(a1)')ans
      if(ans.ne.yes.and.ans.ne.yes2)stop 'Check input file'
C......find out which clock to use. At present there are two
C......0=OPC PC's internal clock; 1=Polarstern Wempe clock
      write(ioitt,508)
 508  format(' Which clock provides time? Enter'/
```

```
     & ' 0 for OPC pc internal clock'/
     & ' 1 for Polarstern Wempe clock')
      read(initt,*)iclock
      if(iclock.ne.0.and.iclock.ne.1)stop 'Check clock'
C......if the internal clock is in use, we must check that it
increments
C......sequentially. If not, data has lost sync. So we need to set a
C......start value. I can set it to -999
      ipclk0=-999
C
C......Open output file
C
      CALL OPENOT(IODISK)
      IF(IODISK.EQ.-999) STOP 'No output file'
C......enter header information
C......a new file so mods may be made
C
      NOMOD=.FALSE.
C......get environ vars
      pltnam='SHIPNAME'
      call envstr(pltnam)
      pltnum='CRUISE'
      call envstr(pltnum)
      nofldS=2
      fldnam(1)='time    '
      fldnam(2)='count   '
      fldunt(1)='seconds '
      fldunt(2)='        '
C......get data name
      write(ioitt,505)
 505  format(' Enter 8-char Data Name like opa13nnn')
      read(initt,'(a8)')datnam
C
C......get a suitable start time, which the user should have logged
C......to the second from whatever master clock is in use
      write(ioitt,509)
 509  format('Enter start time as YYMMDD,HHMMSS')
      read(initt,*)iymd,ihms
C
      call prnjul(iymd,julbeg)
      begsec=prtsec(ihms)
C......set time0 to seconds past start time
      time0=0.
C......reset the input buffer pointer
      ipos=28
C
C......main count proc starts here
C
C......no. of data cycles in output buffer is ipoint
      ipoint=0
      norecs=0
C......set ncount=0
 10   ncount=0
C......start position in buffer B is ipos
C......length of buffer B is nblen
C      if buffer is nearly used, then repack and read more
 20   if((nblen-ipos).lt.30)then
 120     do 15 i=ipos,nblen
```

```
         c(i-ipos+1)=c(i)
 15      iii(i-ipos+1)=iii(i)
         nblen=nblen-ipos+1
         ipos=1
         read(indisk,REC=irec,ERR=800,IOSTAT=IOST,end=500)bb
C.........copy data out of buffer into split buffers
 16      irec=irec+1
         do 7 i=1,1600
         j=nblen+i
         iii(j)=b(i)
         if(iii(j).lt.0)iii(j)=iii(j)+256
         c(j)=bb(i)
 7       continue
         nblen=nblen+1600
C        write(ioitt,'(165(10I4/))')(iii(i),i=1,1650)
C........if time word is out of sync, we may need to skip back
C........into the correcting code
         if(itsync.ne.0)goto 27
      endif
C......read 1 byte, look for 1,2,3 or 12 in first 4 bits
      itype=iii(ipos)/16
C     calling(itype,ioitt)
C
C......major loop is here
C
C......commonest value should be 1, particle count, so put first
      if(itype.eq.1)then
C.........read count
         ncount=ncount+1
         count(ipoint+ncount)=(iii(ipos)-16)*256+iii(ipos+1)
         ipos=ipos+2
         goto 20
      endif
C.....ignore irradiation, type 2 for now
      if(itype.eq.2)then
         ipos=ipos+2
         goto 20
      endif
C......type 3 is the internal clock
      if(itype.eq.3)then
C.........skip if it is not in use
         if(iclock.ne.0)then
            ipos=ipos+2
            goto 20
         endif
C.........it is in use, so increment the time
C.........first we must check the clock word to see if it is in sync
         ipclk1=(iii(ipos)-48)*256+iii(ipos+1)
C........has clock incremented by 1?
         igap=ipclk1-ipclk0
         if(igap.eq.1)then
C........yes it has, this is normal result
 21         time1=time0+0.5
            ipos=ipos+2
            ipclk0=ipclk1
            goto 40
         endif
C.........no, it hasn't. Is it at start? If so, treat as normal
```

```
C........DATA LOSS COULD OCCUR HERE IF FIRST CLOCK FOUND IS BAD
         if(ipclk0.eq.-999)goto 21
C........is the jump from 4095 to 0? Also normal
         if(ipclk0.eq.4095.and.ipclk1.eq.0)goto 21
C........anything else is not normal. We need to search for next
C........clock word. First set out-of-sync warnings.
         write(ioitt,510)ipclk0,ipclk1
 510     format('Loss of internal clock sync, clock jump ',i4,
     &   ' to ',i4)
         itsync=-999
C........we have the first clock word, ipclk1.
C........now look for a second clock word, in increments of 2bytes.
C........but we must keep first clock in buffer.
         ipos0=ipos
 28      ipos=ipos+1
          calling(14,ioitt)
         if((nblen-ipos).lt.30)then
            ipos1=ipos
            ipos=ipos0
            goto 120
         else
            goto 26
         endif
 27      ipos=ipos1-ipos0+1
C........is this a clock word?
 26      if((iii(ipos)/16).ne.3)goto 28
C........unpack second clock word
               CALLING(13,ioitt)
         ipclk2=(iii(ipos)-48)*256+iii(ipos+1)
C........are 1st and 2nd clock words in sync?
C........and an even number of bytes apart
         igap=ipclk2-ipclk1
         ieven=ipos-ipos0
         ieven=mod(ieven,2)
         write(ioitt,*)ipclk1,ipclk2,igap,ieven,ipos0,ipos
         if(igap.eq.1.and.ieven.eq.0)goto 29
         if(ipclk1.eq.4095.and.ipclk2.eq.0.and.ieven.eq.0)goto 29
C........no, they're not. Go back to ipos0+1 and start again
         ipclk1=ipclk2
         ipos0=ipos
         goto 28
C........at last we have two times in sync
C........So we ignore all data between ipclk0 and ipclk1
 29      ncount=0
C........and calculate the clock jump between ipclk0 and 1.
         igap=ipclk1-ipclk0
C........if it is negative, then it has gone through 4096
         if(igap.le.0)igap=igap+4096
         write(ioitt,511)ipclk1,igap
 511     format('Sync regained at clock value ',i4/
     &   'Time gap is 0.5*',i4,' seconds')
         time0=time0+0.5*igap
         itsync=0
         ipclk0=ipclk1
         ipos=ipos0+2
         goto 10
      endif
C......type 12 is the Wempe clock
```

```
      if(itype.eq.12)then
C......debug
C         if((ipos+28).gt.1600)write(ioitt,*)ipos,nblen
C.........first check for a complete clock stream with '12' 28 chars
C.........further on.
          if((iii(ipos+28)/16).ne.12)then
C............not a good clock stream, start to resync
C............first keep count of no. of clock problems
             iclker=iclker+1
             ipos=ipos+1
             goto 20
          endif
C.........clock stream is good, but
C.........skip if it is not in use
          if(iclock.ne.1)then
             ipos=ipos+29
             goto 20
          endif
C.........it is in use, so increment the time
          do 35 i=1,6
          homise(i:i)=c(ipos+i)
 35       continue
          yemoda(1:1)=c(ipos+20)
          yemoda(2:2)=c(ipos+21)
          yemoda(3:3)=c(ipos+15)
          yemoda(4:4)=c(ipos+16)
          yemoda(5:5)=c(ipos+12)
          yemoda(6:6)=c(ipos+13)
          read(homise,'(I6)',ERR=37)jhms
          read(yemoda,'(I6)',ERR=37)jymd
C         write(ioitt,*)jymd,jhms
C......update pointer before repacking time
          ipos=ipos+29
          call prnjul(jymd,julend)
          endsec=prtsec(jhms)
          time1=endsec-begsec+(julend-julbeg)*86400.
          goto 40
C.........an unsuitable character has been encountered in time word
C.........ignore this time altogether
 37       ipos=ipos+29
             calling(1,ioitt)
             write(ioitt,*)ipos
C            write(ioitt,'(165(10I4/))')(iii(i),i=1,1650)
          goto 20
      else
C......type is none of 1,2,3,12 so out of sync, try next byte
          ipos=ipos+1
          goto 20
      endif
C......time interval
C......here is where we create the time variable
C......there may be no counts in the interval
 40   if(ncount.eq.0)goto 60
      tint=(time1-time0)/(ncount+1)
      do 50 i=1,ncount
      time(ipoint+i)=time0+i*tint
 50   continue
 60   time0=time1
```

```
C......write out if output buffer is reasonably full
      ipoint=ipoint+ncount
      if(ipoint.gt.1500)then
         call otdata(iodisk,1,norecs+1,ipoint,time(1),noflds,
     &   norecs+ipoint)
         call otdata(iodisk,2,norecs+1,ipoint,count(1),noflds,
     &   norecs+ipoint)
C........update norecs
         norecs=norecs+ipoint
C........zeroise output buffer pointer
         ipoint=0
      endif
C......carry on reading from input buffer
      goto 10
C
C......get here if end of input file found
C......write out any data in output buffer
 500  if(ipoint.gt.0)then
         call otdata(iodisk,1,norecs+1,ipoint,time(1),noflds,
     &   norecs+ipoint)
         call otdata(iodisk,2,norecs+1,ipoint,count(1),noflds,
     &   norecs+ipoint)
C........update norecs
         norecs=norecs+ipoint
      endif
C
C......wrap up
C
      do 70 i=1,noflds
   70 CALL UPRLWR(IODISK,I,1,NORECS,ALRLIM(I),UPRLIM(I),ABSENT(I),
     &NOFLDS,NORECS)
C
      CALL PFINIS(IODISK,PROG,
     & MAGIC,NOFLDS,NORECS,NROWS,NPLANE,ICENT,IYMD,IHMS,
     & FLDNAM,FLDUNT,ALRLIM,UPRLIM,ABSENT,
     & ALAT,ALONG,DEPTHI,DEPTHW,OPWRIT,RAWDAT,PIPEFL,ARCHIV,VERS,
     & DATNAM,PREFIL,POSTFL,PLATYP,PLTNUM,RECINT,PLTNAM,INSTMT,COMENT)
C
C......write out clock sync problems
 520  format('No. of sync problems =',I4)
      write(ioitt,520)iclker
      STOP
 600  stop 'error opening input file'
 700  write(ioitt,*)IOST
      goto 4
 800  write(ioitt,*)IOST
      goto 16
      END
      subroutine ing(n,ioitt)
      integer i(12)
      data i/12*0/
      if(n.gt.12.or.n.lt.1)goto 10
      i(n)=i(n)+1
      if(i(n).gt.10)return
 10   write(ioitt,500)n
 500  format(' Calling',i4)
      return
      end
```

POPCAV

POPCAV is a fairly minor variant of PAVRGE. The input required from the user is identical. Variables named 'count' and 'distrun' must be present in the input stream. All variables are averaged in the normal way except count. Five extra output variables are created for the 5 size classes. The actual values of 'count' are used only to determine into which size class a particular data cycle falls. Output values are the number of data cycles in the bin normalised by the volume of water.

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C *****POPCAV
C
      PROGRAM POPCAV
C
C     Created by RTP in 3/1/96
C
      IMPLICIT REAL*8 (A-H,O-Z,a-h,o-z)
C
C......Special version of pavrge for opc data from SeaSoar, once
C......press and distrun have been merged from the ss file. Instead
C......of averaging, the total count is kept, divided by the volume
C......of water that has passed through the tunnel, to give
C......counts per m3.
C......Distrun is averaged in the usual way.
C......Averaging uses SCANUP on one var
C......Empty intervals are filled with ABSENT data
C
#include "datadf.h"
#include "psio.h"
C
      PARAMETER (IRECD2=IRECXX/2)
      INTEGER NARR(IRECD2)
      DIMENSION ARR(IRECD2),BUFA(IRECXX),bufb(irecxx)
C......this version bins into five size classes as well as total
      dimension bufc(irecd2,5)
      dimension size(6),anum(5)
      CHARACTER*8 TIME,PROG
      COMMON/WORK/BUFA
C
       DATA PROG/'POPCAV01'/
       DATA TIME/'time    '/
C......5 size classes, 250-350microns, 350-500,500-800,800-1000
C......and greater than 1000. Standard ESD from manual table
C......gives smallest count in each size class below
      data size/6.5,11.5,23.5,56.5,86.5,4095.5/
C
C......Initialise all streams
C
      NUMWRD=IRECXX
C
      CALL PROGHD(PROG)
C
      CALL OPENIN(INDISK)
      IF(INDISK.EQ.-999)STOP 'no input file'
C
C......And an output file
```

```
C
      CALL OPENOT(IODISK)
      IF(IODISK.EQ.-999) STOP 'No output file'
C
      CALL READPR(INDISK,
     & MAGIC,NOFLDS,NORECS,NROWS,NPLANE,ICENT,IYMD,IHMS,
     & FLDNAM,FLDUNT,ALRLIM,UPRLIM,ABSENT,
     & ALAT,ALONG,DEPTHI,DEPTHW,OPWRIT,RAWDAT,PIPEFL,ARCHIV,VERS,
     &
DATNAM,PREFIL,POSTFL,PLATYP,PLTNUM,RECINT,PLTNAM,INSTMT,COMENT)
C......Get averaging info
C
 10   WRITE(IOITT,501)
 501  FORMAT(' Enter NUMBER of scan variable,START,COUNT,XMNDST')
C     READ(INITT,*)ISCAN,START,COUNT,XMNDST
      NVARS=1
      NREAL=3
      ARR(1)=-999
      ARR(2)=-999
      ARR(3)=0.
      CALL VINPUT(FLDNAM,ISCAN,ARR,NVARS,NREAL,NOFLDS)
      START=ARR(1)
      COUNT=ARR(2)
      XMNDST=ARR(3)
      IF(ISCAN.LT.1.OR.ISCAN.GT.NOFLDS.OR.
     &   START.EQ.-999.OR.COUNT.EQ.-999)GO TO 10
C
C......Find which variable is 'distrun' and which
C......is 'count'.
      idist=-1
      icount=-1
      do 45 k=1,nofldS
      if(fldnam(k).eq.'distrun ')idist=k
      if(fldnam(k).eq.'count   ')icount=k
 45   continue
      if(idist.eq.-1)stop 'No distrun variable'
      if(icount.eq.-1)stop 'No count variable'
C......change output name for count
      fldnam(icount)='zoodens '
      fldunt(icount)='c-nt/dm3'
C...... deal with extra output fields
      nofld2=noflds+5
      fldnam(noflds+1)='250-350u'
      fldnam(noflds+2)='350-500u'
      fldnam(noflds+3)='500-800u'
      fldnam(noflds+4)='.8-1.0mm'
      fldnam(noflds+5)='.gt.1.mm'
      do 1 ii=1,5
      absent(noflds+ii)=absent(icount)
 1    fldunt(noflds+ii)=fldunt(icount)
C
C......Is TIME the scan variable.
C
      IF(FLDNAM(ISCAN).NE.TIME)GO TO 40
C
C......Yes, reset START to be relative to start of day
C......Convert IHMS to SECS
C
```

```
      CALL PRNSEC(IHMS,ISEC)
      START=START-ISEC
C
C......Scan the indep var
C
 40   NIN=IRECD2
      LSTCYC=0
      NOREC2=0
 50   CALL SCANUP(INDISK,ISCAN,COUNT,LSTCYC,START,XMNDST,
     & NARR,ARR,NIN,NOUT,NOFLDS,NORECS)
C
C......SCANUP has returned NOUT values, so NOUT-1 intervals
C......to be averaged and written out
C
C
C......Deal with scan var first, to free ARR as BUFFER for
C......other vars
C
C......Set ARR to midpoint of scan interval
C
      DO 53 I=1,NOUT-1
 53   ARR(I)=ARR(I)+0.5*COUNT
C
C......Write out
C
      CALL OTDATA(IODISK,ISCAN,NOREC2+1,NOUT-1,ARR,
     & NOFLD2,NOREC2+NOUT)
C
C......Now loop to other vars
C
      DO 120 K=1,NOFLDS
      IF(K.EQ.ISCAN)GO TO 120
C    *****  NSCAN - POSITION IN CORE ARRAY FOR NARR,ARR
C    *****  CURRENTLY REACHED - NSCANE = NEXT SCAN
      NSCAN=1
      NSCANE=2
C    *****  NDAT - DATA CYCLE ON WHOLE INPUT FILE CURRENTLY REACHED
      NDAT=NARR(1)-1
C    *****  SUM AND NUM HOLD SUM AND NUMBER IN CURRENT AVERAGE
      SUM=0.0
      NUM=0
      adist=0.
      do 54 ii=1,5
 54   anum(ii)=0.
C
C......Read input from D.C. NARR(1) to NARR(NOUT)-1
C
      NL=NUMWRD
      N1=NARR(1)
      N2=NARR(NOUT)-1
      DO 100 I=N1,N2,NUMWRD
      IF((I+NL-1).GT.N2)NL=N2-I+1
      CALL INDATA(INDISK,K,I,NL,BUFA,NOFLDS,NORECS)
C......to process count we need distrun as well
      if(k.eq.icount)call indata(indisk,idist,i,nl,bufb,
     & noflds,norecs)
      DO 90 J=1,NL
      NDAT=NDAT+1
```

```
 55   IF(NDAT-NARR(NSCANE))60,65,65
C    *****  STILL IN AVERAGING INTERVAL
 60   IF(BUFA(J).EQ.ABSENT(K))GO TO 90
C......two options, for count or not-count
      if(k.ne.icount)then
         SUM=SUM+BUFA(J)
         NUM=NUM+1
      else
C........for count, I dont need the sum, only the delta distrun
C........of water. Cheat here, using either forward or backward
C........difference to keep within array bounds. At near-constant
C........tow speed, errors are totally negligible
         k1=j-1
         if(k1.eq.0)k1=1
         k2=k1+1
C........there is just a possibility that nl=1, so
         if(k2.gt.nl)k2=nl
         adist=adist+bufb(k2)-bufb(k1)
         num=num+1
C........now bin into the 5 size classes
         do 62 ii=1,5
         if(bufa(j).gt.size(ii).and.bufa(j).lt.size(ii+1))
     &    anum(ii)=anum(ii)+1.
 62      continue
      endif
      GO TO 90
C    *****  END OF AVERAGING INTERVAL REACHED, CHECK IF EMPTY
 65   IF(NUM)70,70,75
C    *****  EMPTY FILL WITH ABSENT DATA
 70   continue
      ARR   (NSCAN)=ABSENT(K)
C......except for count, when there may well be no counts in the
C......interval
      if(k.eq.icount)then
         arr(nscan)=0.
         do 71 ii=1,5
         bufc(nscan,ii)=0.
 71      continue
      endif
      GO TO 80
C    *****  THERE IS DATA IN INTERVAL, TAKE AVERAGE
 75   continue
C......again processing is different for count
      if(k.ne.icount)then
         arr(NSCAN)=SUM/NUM
      else
         arr(nscan)=num
C........Normalise counts to counts per m3. Vol of water that has
C........passed is distrun*1000(m)*0.001(m2 - aperture). So no
C........scaling is necessary. adist already contains m3 of water
         if(adist.eq.0.)then
            arr(nscan)=0.
            do 76 ii=1,5
            bufc(nscan,ii)=0.
 76         continue
         else
            arr(nscan)=arr(nscan)*0.001/adist
            do 77 ii=1,5
```

```
            bufc(nscan,ii)=anum(ii)*0.001/adist
 77         continue
         endif
      endif
      SUM=0.
      NUM=0
      adist=0.
      do 78 ii=1,5
 78   anum(ii)=0.
 80   NSCAN=NSCAN+1
      NSCANE=NSCANE+1
      GO TO 55
 90   CONTINUE
 100  CONTINUE
C
C
C......Loop stops just before statement 55 satisfied for last
C......output cycle
C......So check if last interval empty
C
      IF(NUM)105,105,110
C
C......Empty
C
 105  ARR(NSCAN)=ABSENT(K)
      if(k.eq.icount)then
         arr(nscan)=0.
         do 106 ii=1,5
 106     bufc(nscan,ii)=0.
      endif
      GO TO 112
C
C......Not empty, take average
C
 110  continue
      if(k.ne.icount)then
         ARR(NSCAN)=SUM/NUM
      else
         arr(nscan)=num
         if(adist.eq.0.)then
            arr(nscan)=0.
            do 111 ii=1,5
 111        bufc(nscan,ii)=0.
         else
            arr(nscan)=arr(nscan)*0.001/adist
            do 109 ii=1,5
 109        bufc(nscan,ii)=anum(ii)*0.001/adist
         endif
      endif
C
C......If several values of NARR(NSCANE) are the same, we may
C......get here before filling NOUT-1 data cycles
C......If so, rest must be absent data
C
 112  IF(NSCAN.EQ.(NOUT-1))GO TO 115
      DO 113 I=NSCAN+1,NOUT-1
      ARR(I)=ABSENT(K)
C......or zero for count
```

```
      if(k.eq.icount)then
         arr(i)=0.
         do 114 ii=1,5
 114     bufc(i,ii)=0.
      endif
 113  continue
C
C......End of scan array, write out averaged D.C.S
C
 115  CALL OTDATA(IODISK,K,NOREC2+1,NOUT-1,ARR,NOFLD2,
     & NOREC2+NOUT-1)
C......and for icount, write out the 5 extra variables
      if(k.ne.icount)goto 120
      do 116 ii=1,5
      call otdata(iodisk,noflds+ii,norec2+1,nout-1,
     & bufc(1,ii),nofld2,norec2+nout-1)
 116  continue
C
C......Loop for next variable
C
 120  CONTINUE
C
C......Update NOREC2
C
      NOREC2=NOREC2+NOUT-1
C
C......Has SCANUP got to end? LSTCYC=0, if not get more
C
      IF(LSTCYC.NE.0)GO TO 50
C
C......Yes, reset limits
C
      WRITE(IOITT,502)
  502 FORMAT(' Calculating limits - please wait')
C
      DO 20 I=1,NOFLD2
   20 CALL UPRLWR(IODISK,I,1,NOREC2,ALRLIM(I),UPRLIM(I),ABSENT(I),
     &NOFLD2,NOREC2)
C
      CALL PFINIS(IODISK,PROG,
     & MAGIC,NOFLD2,NOREC2,NROWS,NPLANE,ICENT,IYMD,IHMS,
     & FLDNAM,FLDUNT,ALRLIM,UPRLIM,ABSENT,
     & ALAT,ALONG,DEPTHI,DEPTHW,OPWRIT,RAWDAT,PIPEFL,ARCHIV,VERS,
     &
DATNAM,PREFIL,POSTFL,PLATYP,PLTNUM,RECINT,PLTNAM,INSTMT,COMENT)
C
      STOP
      END
```

PGRIDO

PGRIDO is a relatively minor variant of PGRIDS, but only three variables, 'distrun', 'press' and 'count' must be given. Three extra output variables are created, for the three size classes that are hard-coded in. These three variables and 'count' are processed exactly as described under POPCAV above. Gridding input required is identical to that required by PGRIDS.

```
C *****PGRIDO
C
      PROGRAM PGRIDO
C......Program to create a gridded file suitable for contouring
C...... This is a new version by RTP Dec 95 to handle OPC data
C
C     PSTAR/RVS version by JBN (18-1-89)
C
      IMPLICIT REAL*8 (A-H,O-Z,a-h,o-z)
C
C......from data along a sawtooth track
C
#include "datadf2.h"
#include "psio.h"
C
      CHARACTER*8 PROG
      INTEGER NPOS(IFLDXX)
      dimension datarr(ifldxx,4),count(ifldxx,4)
      DIMENSION ZARR(IRECXX),BIN(IRECXX),BUFA(IRECXX),xarr(irecxx)
C
      DATA PROG/'PGRIDO01'/
C
      CALL PROGHD(PROG)
C......Set max rows and cols
      MAXROW=IFLDXX
C......set max buffer size
      maxbuf=irecxx-1
C
C......Open files
      CALL OPENIN(INDISK)
      IF(INDISK.EQ.-999) STOP 'No input file'
C
      CALL READPR(INDISK,
     & MAGIC,NOFLDS,NORECS,NROWS,NPLANE,ICENT,IYMD,IHMS,
     & FLDNAM,FLDUNT,ALRLIM,UPRLIM,ABSENT,
     & ALAT,ALONG,DEPTHI,DEPTHW,OPWRIT,RAWDAT,PIPEFL,ARCHIV,VERS,
     &
DATNAM,PREFIL,POSTFL,PLATYP,PLTNUM,RECINT,PLTNAM,INSTMT,COMENT)
C
      CALL OPENOT(IODISK)
      IF(IODISK.EQ.-999) STOP 'No output file'
C
C......List of vars to be gridded, starting with X,Y
      WRITE(IOITT,502)
 502  FORMAT(' Enter var names/numbers for distrun, press, count,'
     & '. Three vars only.'
     & /' X must be monotonic increasing')
   10 CONTINUE
C     CALL READVR(NPOS,NOFLD2,NOFLDS)
```

```
      CALL READNM(FLDNAM,NPOS,NOFLD2,NOFLDS)
      if(nofld2.ne.3)stop 'Exactly 3 input fields required'
C
C......Make up output DD
      DO 15 I=1,NOFLD2
        FLDNA2(I)=FLDNAM(NPOS(I))
        FLDUN2(I)=FLDUNT(NPOS(I))
        ABSEN2(I)=ABSENT(NPOS(I))
 15   CONTINUE
C......add the new OPC fields
      nofld2=6
      fldna2(3)='zoodens '
      fldna2(4)='z.2-1.mm'
      fldna2(5)='z1.-3.mm'
      fldna2(6)='z.gt.3mm'
      do 16 i=3,6
      absen2(i)=absen2(3)
 16   fldun2(i)='count/m3'
      ABSX=ABSENT(NPOS(1))
      ABSY=ABSENT(NPOS(2))
C
C.....Get Y range info
      NYVAR=NPOS(2)
 20   WRITE(IOITT,504)
 504  FORMAT(' Y grid min, max, DELTAY?')
      READ(INITT,*)YMIN,YMAX,YSTEP
      IF(YMAX.LE.YMIN.OR.YSTEP.LE.0.)GO TO 25
      NROWS=(YMAX-YMIN)/YSTEP+1.1
      IF(NROWS.GT.MAXROW)GO TO 26
C     IF(YMIN.LT.ALRLIM(NPOS(2)))GO TO 20
C     IF((YMIN+(NROWS-1)*YSTEP).GT.UPRLIM(NPOS(2)))GO TO 20
      GOTO 30
C
C......errors
 25   WRITE(IOITT,506)
 506  FORMAT(' YMAX less than YMIN or YSTEP negative, respecify')
      GOTO 20
 26   WRITE(IOITT,507)MAXROW
 507  FORMAT( ' Too many rows implied, maximum ',I3,', respecify')
      GOTO 20
C
C......Ok, reset YMAX to last row value
 30   YMAX=YMIN+(NROWS-1)*YSTEP
C
C......get X range info
 35   WRITE(IOITT,503)
 503  FORMAT(' Enter XMIN, XMAX, separation of X grid points, X',
     & ' search ''radius''')
      READ(INITT,*)XMIN,XMAX,XSTEP,XRAD
C
C......check
      IF(XMAX.GT.XMIN.AND.XSTEP.GT.0..AND.XRAD.GT.0.)GOTO 36
      WRITE(IOITT,508)
 508  FORMAT(' XMAX less than XMIN or XSEPARATION or XRADIUS
negative',
     & ', respecify')
      GOTO 35
C
```

```
C......get file limits for x
 36   NXVAR=NPOS(1)
      XLOWER=ALRLIM(NXVAR)
      XUPPER=UPRLIM(NXVAR)
C
C......find the first value of x s.t. (X-XRAD).GE.XLOWER, so that X
C......has full search range
      NX=0
      TEST=XLOWER-XMIN+XRAD
      IF(TEST.LE.0.)GOTO 40
      NX=TEST/XSTEP
 40   CONTINUE
C
C......set X to this value
      X=XMIN+NX*XSTEP
C
C......set XMAX in range of file data
      IF((XUPPER-XRAD).LT.XMAX)XMAX=XUPPER-XRAD
C
C.....ensure at least two columns
      IF((X+XSTEP).LE.XMAX)GOTO 45
C
C......complain
      WRITE(IOITT,509)
 509  FORMAT(' Too few columns possible with these values, check',
     & ' file limits')
      GOTO 35
C
C......***MAIN LOOP***
C
C......X is now the x value for the first column
C......set NOREC2 to no. of values written out sofar
 45   NOREC2=0
C
C......set the search limits on X
 50   XSMIN=X-XRAD
      XSMAX=X+XRAD
C
C......if X greater than XMAX, we've got to end
      IF(X.GT.XMAX)GOTO 400
C
C....find data cycle limits to scan using binary chop fast search
      CALL GETCYC(INDISK,NXVAR,1,NORECS,XSMIN,NXMIN,NOFLDS,NORECS)
      CALL GETCYC(INDISK,NXVAR,1,NORECS,XSMAX,NXMAX,NOFLDS,NORECS)
      NXMAX=NXMAX-1
C......for OPC, we need the difference in distrun between current
C......data cycle and the previous. To avoid problems, always skip
C......the first data cycle in the file. This is an insignificant
C......loss. Thus
      if(nxmin.eq.1)nxmin=2
C......if NXMAX is less than NXMIN there is no data in interval
      IF(NXMAX.GE.NXMIN)GOTO 80
C
C......fill output columns with absent data
      DO 70 J=3,NOFLD2
        DO 65 K=1,NROWS
          BIN(K)=ABSEN2(J)
 65     CONTINUE
```

```
        CALL OTDATA(IODISK,J,NOREC2+1,NROWS,BIN,NOFLD2,NOREC2+NROWS)
 70   CONTINUE
      GOTO 305
C
C......sort all data between cycles NXMIN and NXMAX
 80   CONTINUE
      YBIN=1.5*YSTEP-YMIN
      YMULT=1./YSTEP
C     DO 300 J=3,NOFLD2
C......replace this loop, as only count to be read in
      j=3
C
C......Sort all data in search range into NROWS BINS
C......First zero all BINS and BIN counters
      DO 100 K=1,NROWS
      do 100 l=1,4
        datarr(k,l)=0.
        count(k,l)=0.
 100  CONTINUE
      NUM=MAXbuf
      DO 150 K=NXMIN,NXMAX,MAXbuf
        IF((K+NUM-1).GT.NXMAX)NUM=NXMAX-K+1
        CALL INDATA(INDISK,NYVAR,K,NUM,BUFA(2),NOFLDS,NORECS)
        CALL INDATA(INDISK,NPOS(J),K,NUM,ZARR(2),NOFLDS,NORECS)
        call indata(indisk,nxvar,k-1,num+1,xarr(1),noflds,norecs)
        DO 120 L=2,NUM+1
C.........check for absent data
          IF(BUFA(L).EQ.ABSY)GOTO 120
          IF(ZARR(L).EQ.ABSENT(NPOS(J)))GOTO 120
C.........find BIN
          IBIN=(BUFA(L)+YBIN)*YMULT
          IF(IBIN.LT.1.OR.IBIN.GT.NROWS)GO TO 120
C......i have read in counts into zarr. They go into depth bin IBIN
C......but I must sort them further
          count(ibin,1)=count(ibin,1)+1.
          datarr(ibin,1)=datarr(ibin,1)+xarr(L)-xarr(L-1)
C......    sort into 3 sizeclasses, 7-86, 87-686,.gt.686
C..........corresponding to 250-1000,1000-3000,.gt.3000 microns
          if(zarr(l).gt.6.5.and.zarr(l).lt.86.5)then
            count(ibin,2)=count(ibin,2)+1.
C           datarr(ibin,2)=datarr(ibin,2)+xarr(L)-xarr(L-1)
          endif
          if(zarr(l).gt.86.5.and.zarr(l).lt.686.5)then
            count(ibin,3)=count(ibin,3)+1.
C           datarr(ibin,3)=datarr(ibin,3)+xarr(L)-xarr(L-1)
          endif
          if(zarr(1).gt.686.5)then
            count(ibin,4)=count(ibin,4)+1.
C           datarr(ibin,4)=datarr(ibin,4)+xarr(L)-xarr(L-1)
          endif
 120    CONTINUE
 150  CONTINUE
C
C......Binning of data for this column is complete.
C......Normalise counts to counts per m3. Vol of water that has
C......passed is  (distrun)(km)*1000(m)*0.001m2 where 1000
C......converts km to m and area of aperture is 0.001 m2. So no
C......scaling is necessary. Datarr already contains m3 of water.
```

```
      DO 200 K=1,NROWS
      do 200 L=1,4
      if(datarr(k,1).eq.0.)then
         count(k,L)=absen2(L+2)
      else
         count(k,L)=count(k,L)/datarr(k,1)
      endif
 200  CONTINUE
C
C......Write out
      do 290 L=1,4
      CALL OTDATA(IODISK,L+2,NOREC2+1,NROWS,count(1,L),
     &   NOFLD2,NOREC2+NROWS)
 290  continue
 300  CONTINUE
C
C......Write out vars 1 and 2
 305  CONTINUE
      DO 310 K=1,NROWS
 310  BIN(K)=X
      CALL OTDATA(IODISK,1,NOREC2+1,NROWS,BIN,NOFLD2,NOREC2+NROWS)
C
C......Y variable
      DO 320 K=1,NROWS
 320  BIN(K)=YMIN+(K-1)*YSTEP
      CALL OTDATA(IODISK,2,NOREC2+1,NROWS,BIN,NOFLD2,NOREC2+NROWS)
C
C......update NOREC2
      NOREC2=NOREC2+NROWS
C
C......finished with current X column
C......reset X
      X=X+XSTEP
C
C......loop
      GOTO 50
C
C......wrap up
  400 DO 19 I=1,NOFLD2
   19 CALL UPRLWR(IODISK,I,1,NOREC2,ALRLIM(I),UPRLIM(I),ABSEN2(I),
     &NOFLD2,NOREC2)
C
      CALL PFINIS(IODISK,PROG,
     & MAGIC,NOFLD2,NOREC2,NROWS,NPLANE,ICENT,IYMD,IHMS,
     & FLDNA2,FLDUN2,ALRLIM,UPRLIM,ABSEN2,
     & ALAT,ALONG,DEPTHI,DEPTHW,OPWRIT,RAWDAT,PIPEFL,ARCHIV,VERS,
     &
DATNAM,PREFIL,POSTFL,PLATYP,PLTNUM,RECINT,PLTNAM,INSTMT,COMENT)
C
      STOP
      END
```

Execs developed for OPC processing

OPCEXEC0

```
#################################################################
###########
#
# opcexec0
#
# Description:
# This is the first exec in a series to processes OPC data at sea
# opcexec0 reads in the data from the PC binary file and converts it
# to Pstar format.  It sets up the dataname and other header details.
#
# Files produced:
# o4$CRUISE$num Pstar format of data; only time and count because
# many data cycles
#  o4$CRUISE$num.arch   archive copy of o4psl$num.
#
#  Main processing steps:
#  STEP_01      Read in data using datapup
#  STEP_02      Archive
#
#  History:
#  Version Date     Author   Description
#  01               RTP      Original version.
#  NEXT    ??/??/?? ???      Please make a note of your changes here
#                            - using as many lines as necessary.  If
#                            the  changes are  substantial perhaps a
#                            new exec might be better?
#
#################################################################

########## Check these variables ##########################
#  These variables should be checked when setting up this  #
#  exec for the first time at sea.                         #

# Well, its not a variable yet, but the cruise "$CRUISE" (Polarstern)
#  needs to be generalised.

########## Initialisation ################################

#  Check directories
#  This exec looks at P_OPC for a directory to run from and...
   if ($?P_OPC) then
     echo " "
     echo " Changing directory to P_OPC: $P_OPC"
     cd $P_OPC
   endif

#########################################################

########## Get information from the user ##################

   echo "> This exec will require the following information:"
   echo ">    output OPC file number "
   echo ">    OPC input file name like r01.d00"
   echo ">    which clock? Polarstern or internal"
```

```
   echo ">     start date and time"
   echo -n "> Continue (y/n)? "
   set ans = $<
   if ($ans != "y") exit

   echo -n "> enter OPC output file number (nnn): "
   set num = $<
   if ( -e ./o4$CRUISE$num ) then
      echo -n "./o4$CRUISE$num exists.  Overwrite? (y/n)"
      set ans = $<
      if ( $ans != "y" ) exit(1)
   endif
   if ( -e ./oa$CRUISE$num ) then
      echo -n "./oa$CRUISE$num exists.  Overwrite? (y/n)"
      set ans = $<
      if ( $ans != "y" ) exit(1)
   endif


#  First run the program just to get the start time and check for
#  right file
   echo -n "> enter OPC input file name like r01.d00: "
   set pc_fil = $<
    echo $pc_fil
#   set echo
   /bin/rm -f opcexec0.talk
   touch opcexec0.talk


   cat << ! | sed 's/^   //' | popcin >> opcexec0.talk
   $pc_fil
   n
!
   awk 'NR>5&&NR<8' < opcexec0.talk
      echo -n "Is this the right file? (y/n)"
      set ans = $<
      if ( $ans != "y" ) exit(1)
#
# Rename  pc input file - this is a special for run8,
# named r07 on the pc files in error.
#
   set run = `echo $pc_fil | awk '{printf(substr($1,3,1))}'`
   if ( $run == 7)then
     set ext = $pc_fil:e
     /bin/mv -i opa13$pc_fil opa13r08.$ext
     set pc_fil = r08.$ext
     echo "file name has been changed to $pc_fil"
   endif
#  set the pc input file to read only mode
   chmod 444 opa13$pc_fil
   echo "opc input file opa13$pc_fil set to Read Only"

#  Now get rest of input
   echo -n "> Which clock? 0 = internal, 1 = Polarstern Wempe: "
   set clock = $<
#  ask user for start time
   echo -n "> enter start date (YYMMDD): "
   set startd = $<
```

```
   echo -n "> enter start time (HHMMSS): "
   set startt  = $<

##############################################################

########## Main processing steps ##########################

##############################################################
#  STEP_01 - Read in data
#

   echo "running popcin"
   cat << ! | sed 's/^   //' | popcin >> opcexec0.talk
   $pc_fil
   Y
   $clock
   o4$CRUISE$num
   o4$CRUISE$num
   $startd $startt
!
   if ($status != 0) then
     echo "problem running popcin - see opcexec0.talk"
     exit
   else
     tail -22 opcexec0.talk
   endif

##############################################################
#  STEP_02 - Read in data
#

   echo "running popain"
   cat << ! | sed 's/^   //' | popain >> opcexec0.talk
   $pc_fil
   Y
   $clock
   oa$CRUISE$num
   oa$CRUISE$num
   $startd $startt
!
   if ($status != 0) then
     echo "problem running popain - see opcexec0.talk"
     exit
   else
     tail -22 opcexec0.talk
   endif

########## Archiving ######################################

#  Save a copy for archiving
   arch_cp o4$CRUISE$num
   arch_cp oa$CRUISE$num

########## Keep directories tidy ##########################

#  /bin/rm -f opcexec0.talk

########## The End ########################################
```

```
echo " "
echo "> Files created:   o4$CRUISE$num "
echo ">                  oa$CRUISE$num "
echo "> End of opcexec0 for file $num from $pc_fil"
echo " "
the_end:
exit
```

OPCEXEC1

```
#################################################################
#
#  opcexec1
#
#  Description:
#  This exec is number 2 in a series of OPC processing execs.
#  It uses the output from opcexec0.
#  It merges the full rate data with SeaSoar and navigation data.
#  Before running this step, the input file must be truncated so
#  that its times lie within those of the matching SeaSoar 12
#  hour file for PMERGE
#
#  Processing steps:
#  STEP_01  PCALIB, subtract 150sec from time to match SeaSoar file
#  STEP_02  PMERGE, add press and distrun from SeaSoar file.
#  STEP_03  PGRIDO, grid to match relevant GR SeaSoar file.
#  STEP_04  PMERGE, merge lat and long from navigation file
#  STEP_05  archive, as earlier files are very large
#
#  History:
#  Version Date     Author   Description
#  00      13/12/95 RTP      Original version.
#  NEXT    ??/??/?? ???      Please make a note of your changes here
#                            - using as many lines as necessary.  If
#                            the  changes are  substantial perhaps a
#                            new exec might be better?
#
#################################################################

#################################################################

########## Initialisation ##################################

#  Check directories
#  This exec looks at P_OPC for a directory to run from and...
#  set echo
   if ($?P_OPC) then
     echo " "
     echo " Changing directory to $P_OPC"
     cd $P_OPC
   endif

#  set up variables and files

   /bin/rm -f opcexec1.talk
   touch opcexec1.talk

########## Get information from the user ##################

   echo "> This exec will require the following information:"
#  This version assumes opc and ss file numbers are in step
   echo ">    opc/seasoar file number "
   echo ">    Distrun limits for gridding"
   echo -n "> Continue (y/n)? "
   set ans = $<
   if ($ans != "y") exit
```

```
   echo -n "> Enter seasoar/opc file number : "
   set num = $<

   if (! -e op$CRUISE$num) then
     echo " File op$CRUISE$num does not exist"
     exit
   endif

   if (! -e /users/pstar/data/seasoar/ss$CRUISE$num) then
     echo " File ss$CRUISE$num does not exist"
     exit
   endif

   echo " "

   if (! -e /users/pstar/data/underway/nav${CRUISE}02) then
     echo " File nav${CRUISE}02 does not exist"
     exit
   endif

   echo -n "> Enter start distrun for gridding: "
   set startd = $<
   echo -n "> Enter stop distrun for gridding: "
   set stopd = $<
##############################################################
#  STEP_01  subtract 150 secs from time
#
   echo "> Running pca1ib - "
   cat << ! | sed 's/^   //' | pcalib >> opcexec1.talk
   op$CRUISE$num
   y
   time,-150/
   /
!
   if ($status != 0) then
     echo "> ***problem running pcalib. See opcexec1.talk***"
     exit
   endif
print_datnam op${CRUISE}$num new

##############################################################
#  STEP_02  Merge with matching 12 hour SeaSoar fi1e
#

   echo "> Running pmerge - "
   cat << ! | sed 's/^   //' | pmerge  >> opcexec1.talk
   op$CRUISE$num
   op$CRUISE$num.nav
   /
   /users/pstar/data/seasoar/ss$CRUISE$num
   time,press,distrun/
!
   if ($status != 0) then
     echo "> ***problem running pmerge. See opcexec1.ta1k***"
     exit
   endif
print_datnam op${CRUISE}$num.nav new
```

```
##################################################################
#  step_03  grid the file to the same specs as gr file
#
#
   echo "> Running pgrido - "
   cat << ! | sed 's/^   //' | pgrido  >> opcexec1.talk
   op$CRUISE$num.nav
   og$CRUISE$num
   distrun,press,count/
   5,453,8
   ${startd},${stopd},6,3
!
    if ($status != 0) then
      echo "> ***problem running pgrido. See opcexec1.talk***"
      exit
    endif
 print_datnam og$CRUISE$num new

##################################################################
#  step_04 second pmerge to add lat and long
#
   echo "> Running pmerge - "
   cat << ! | sed 's/^   //' | pmerge  >> opcexec1.talk
   og$CRUISE$num
   og$CRUISE$num.tmp
   /
   distrun
   /users/pstar/data/underway/nav${CRUISE}02
   distrun,gps_lat,gps_lon/
!
    if ($status != 0) then
      echo "> ***problem running pmerge. See opcexec1.talk***"
      exit
    endif
 print_datnam og$CRUISE$num.tmp new

#############################################################
#  now check out the atentuation file

   if (! -e at$CRUISE$num) then
     echo " File at$CRUISE$num does not exist"
     exit
   endif

#############################################################
#  STEP_01a  subtract 150 secs from time
#
   echo "> Running pcalib - "
   cat << ! | sed 's/^   //' | pcalib >> opcexec1.talk
   at$CRUISE$num
   y
   time,-150/
   /
!
   if ($status != 0) then
     echo "> ***problem running pcalib. See opcexec1.talk***"
     exit
```

```
   endif
print_datnam at${CRUISE}$num new

##############################################################
#  STEP_02a  Merge with matching 12 hour SeaSoar file
#

   echo "> Running pmerge - "
   cat << ! | sed 's/^   //' | pmerge  >> opcexec1.talk
   at$CRUISE$num
   at$CRUISE$num.nav
   /
   /users/pstar/data/seasoar/ss$CRUISE$num
   time,press,distrun/
!
   if ($status != 0) then
     echo "> ***problem running pmerge. See opcexec1.talk***"
     exit
   endif
print_datnam at${CRUISE}$num.nav new

#################################################################
#  step_03a  grid the file to the same specs as gr file
#
#
   echo "> Running pgrids - "
   cat << ! | sed 's/^   //' | pgrids  >> opcexec1.talk
   at$CRUISE$num.nav
   at$CRUISE$num.tmp
   distrun,press,atten/
   5,453,8
   ${startd},${stopd},6,3
!
    if ($status != 0) then
      echo "> ***problem running pgrids. See opcexec1.talk***"
      exit
    endif
 print_datnam at$CRUISE$num.tmp new

#################################################################
# step 4 PJOIN the count and atten grideed files. This will only
# work if care has been taken to get the start and end times of
# the appended atten and op files very similar, so that gridded
# files have the same number of columns in distrun
#
#
   echo "> Running pjoin - "
   cat << ! | sed 's/^   //' | pjoin  >> opcexec1.talk
   og$CRUISE$num.tmp
   og$CRUISE$num
   /
   at$CRUISE$num.tmp
   3/
!
    if ($status != 0) then
      echo "> ***problem running pjoin. See opcexec1.talk***"
      exit
    endif
```

```
 print_datnam og$CRUISE$num new

########## End of main processing steps ####################

########## Archiving #####################################

   arch_cp og$CRUISE$num
   arch_cp op$CRUISE$num.nav

########## Keep directories tidy ###########################

#  /bin/rm -f opcexec1.talk

########## The End ##########################################

   echo " "
   echo "> end of opcexec1"
   echo "> Next run opcexec2 to plot data"

   the_end:
   exit
```

Table 1

SeaSoar deployments on AntXIII/2

Run	start	time (Z)	day/date	latitude	longitude	distance run (km)
001	start	1347	339/5xii95	39°05.9'S	14°43.9'E	679
	end	1834	339/5xii95	39°38.8'S	14°15.0'E	753
	duration	4h47m	mean speed 8.3 kt			74 km
Reason for recovery						End of trial
002	start	0931	340/6xii95	41°46.7'S	12°46.4'E	1036
	end	0300	343/9xii95	50°13.5'S	5°46.0'E	2127
	duration	2d17h29m	mean speed 8.9 kt			1091 km
Reason for recovery						End of run - mooring position reached
003	start	1427	343/9xii95	50°21.3'S	5°31.3'E	2195
	end	2117	345/11xii95	57°19.5'S	2°06.3'W	3185
	duration	2d6h50m	mean speed 9.7 kt			990 km
Reason for recovery						End of run - ice reached
6.1-3	start	1845	356/22xii95	54°00.6'S	0°05.8'W	6954
	end	1848	359/25xii95	50°28.2'S	8°05.3'E	8235
	duration	3d0h3m	mean speed 9.7 kt			1281 km
Reason for recovery						End of run - station position
6.4-5	start	2240	359/25xii95	50°29.1'S	8°07.8'E	8248
	end	0930	361/27xii95	49°54.9'S	10°15.3'E	8844
	duration	1d10h50m	mean speed 9.2 kt			596 km
Reason for recovery						Cable problem - need to reterminate
6.6-7	start	1715	361/27xii95	49°28.2'S	10°31.4'E	8903
	end	0845	363/29xii95	49°27.8'S	11°24.6'E	9555
	duration	1d15h34m	mean speed 8.8 kt			652 km
Reason for recovery						End of survey
8.1	start	0900	001/1i96	49°41.5'S	11°19.8'E	10292
	end	1430	001/1i96	50°25.0'S	11°23.1'E	10376
	duration	5h30m	mean speed 8.2 kt			84 km
Reason for recovery						Vehicle not flying - kelp holdfast around wing
8.1-11	start	1500	001/1i96	50°27.8'S	11°23.3'E	10381
	end	0627	005/5i96	50°48.6'S	9°30.0'E	11768
	duration	3d15h27m	mean speed 8.5 kt			1387 km
Reason for recovery						End of survey

Totals

duration	15d4h30m	mean speed 9.0 kt	6155 km

Table 2

SeaSoar legs on AntXIII/2

Run	date	time(Z)	latitude	longitude	distrun (km)	course	
1	5xii95	1347	39°05.9'S	14°43.9'E	679	**211°**	
		1834	39°38.8'S	14°15.0'E	753	end	
2	6xii95	0931	**41°46.7'S**	12°46.4'E	1036	**211°**	
	9xii95	0300	50°13.5'S	5°46.0'E	2127	end	
3		1427	50°21.3'S	5°31.3'E	2195	**235°**	
	10xii95	1720	52°50.7'S	0°10.4'E	2663	**195°**	
	11xii95	2117	**57°19.5'S**	2°06.3'W	3185	end	
6.1	22xii95	1845	**54°00.6'S**	**0°5.8'W**	6954	**007°**	
	23xii95	0150	52°48.0'S	0°15.5'E	7093	053°	
	24xii95	0315	**50°14.6'S**	5°48.0'E	7570	090°	
6.2		0400	50°14.3'S	**6°0.5'E**	7584	**180°**	
		1505	**52°00.6'S**	6°00.8'E	7782	090°	
6.3		1940	51°59.6'S	**7°6.0'E**	7857	**000°**	
	25xii95	1005	**49°54.8'S**	7°06.1'E	8089	090°	
6.4		1415	49°54.7'S	**8°9.1'E**	8165	**180°**	
		1848	50°28.2'S	8°05.3'E	8235	end	st'n 10
		2240	50°29.1'S	8°07.8'E	8248	180°	restart
	26xii95	0855	**52°00.0'S**	8°09.4'E	8422	090°	
6.5		1300	51°59.9'S	**9°14.7'E**	8496	**000°**	
	27xii95	0345	**49°44.5'S**	9°15.5'E	8748	090°	
6.6		0755	49°45.2'S	10°17.5'E	8822	180°	
		0930	49°54.9'S	10°15.3'E	8844	end	cable
6.6a		1715	**49°28.2'S**	10°31.4'E	8903	**180°**	restart
	28xii95	1110	52°00.3'S	**10°17.7'E**	9197	090°	
6.7		1530	51°59.8'S	**11°23.1'E**	9273	**000°**	
	29xii95	0845	**49°27.8'S**	11°24.6'E	9555	end	

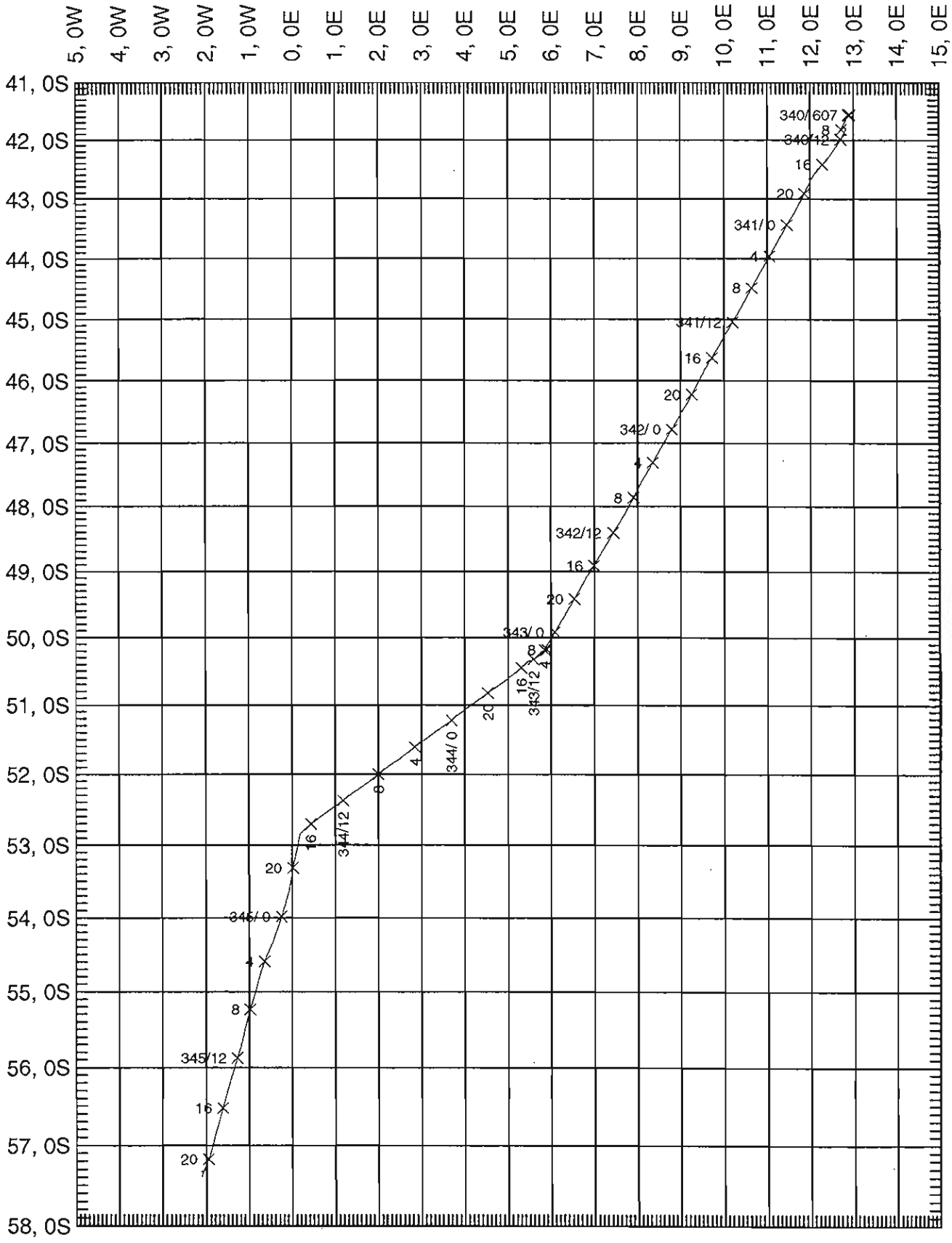
Table 2 (cont.)

SeaSoar legs on AntXIII/2 (cont.)

Run	date	time(Z)	latitude	longitude	distrun (km)	course	
8.1	1i96	0900	**49°41.5'S**	**11°23.1'E**	10292	**180°**	
		1430	50°25.0'S	11°23.1'E	10376	end:	kelp
		1500	50°27.8'S	11°23.3'E	10381	180°	restart
		1725	**50°48.0'S**	11°23.3'E	10418	270°	
8.2		1820	50°48.0'S	**11°12.2'E**	10430	**000°**	
	2i96	0140	**49°42.0'S**	11°12.2'E	10551	270°	
8.3		0230	49°42.0'S	**11°01.3'E**	10565	**180°**	
		1000	**50°48.0'S**	11°01.6'E	10686	270°	
8.4		1050	50°48.0'S	**10°50.4'E**	10698	**000°**	
		1810	**49°42.0'S**	10°50.4'E	10820	270°	
8.5		1900	49°42.0'S	**10°39.5'E**	10834	**180°**	
	3i96	0255	**50°48.0'S**	10°39.5'E	10955	270°	
8.6		0345	50°48.0'S	**10°28.6'E**	10968	**000°**	
		1115	**49°42.0'S**	10°28.6'E	11090	270°	
8.7		1205	49°42.0'S	**10°17.7'E**	11102	**180°**	
		1945	**50°48.0'S**	10°17.7'E	11223	270°	
8.8		2035	50°48.0'S	**10°06.8'E**	11235	**000°**	
	4i96	0400	**49°42.0'S**	10°06.8'E	11357	270°	
8.9		0455	49°42.0'S	**9°55.9'E**	11370	**180°**	
		1255	**50°48.0'S**	9°55.9'E	11492	270°	
8.10		1340	50°48.0'S	**9°45.0'E**	11504	**000°**	
		2100	**49°42.0'S**	9°45.0'E	11626	270°	
8.11		2150	49°42.0'S	**9°34.1'E**	11639	**180°**	
	5i96	0545	**50°48.0'S**	9°34.1'E	11761	270°	
		0627	50°48.6'S	9°30.0'E	11768	end	

Figures

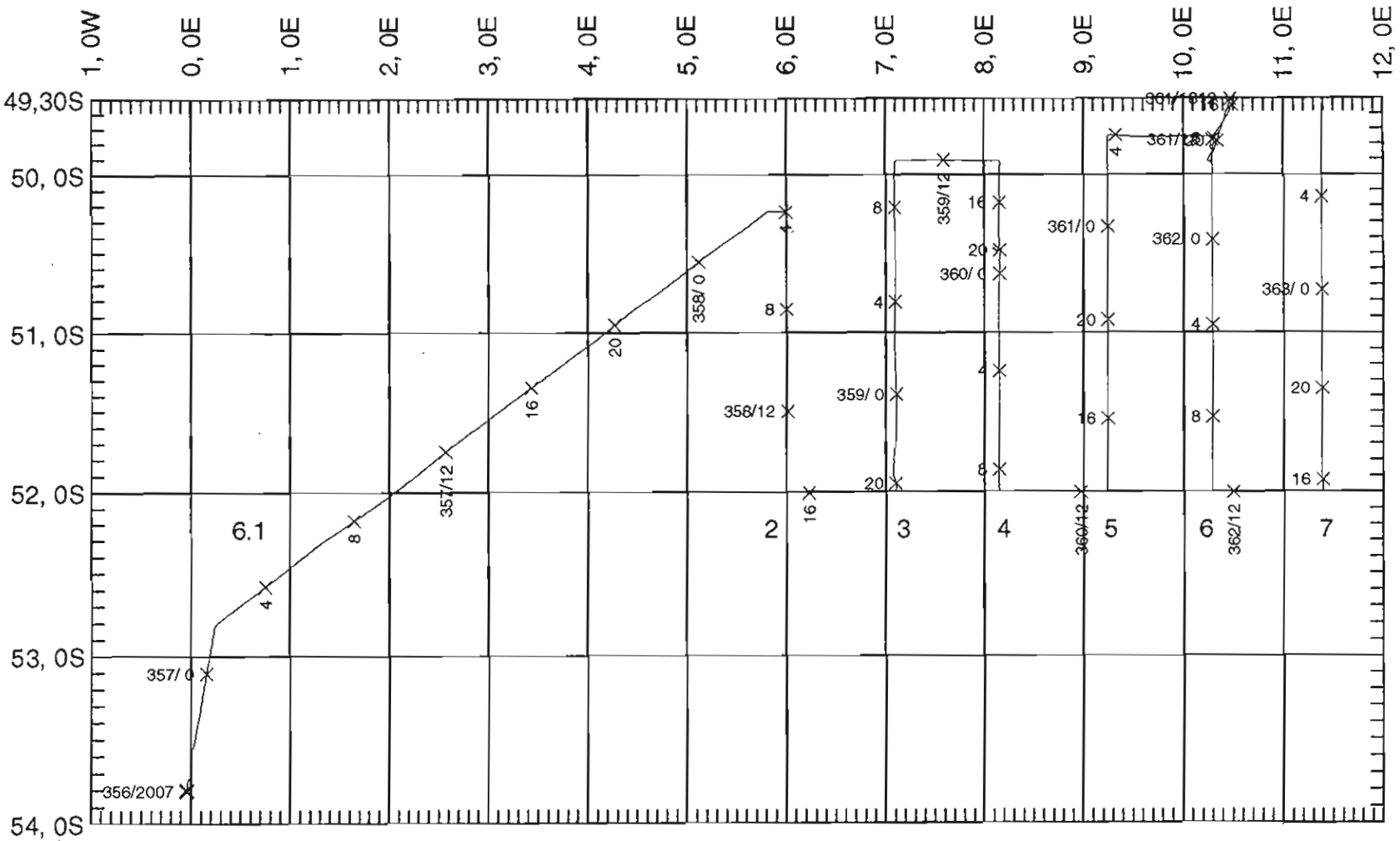
This page intentionally left blank



Mercator 1: 10000000 at 45.0 S

start: 340/ 600 stop: 346/ 0

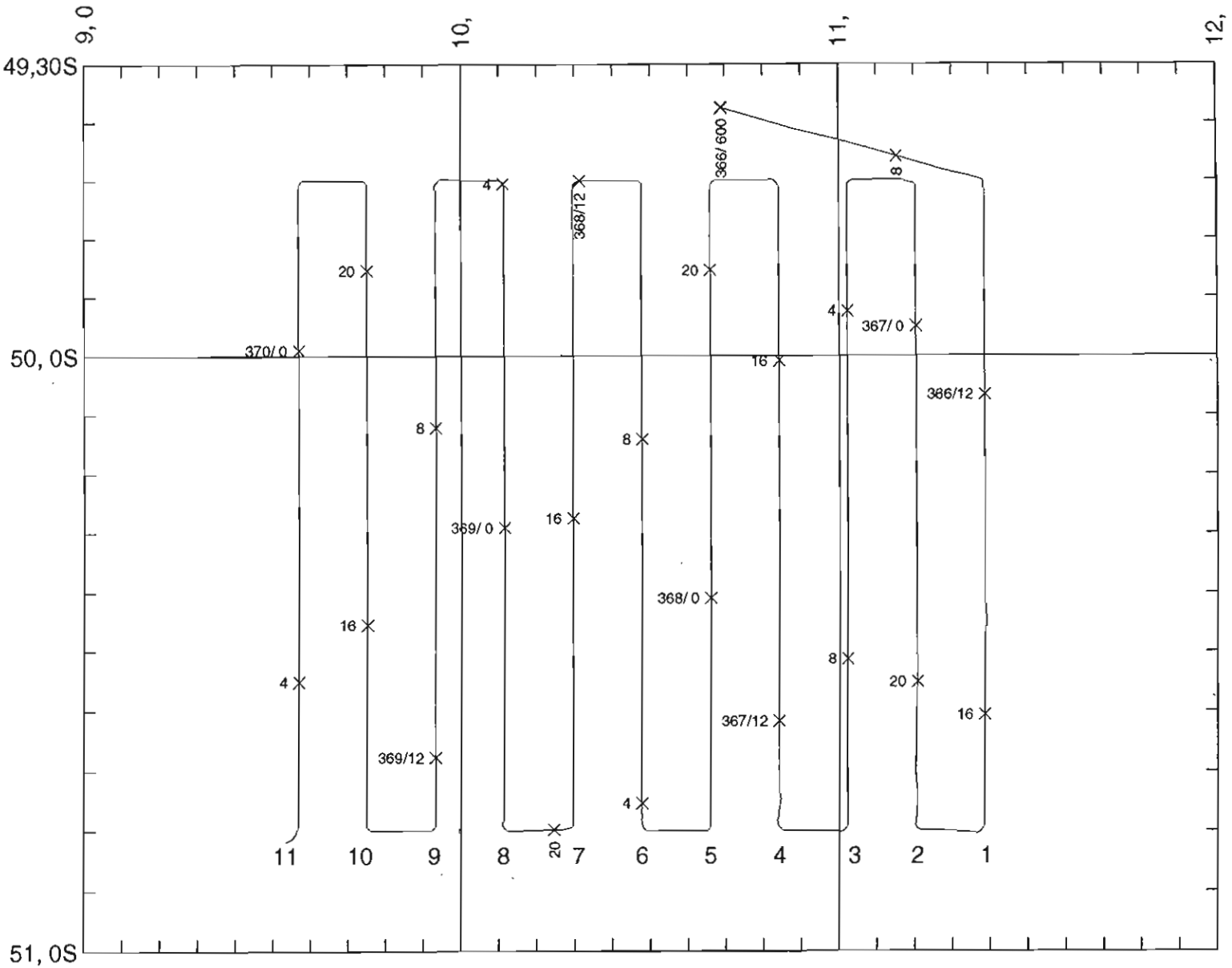
POLARSTERN ANTXIII2



Mercator 1: 5000000 at 45.0 S

start: 356/2000 stop: 363/ 800

POLARSTERN ANTXIII2

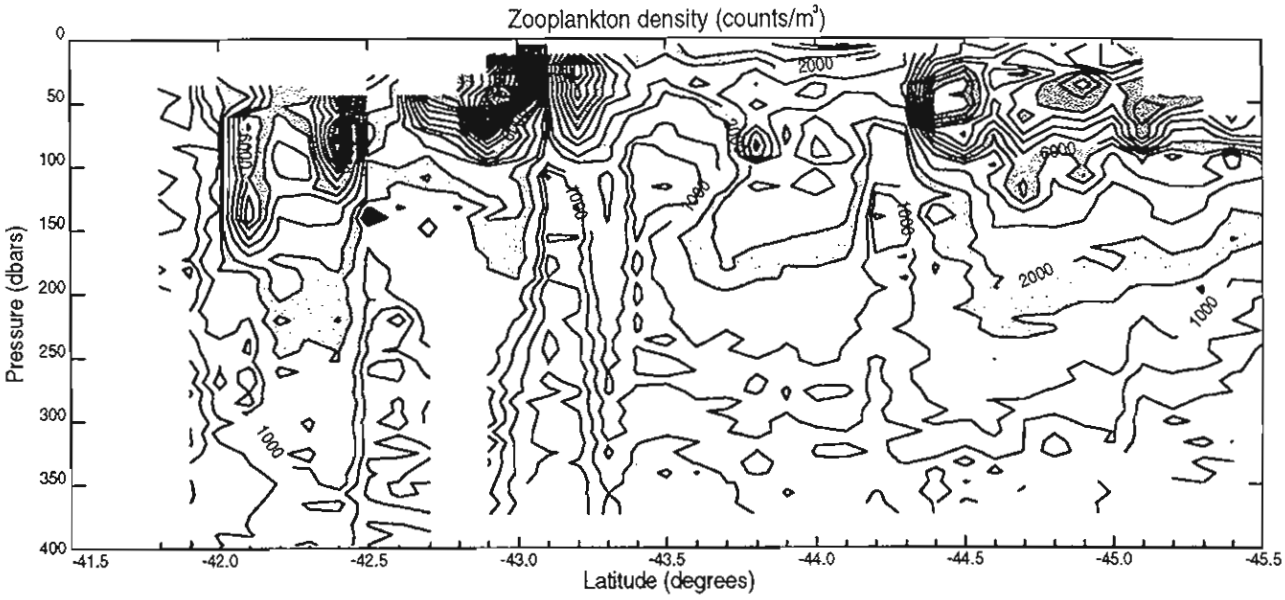


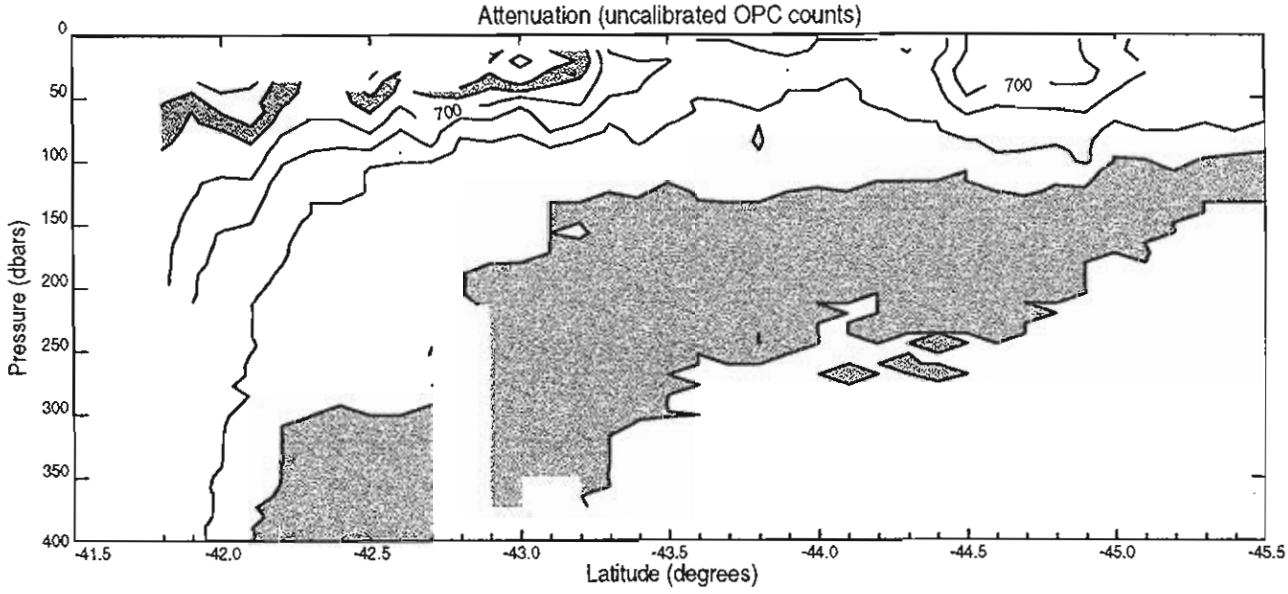
Mercator 1: 1200000 at 45.0 S

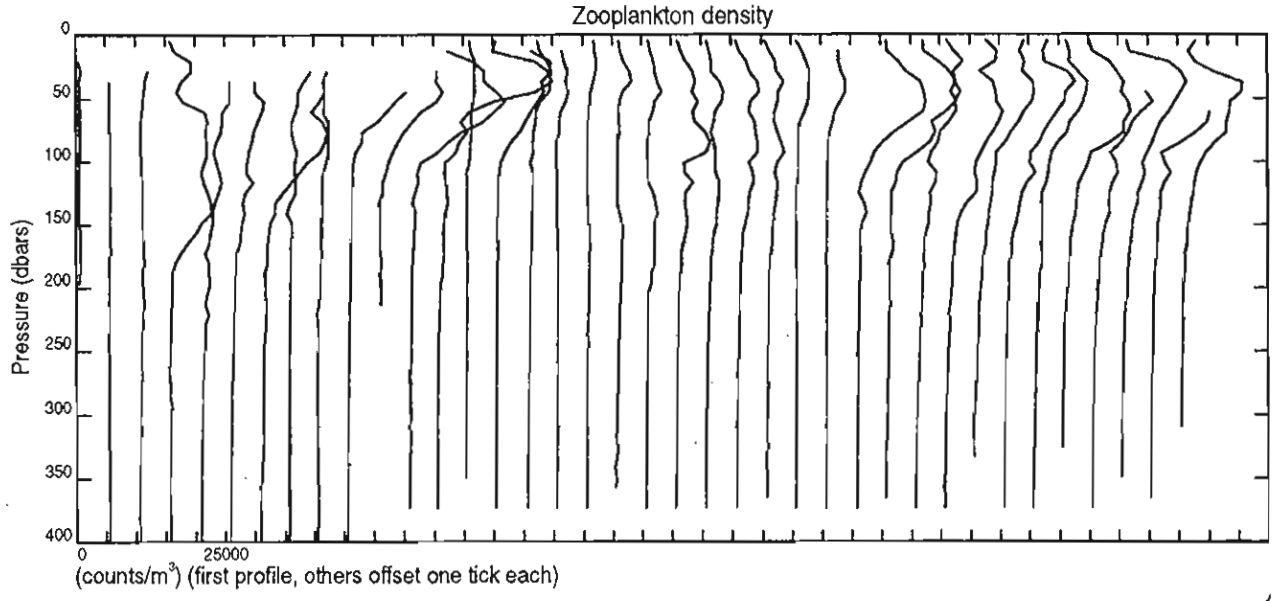
start: 366/ 600 stop: 370/ 600

POLARSTERN ANTXIII2

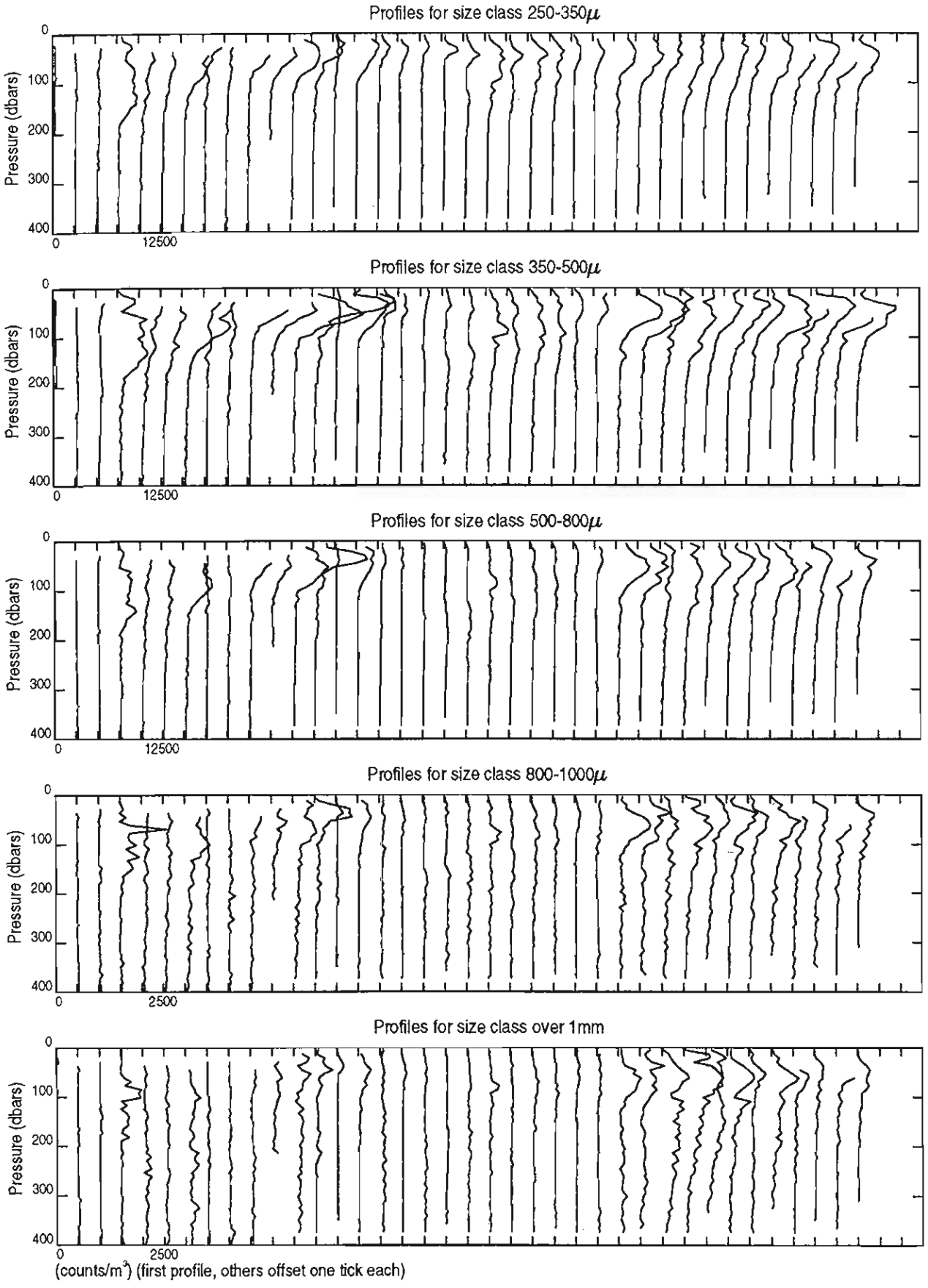
SeaSoar Run 2.1



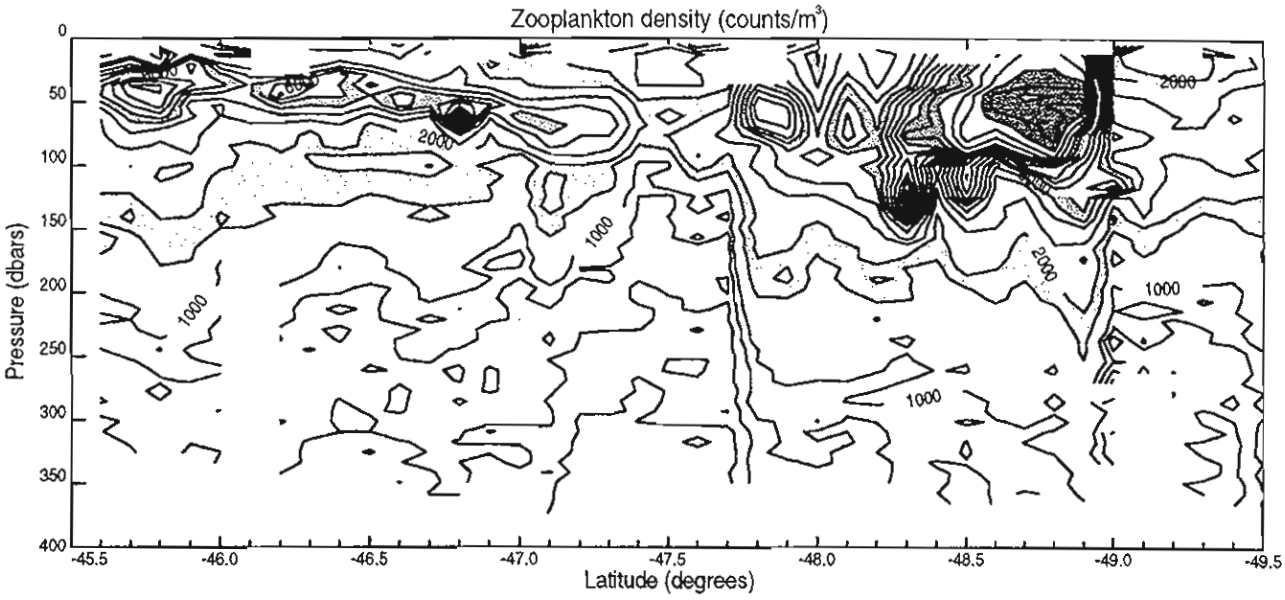


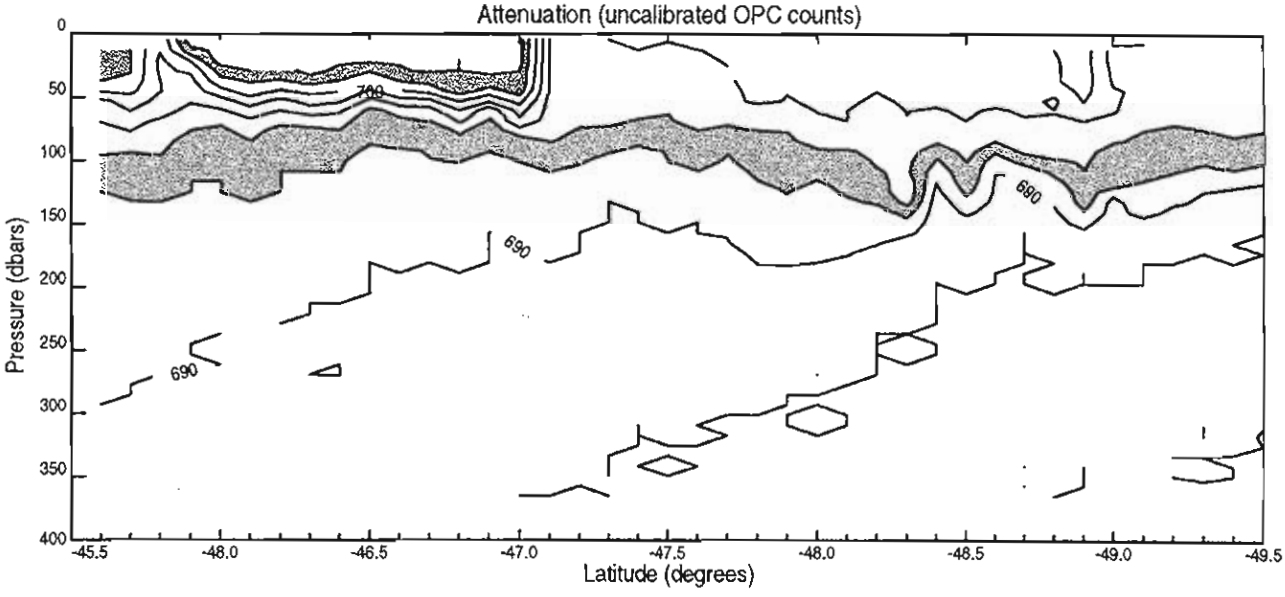


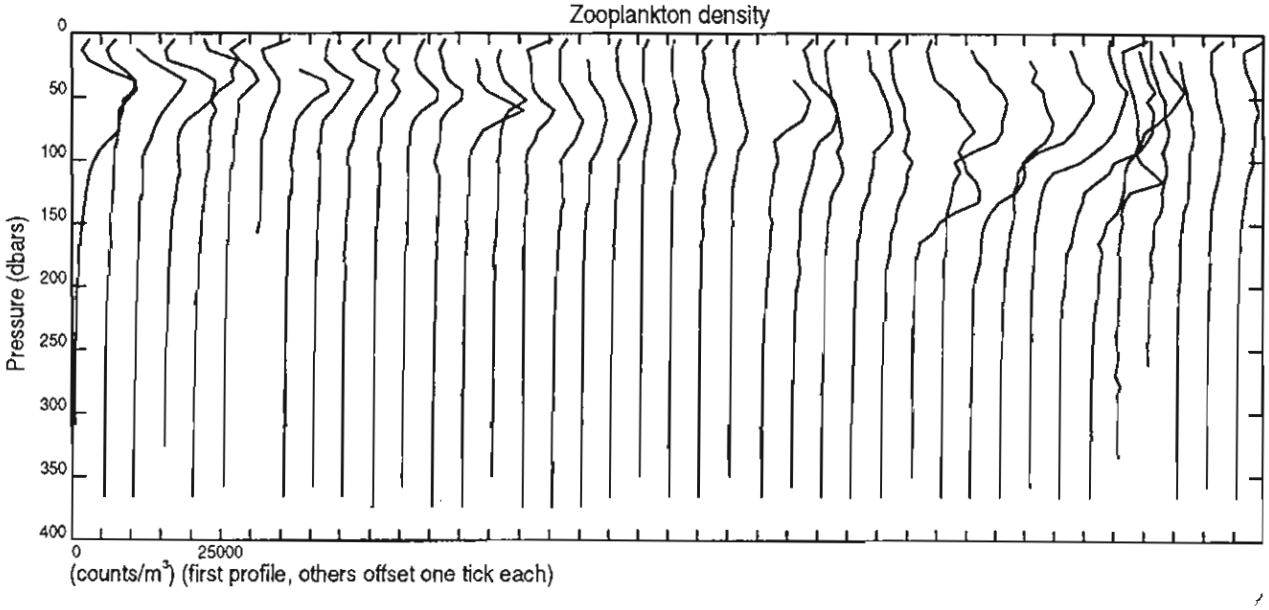
SeaSoar Run 2



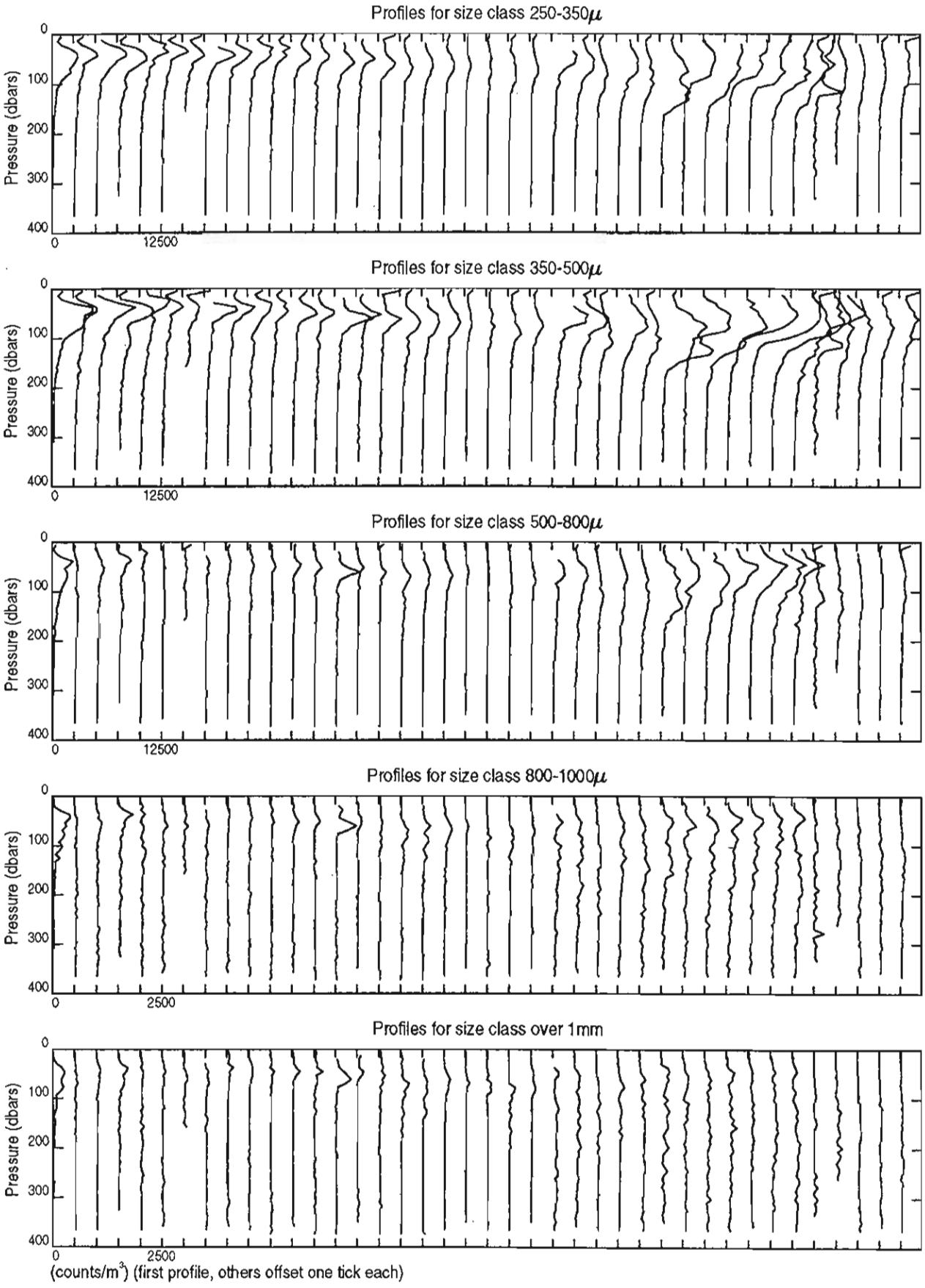
SeaSoar Run 2.2



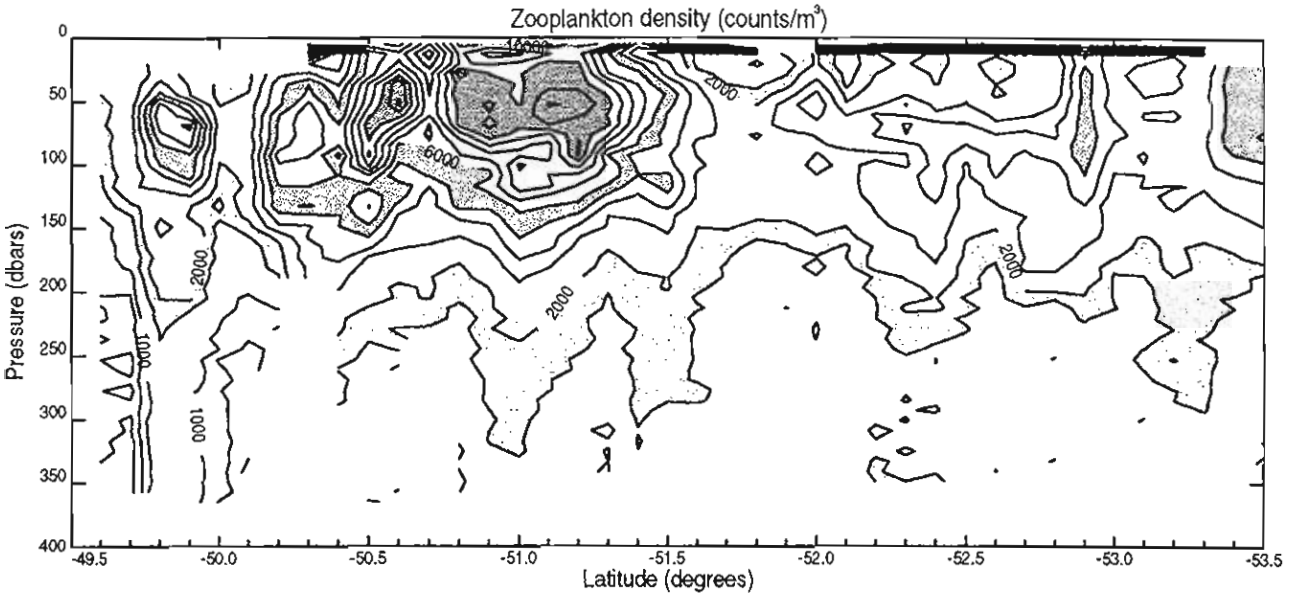


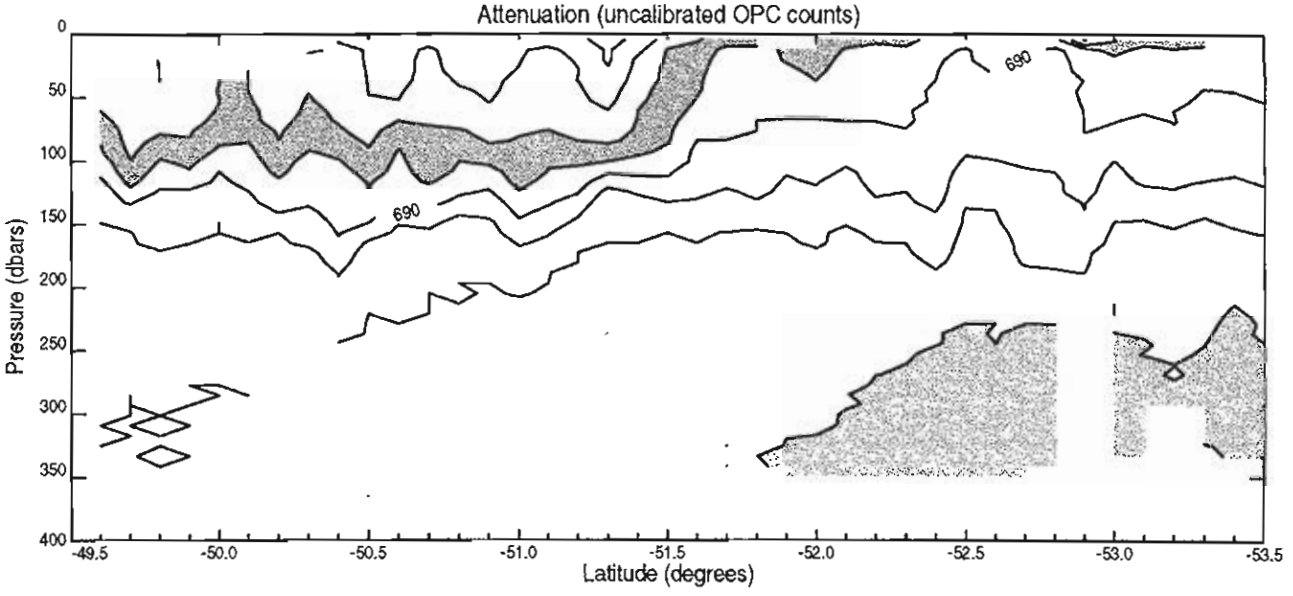


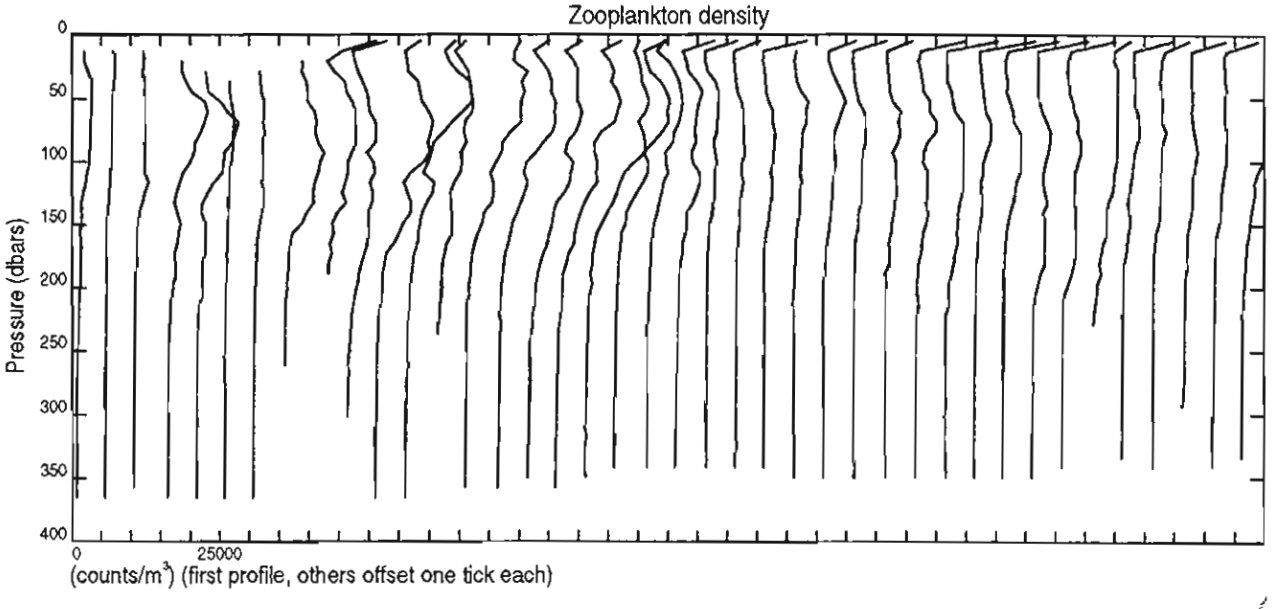
SeaSoar Run 2.2



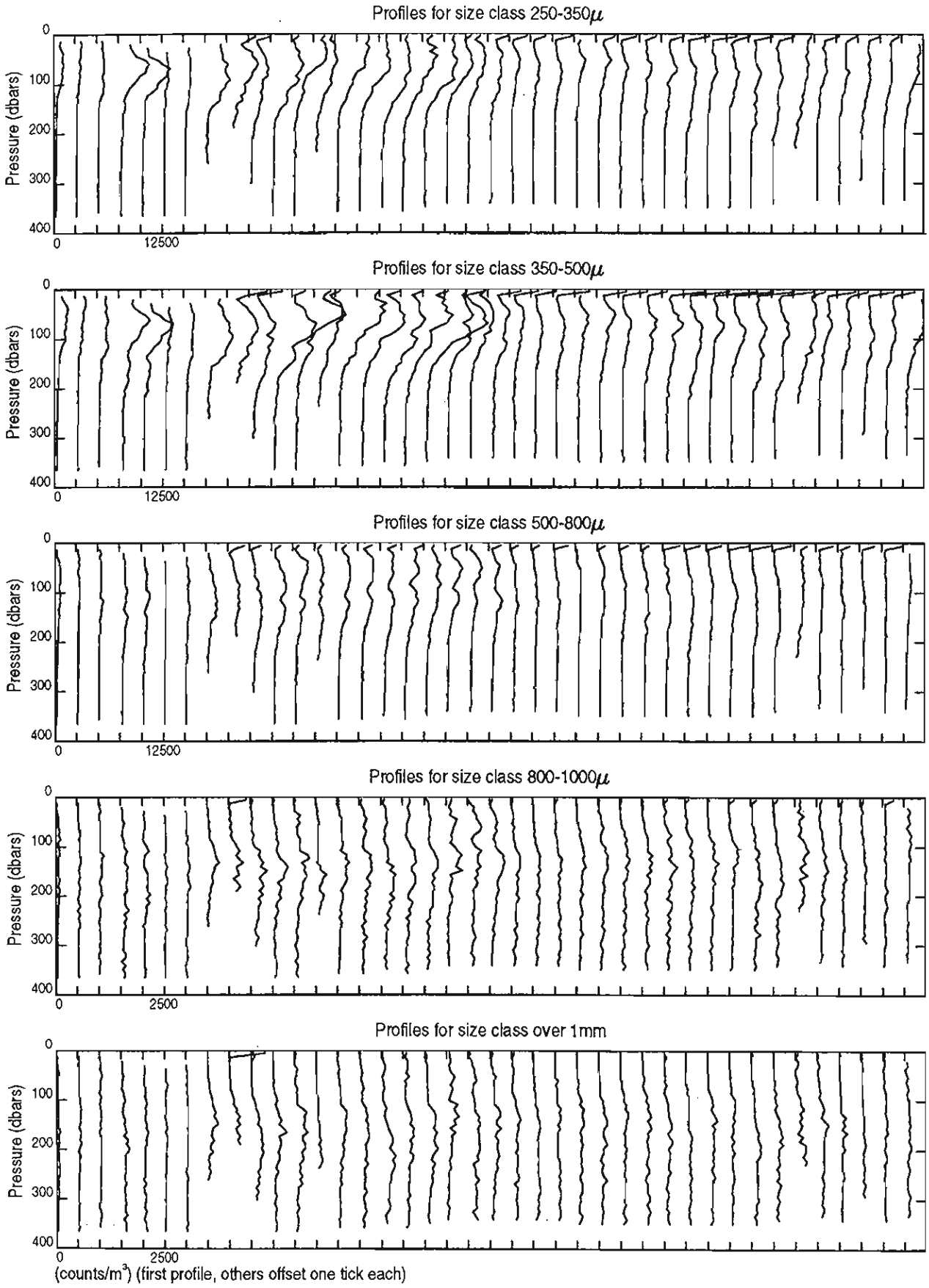
SeaSoar Runs 2.3 & 3



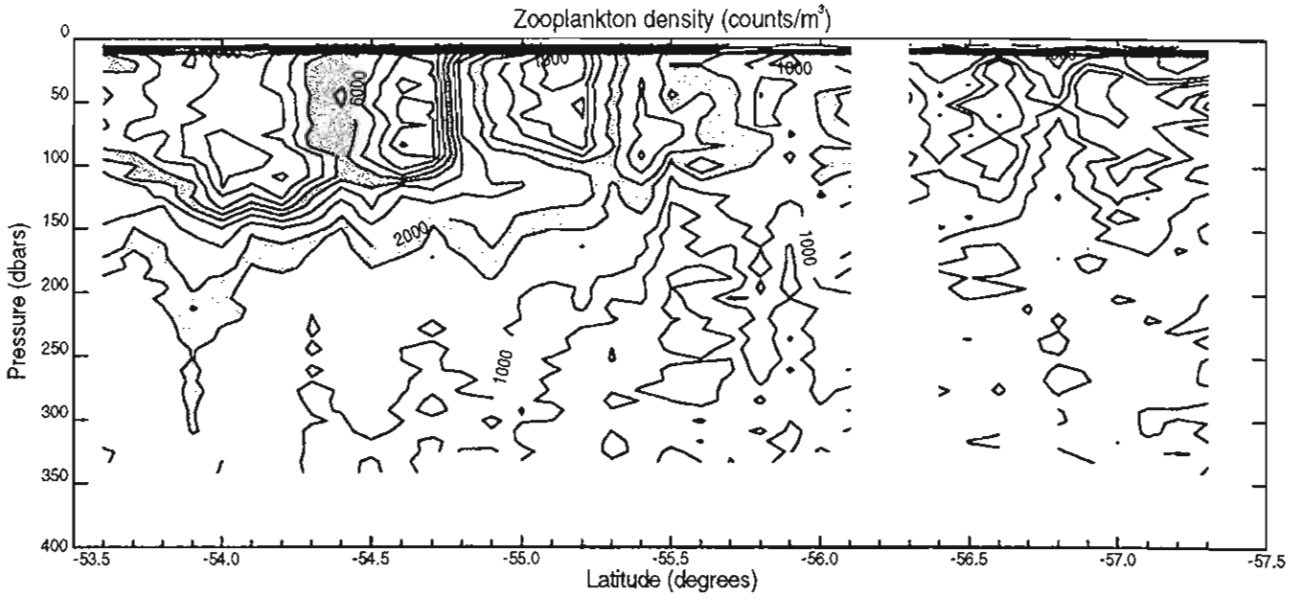


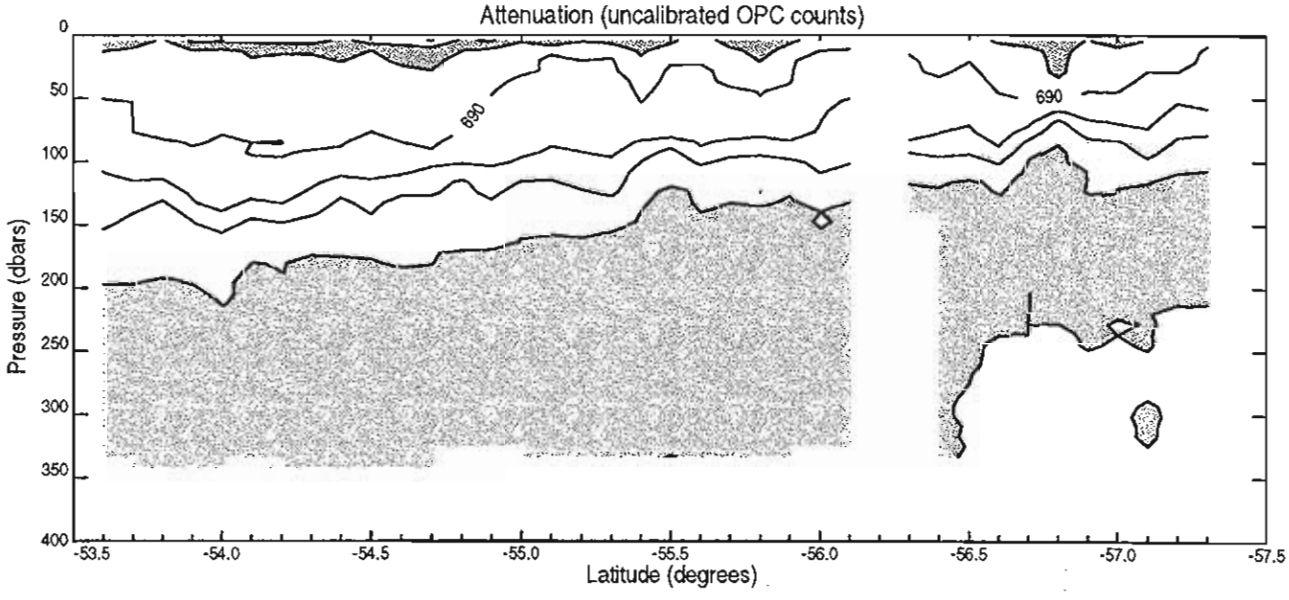


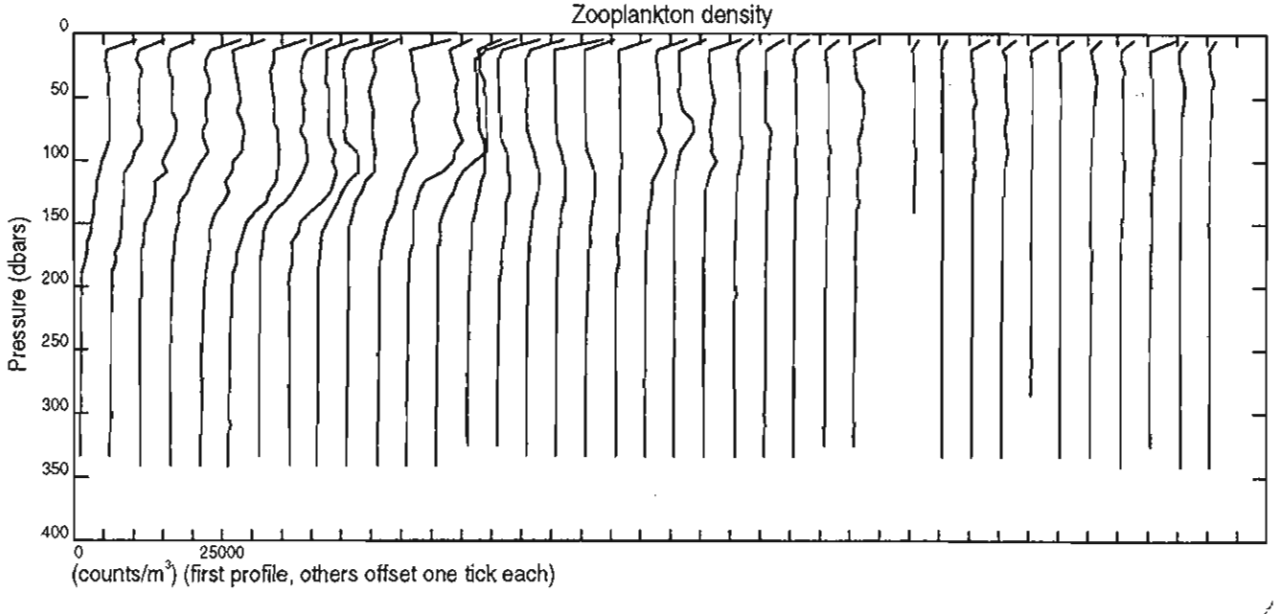
SeaSoar Runs 2.3 & 3



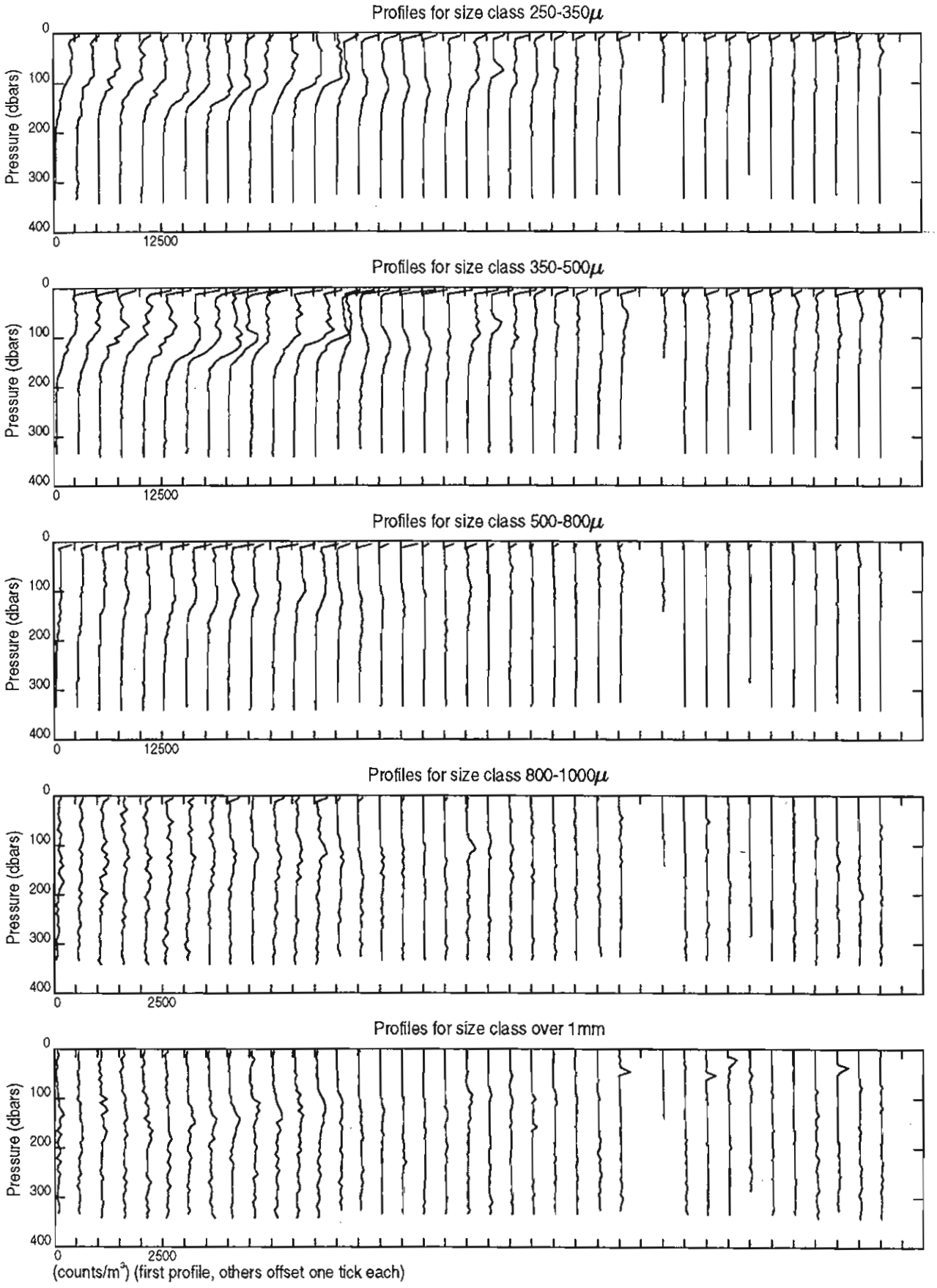
SeaSoar Run 3



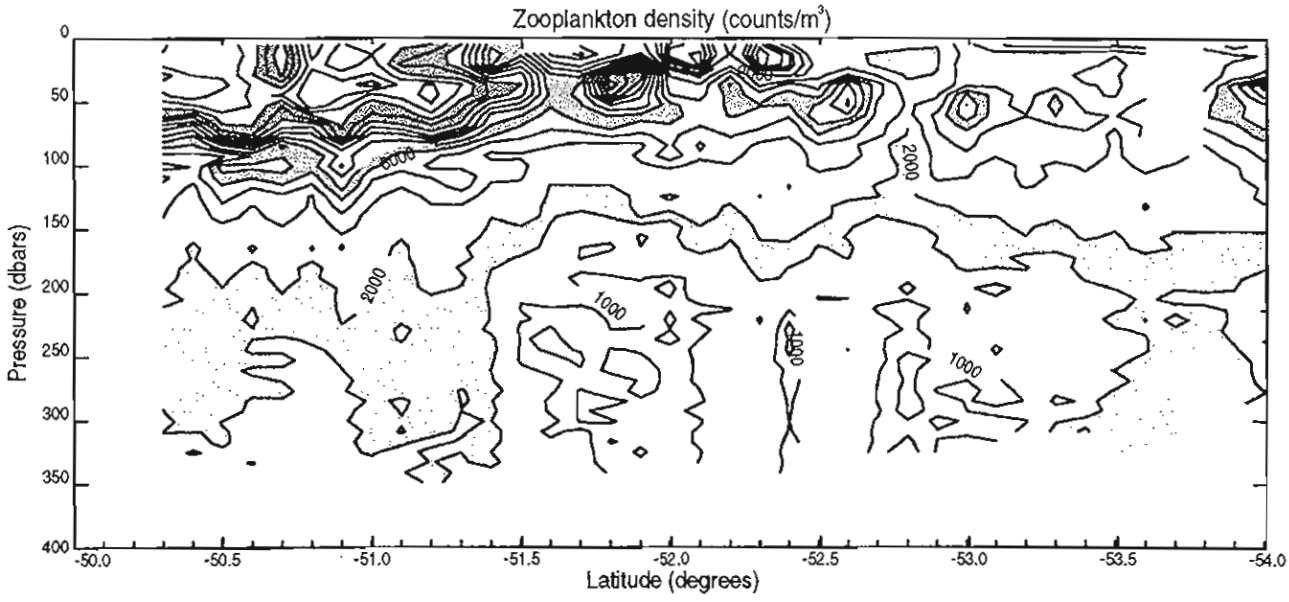


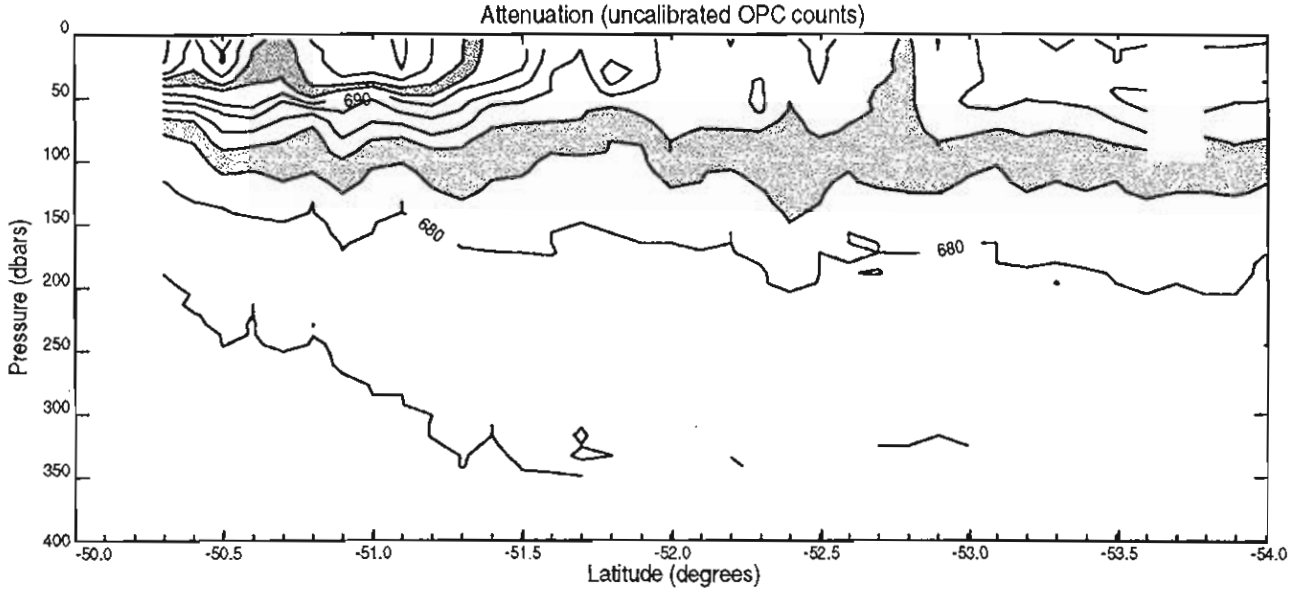


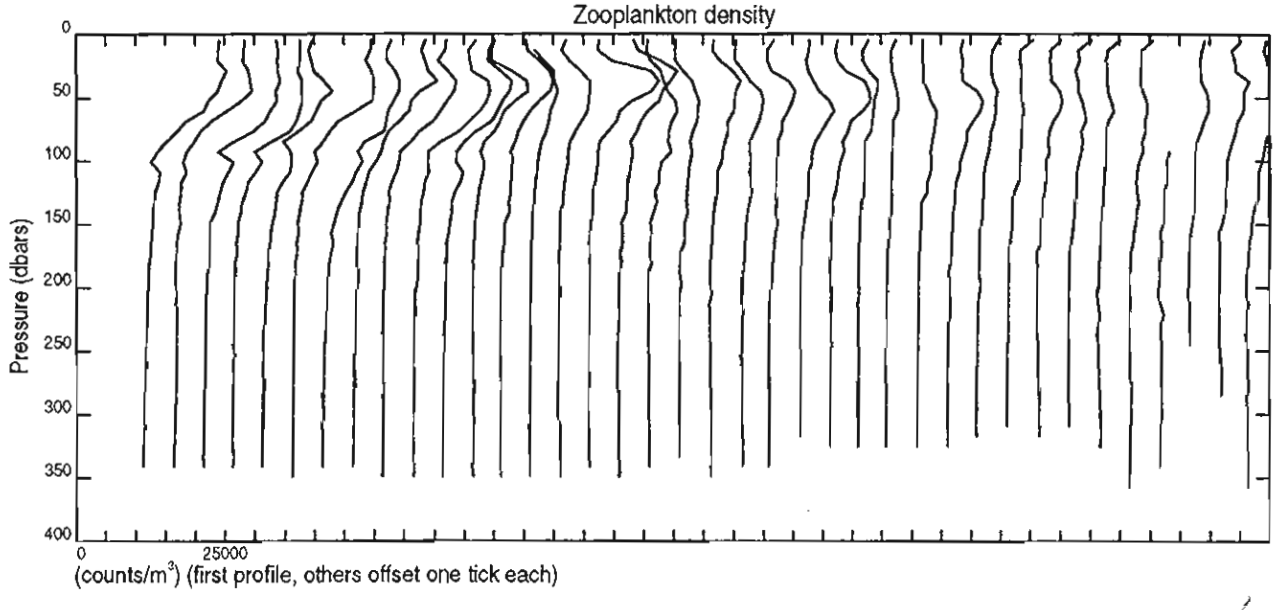
SeaSoar Run 3



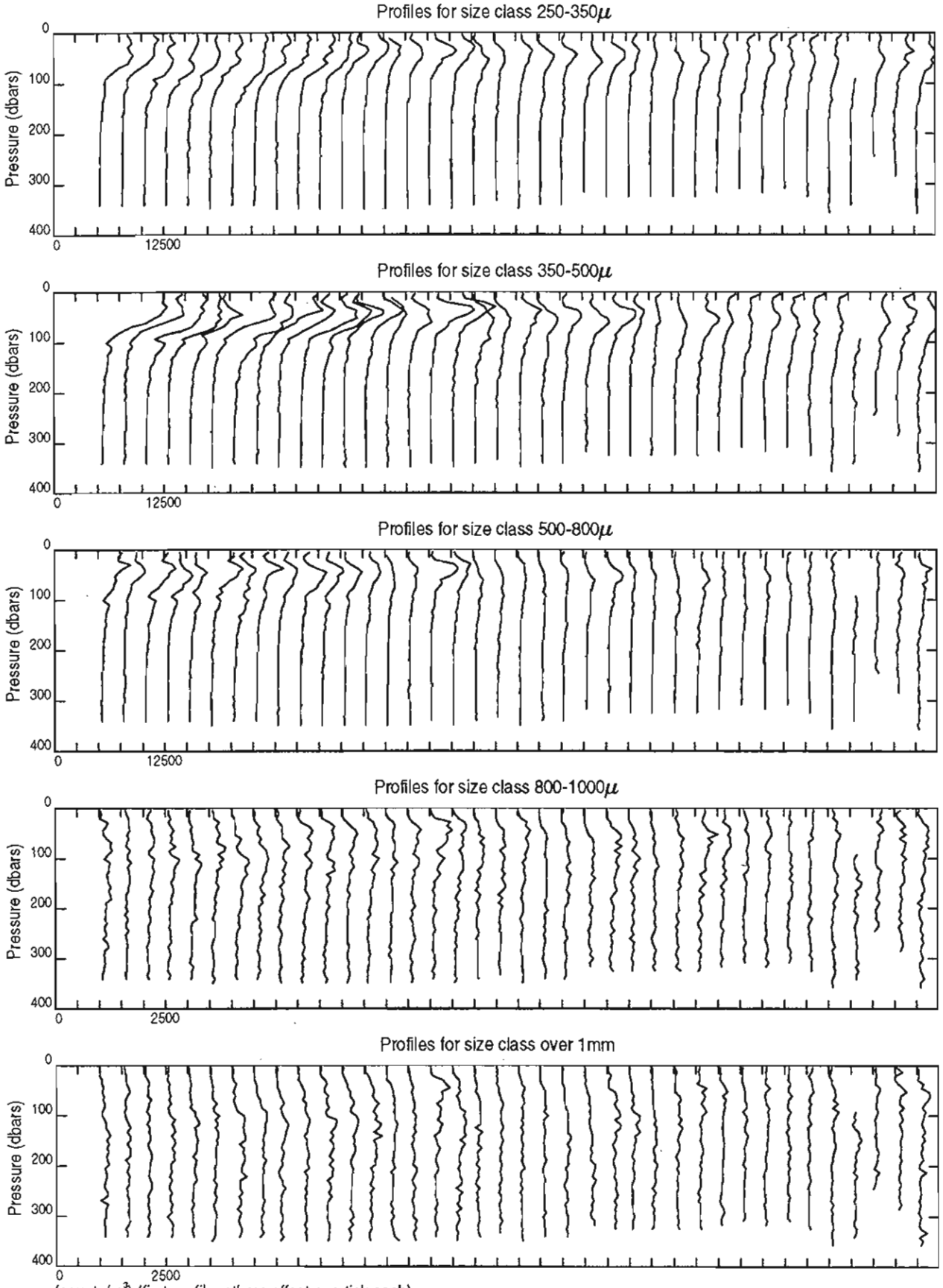
SeaSoar Run 6.1





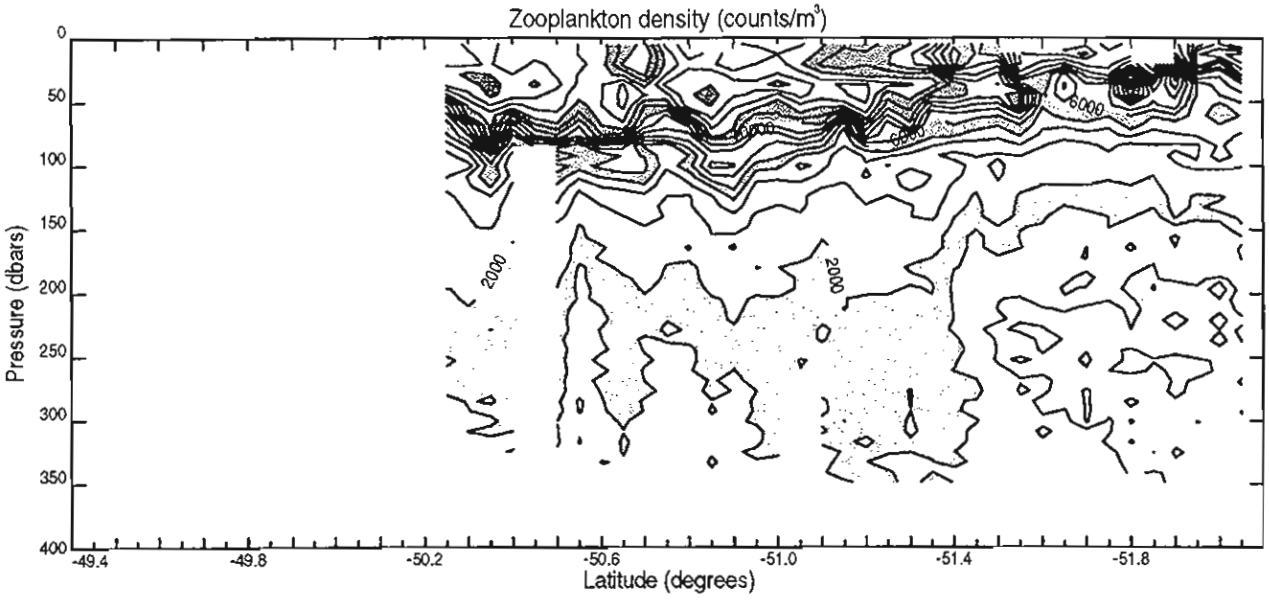


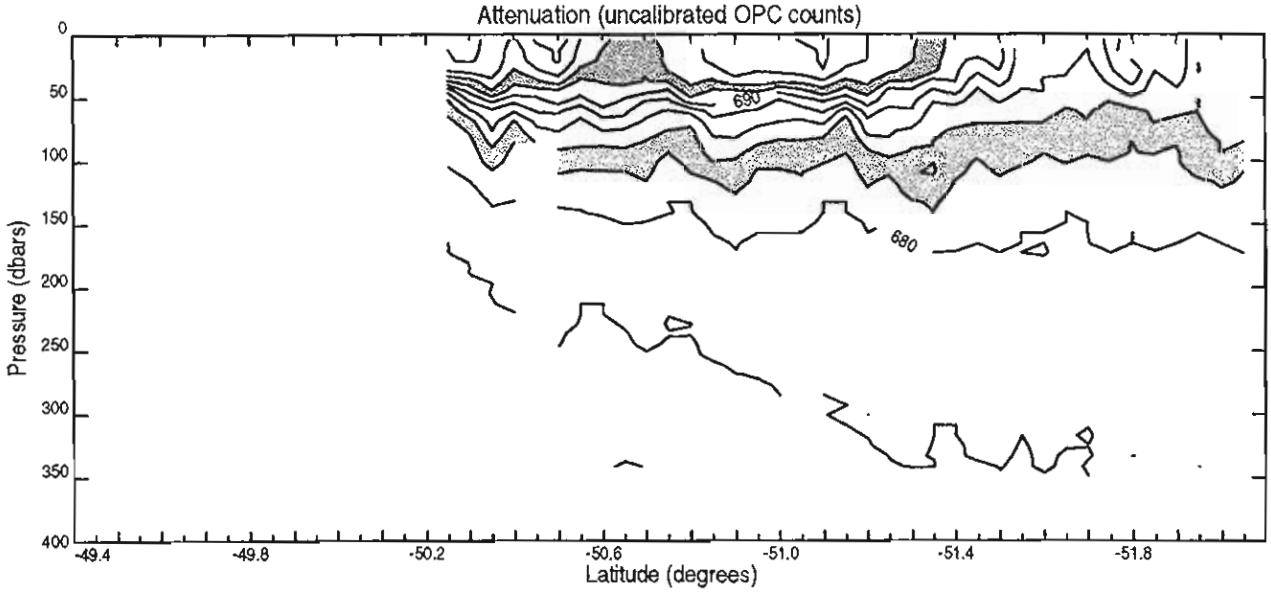
SeaSoar Run 6.1

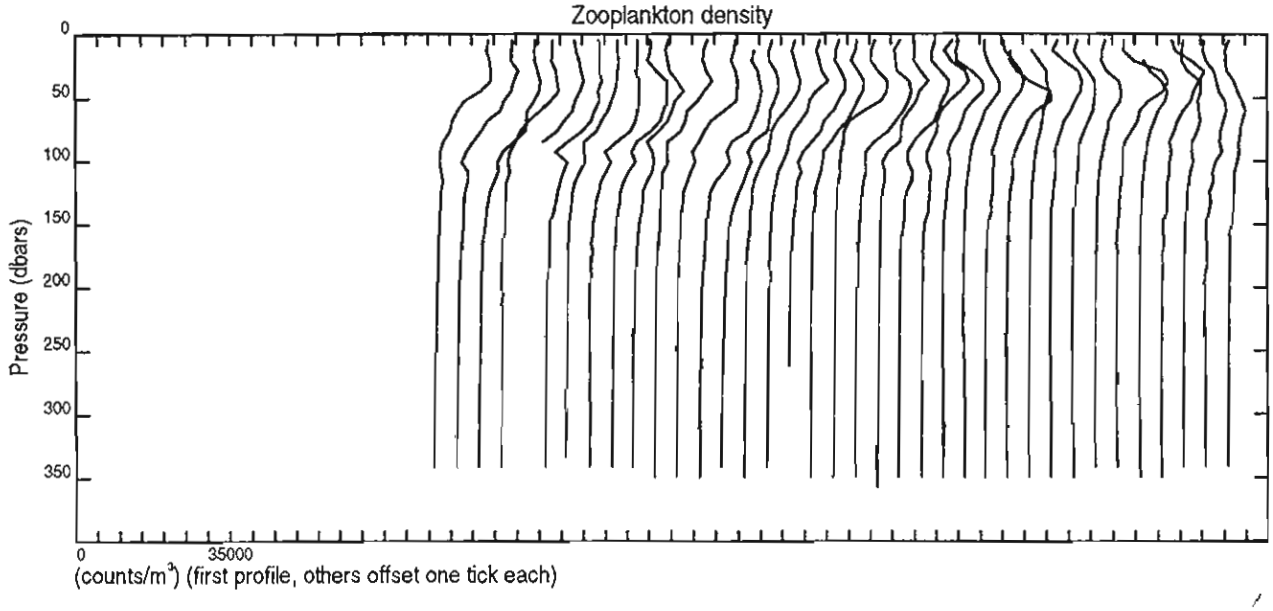


(counts/m^3) (first profile, others offset one tick each)

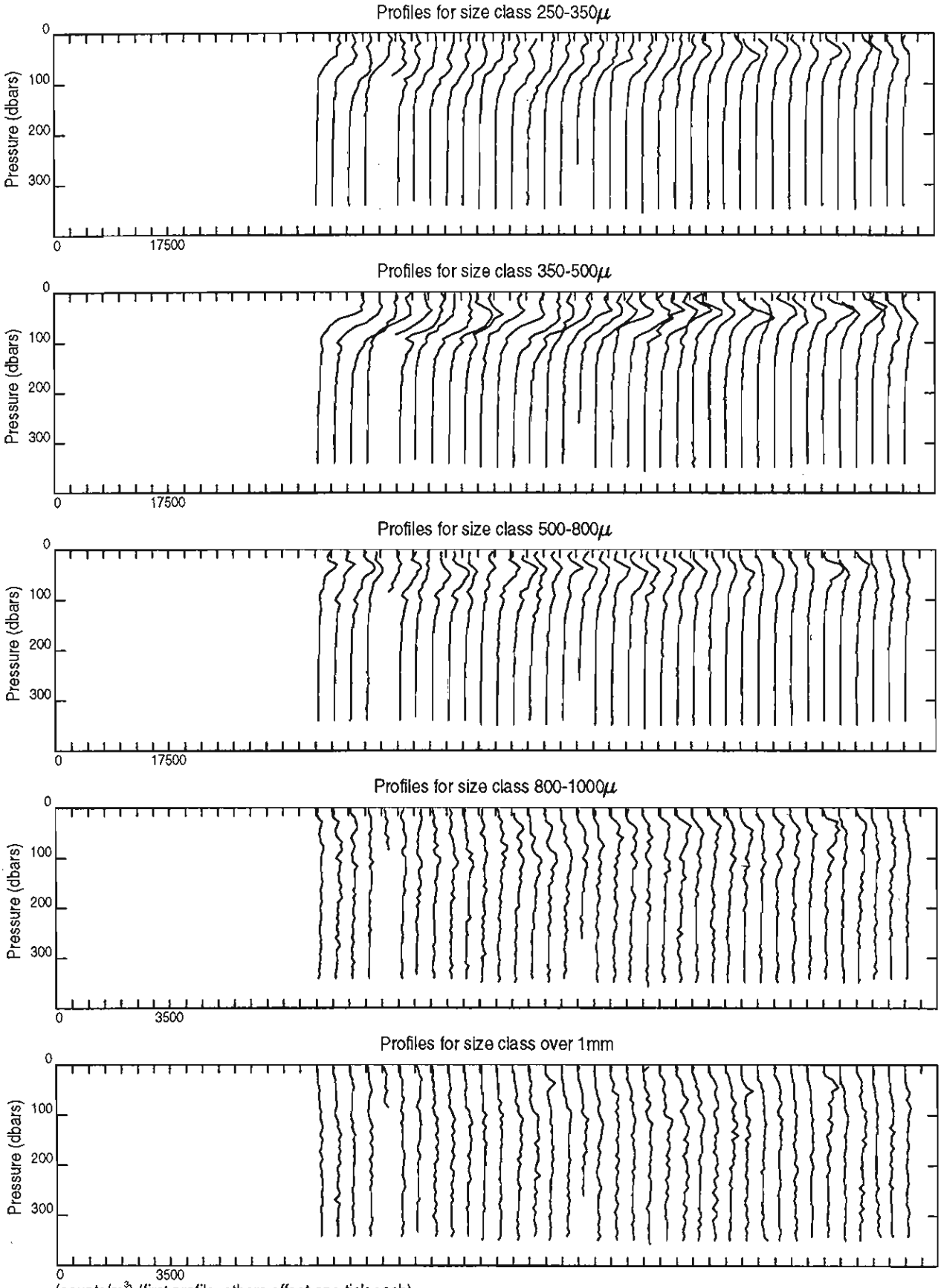
SeaSoar Run 6.1





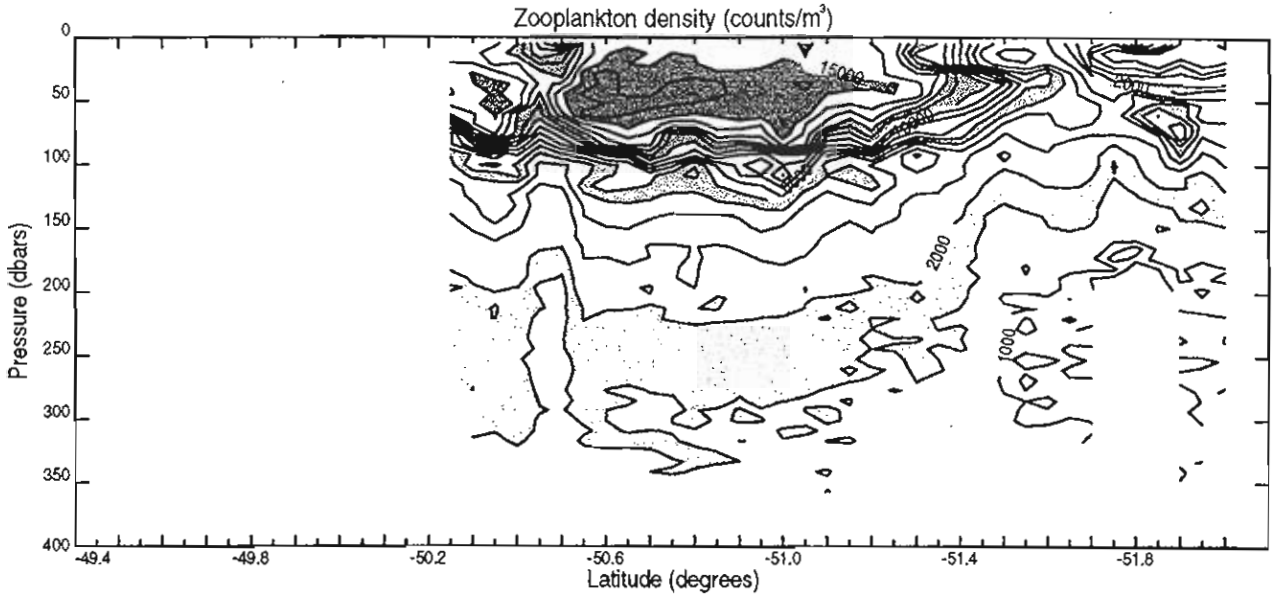


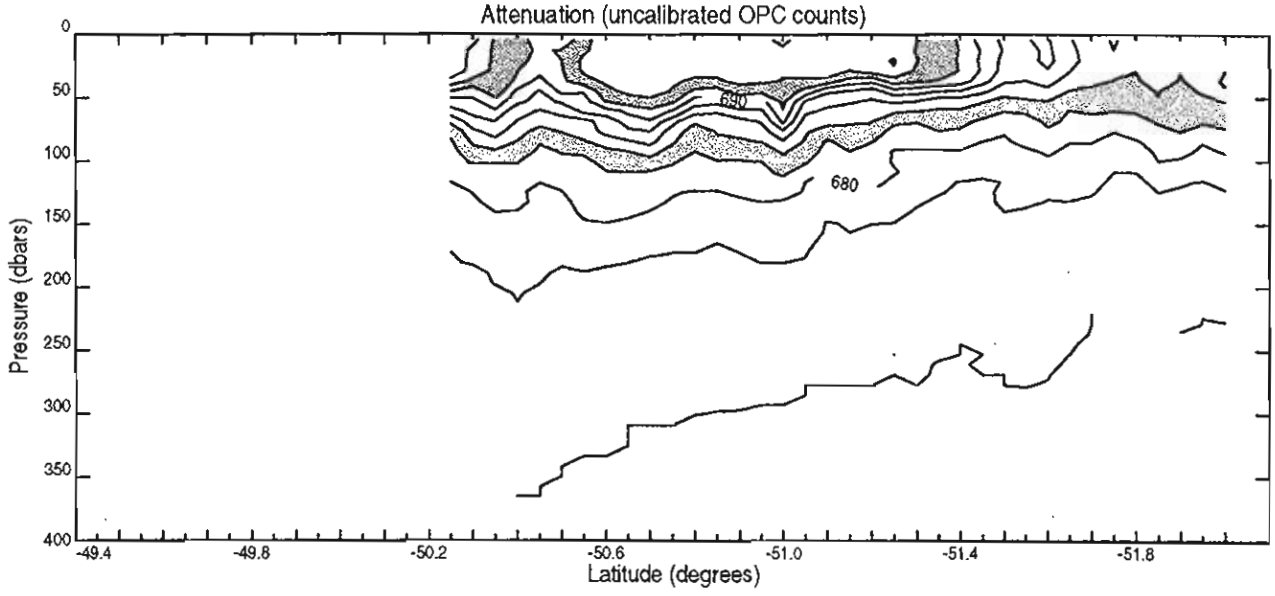
SeaSoar Run 6.1

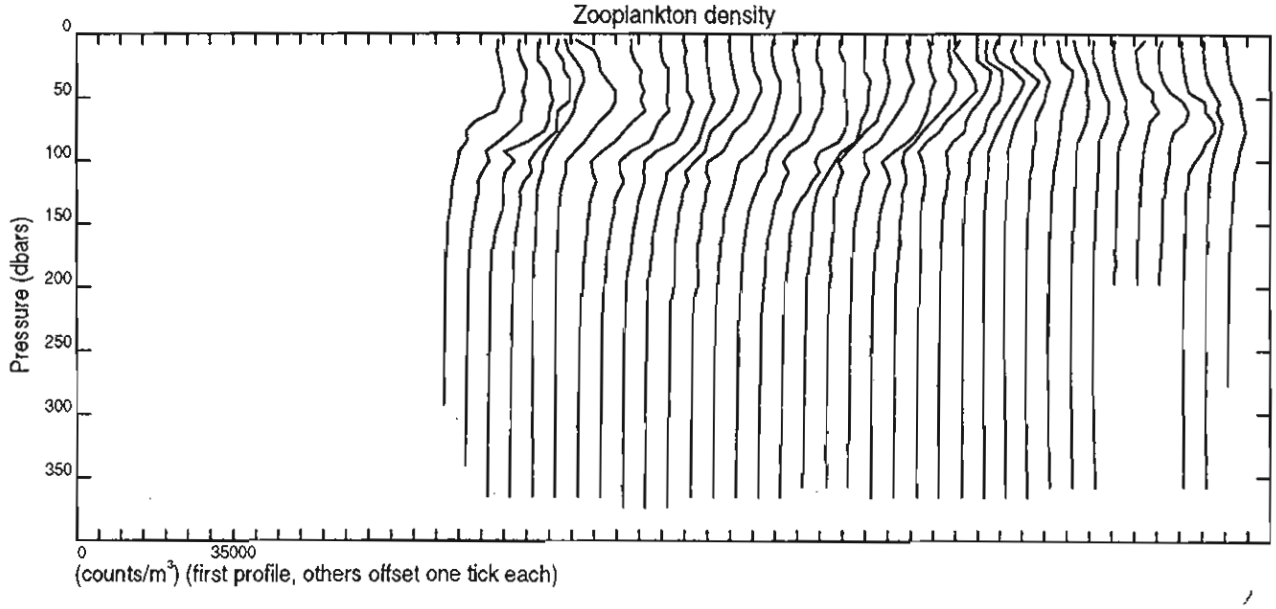


(counts/m^3) (first profile, others offset one tick each)

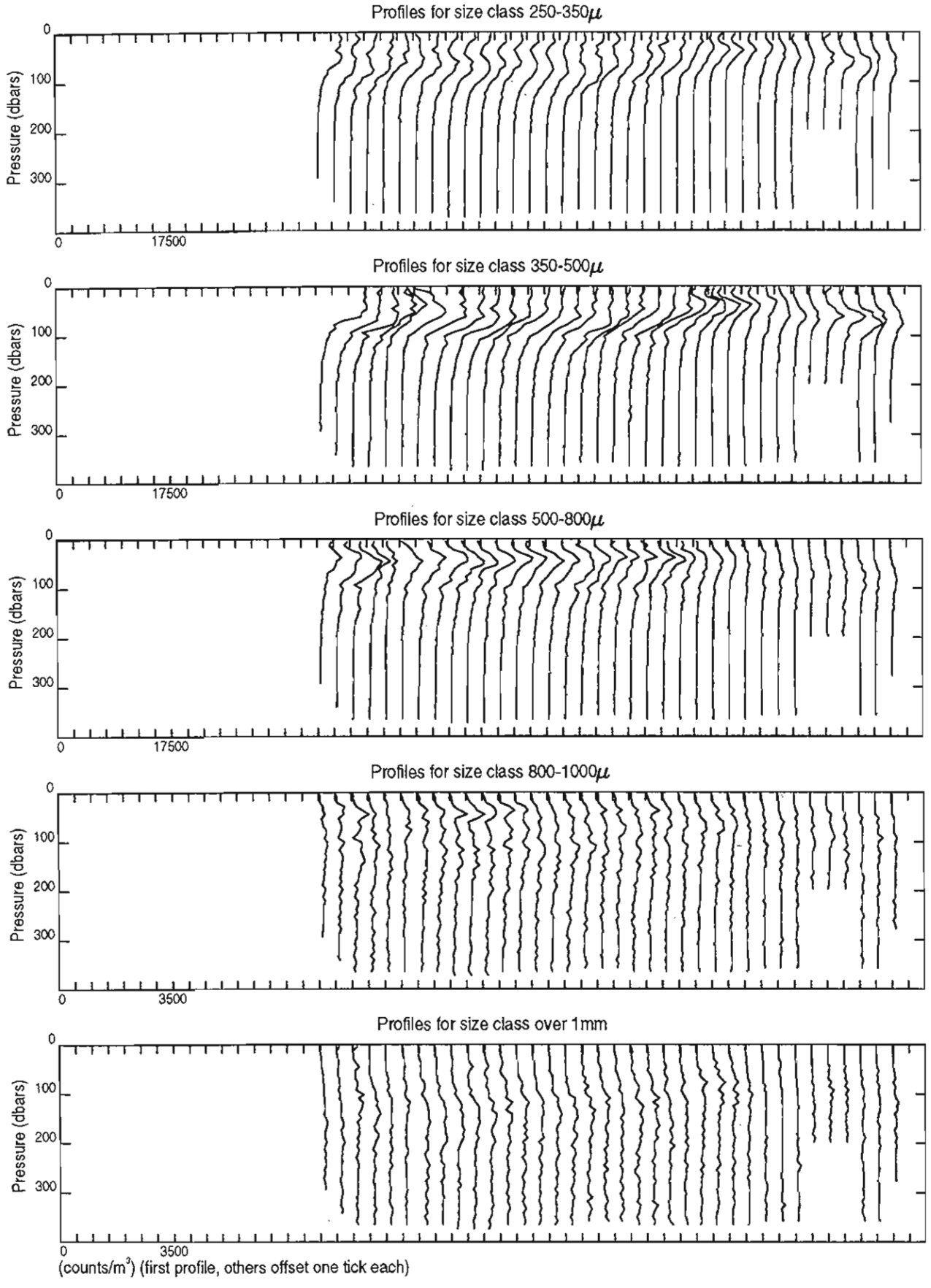
SeaSoar Run 6.2



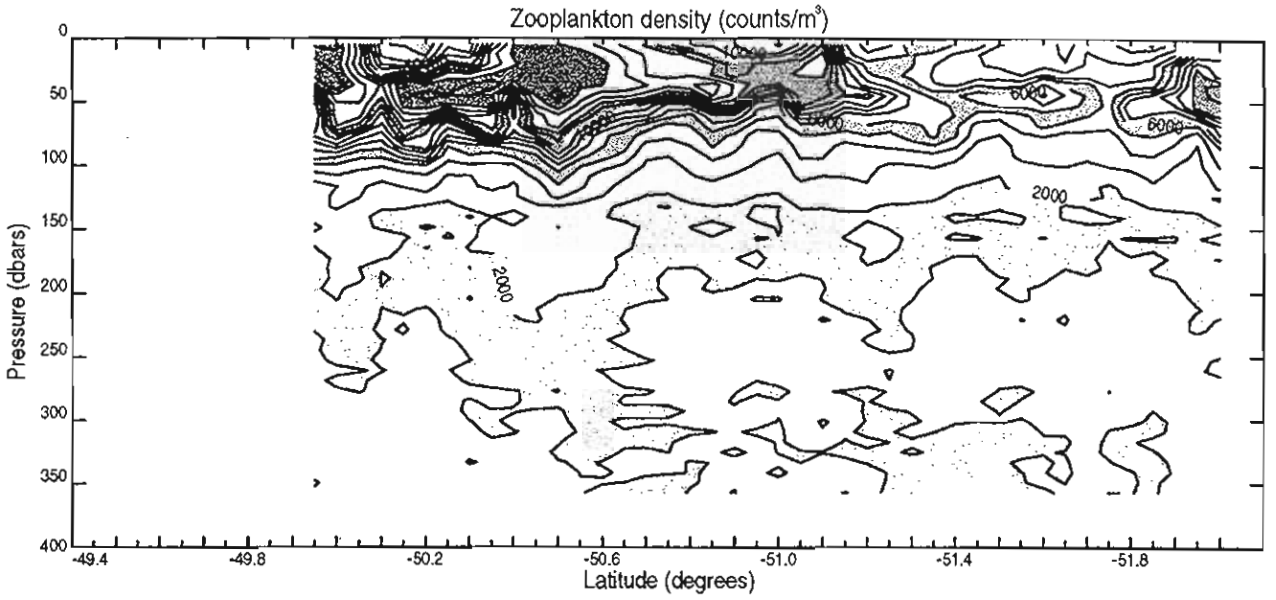


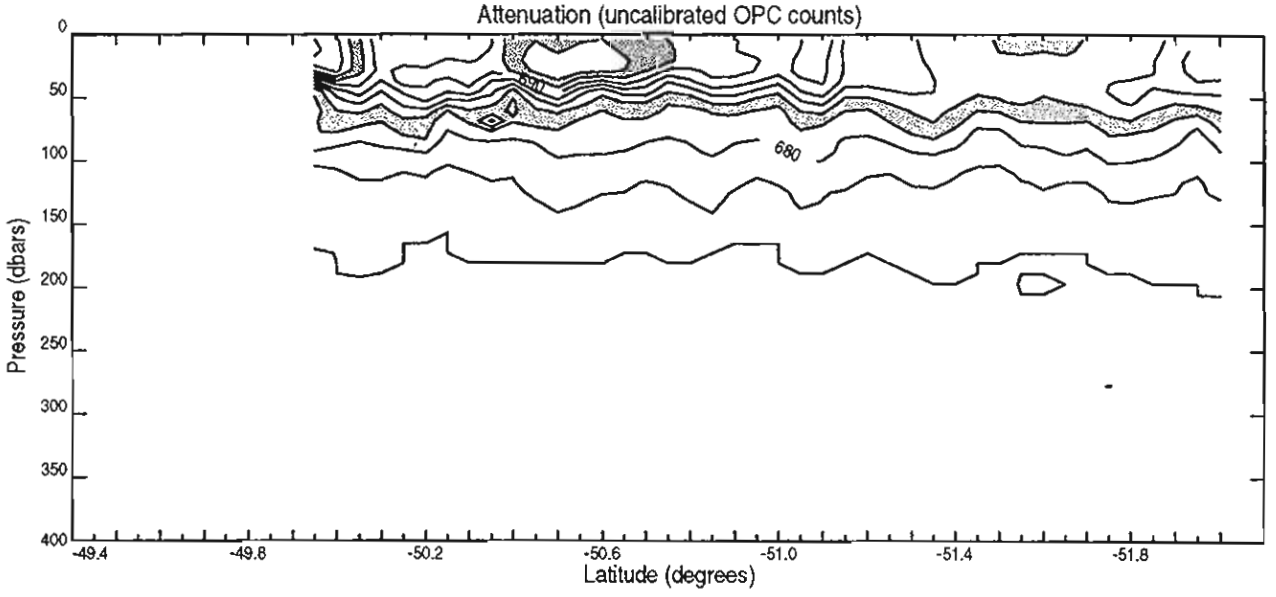


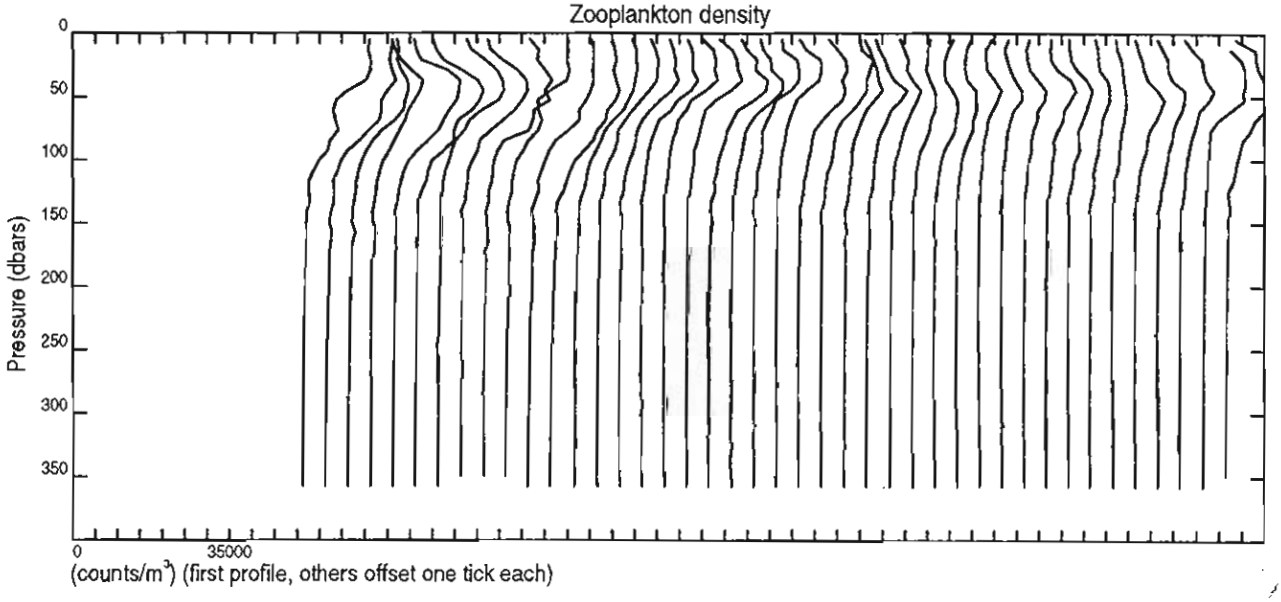
SeaSoar Run 6.2



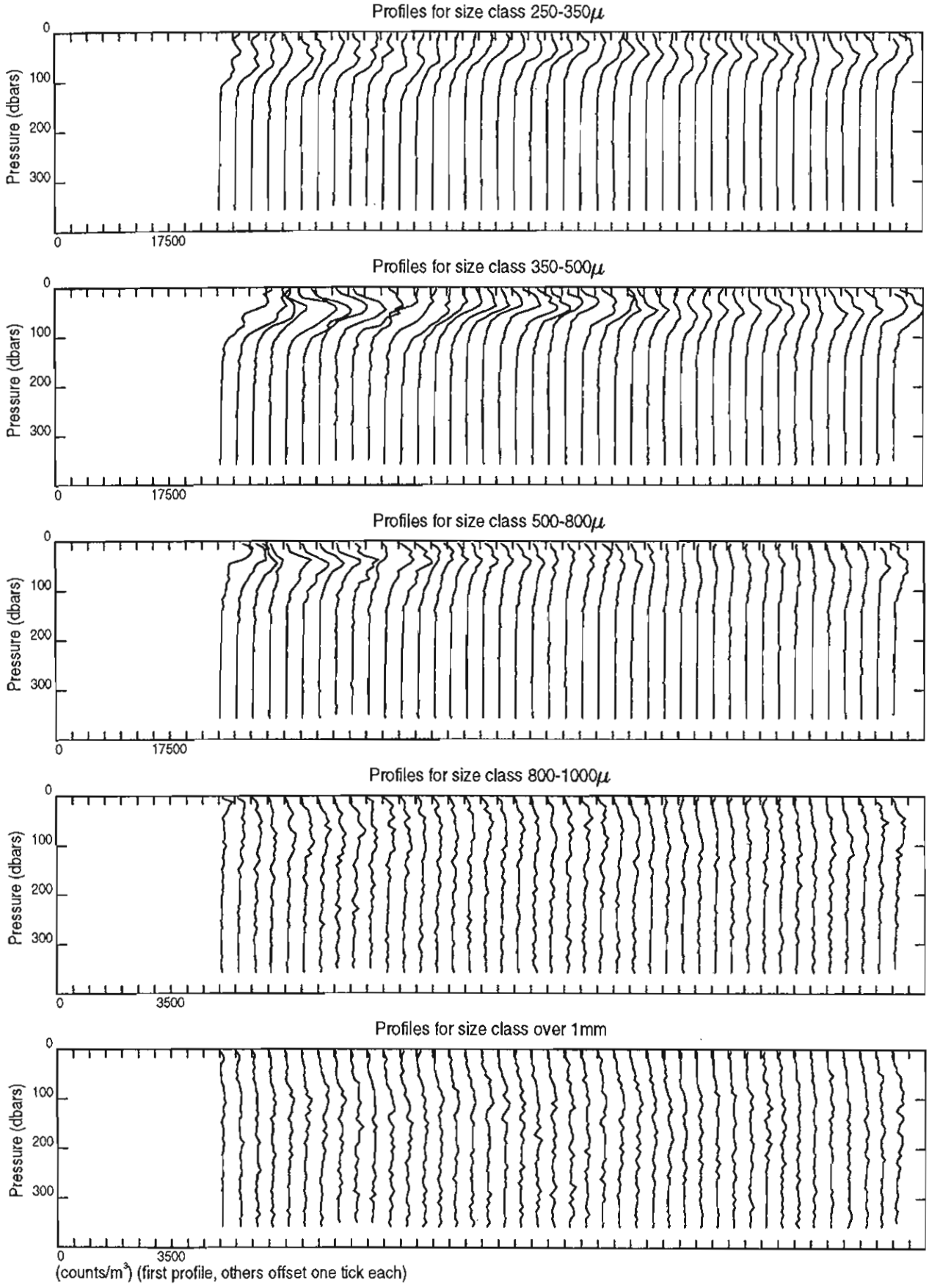
SeaSoar Run 6.3



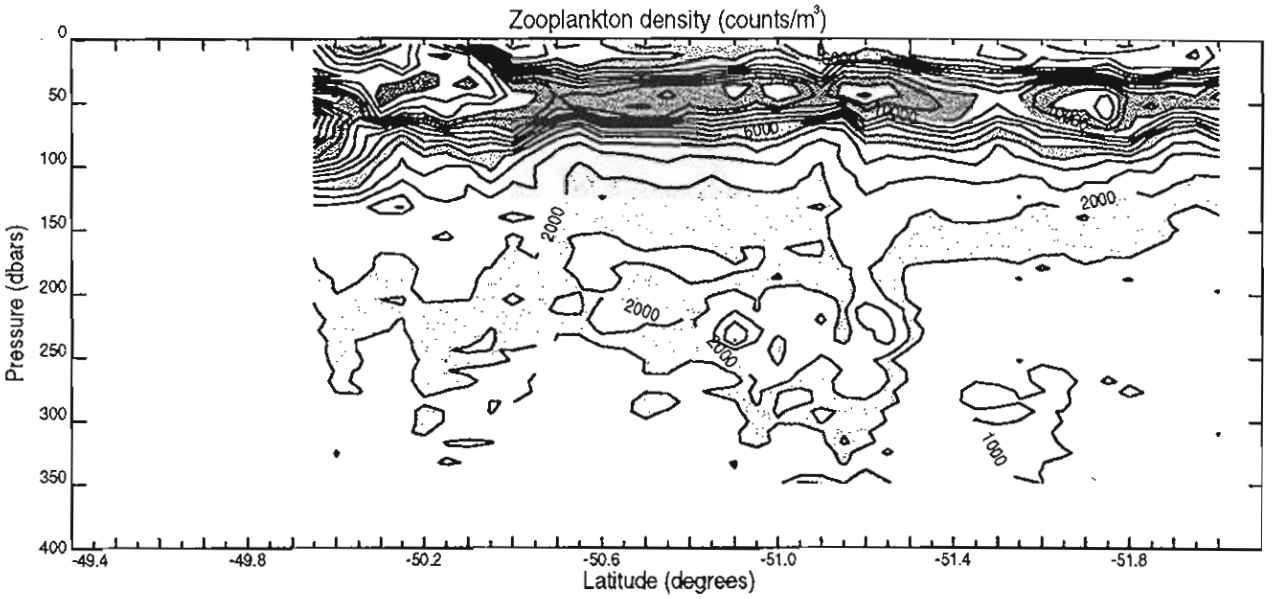


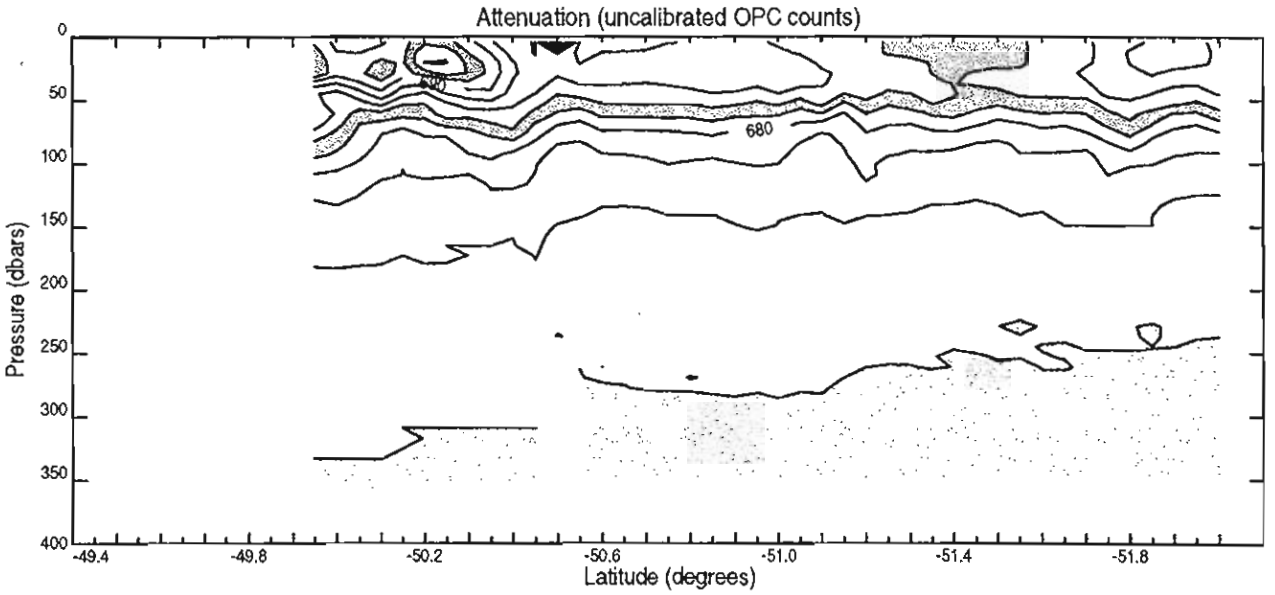


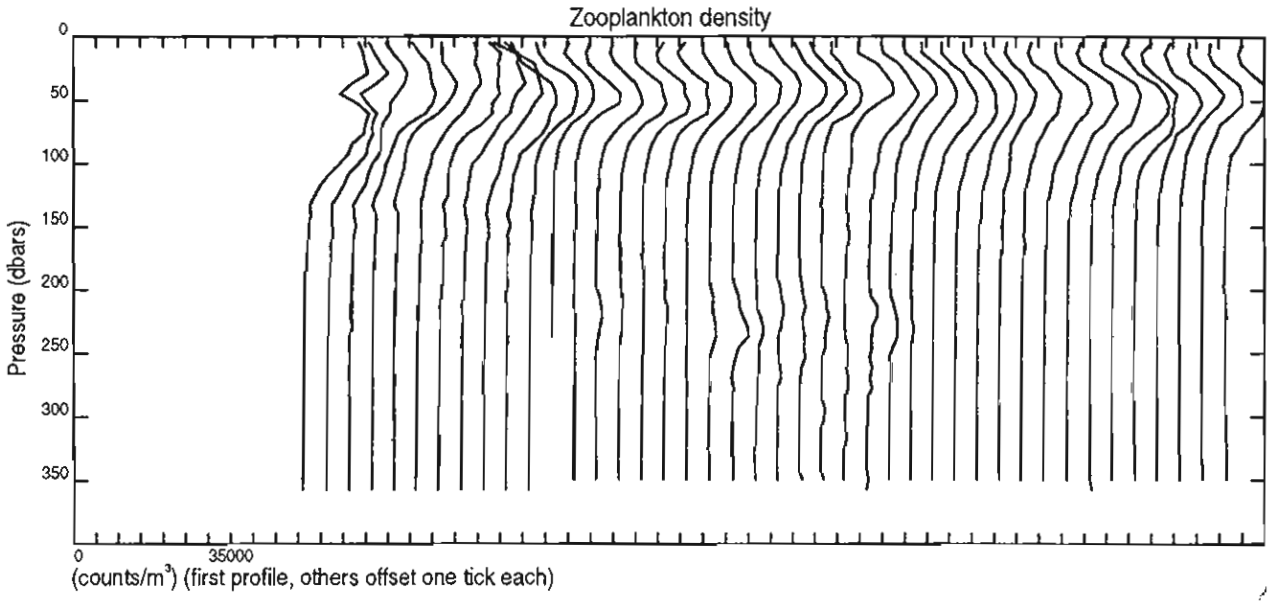
SeaSoar Run 6.3



SeaSoar Run 6.4

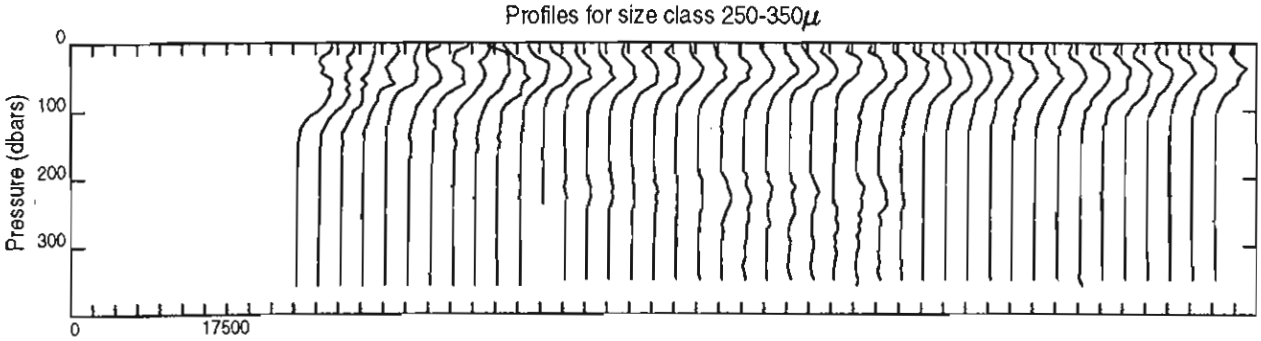




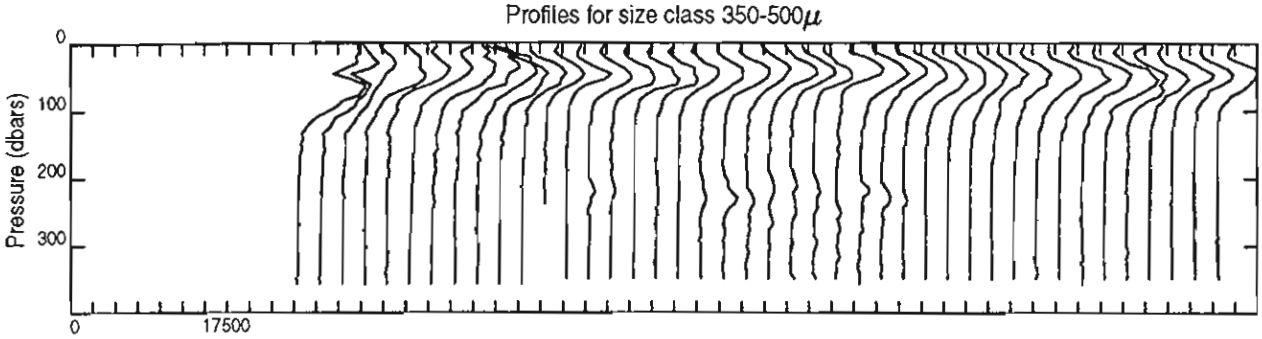


SeaSoar Run 6.4

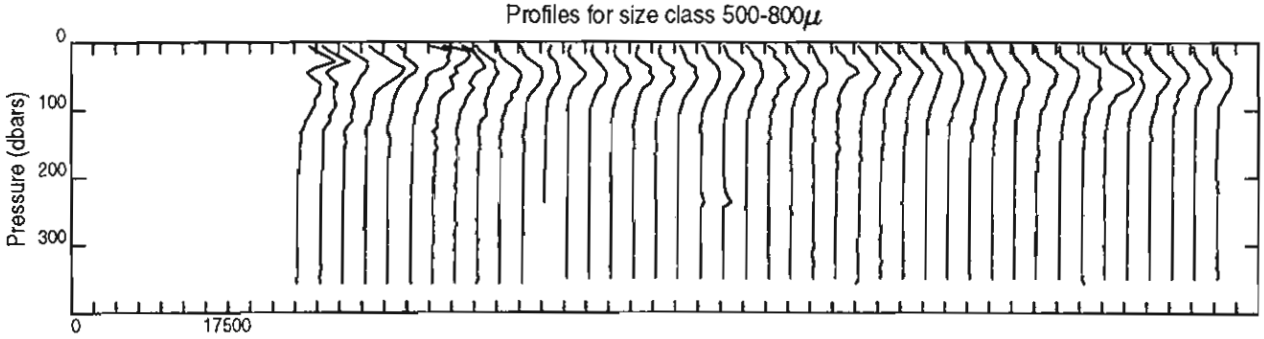
Profiles for size class 250-350μ



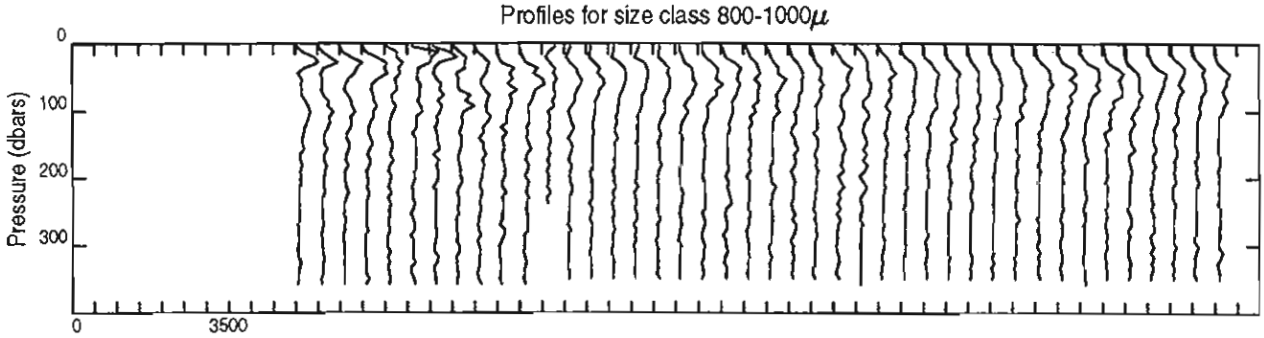
Profiles for size class 350-500μ



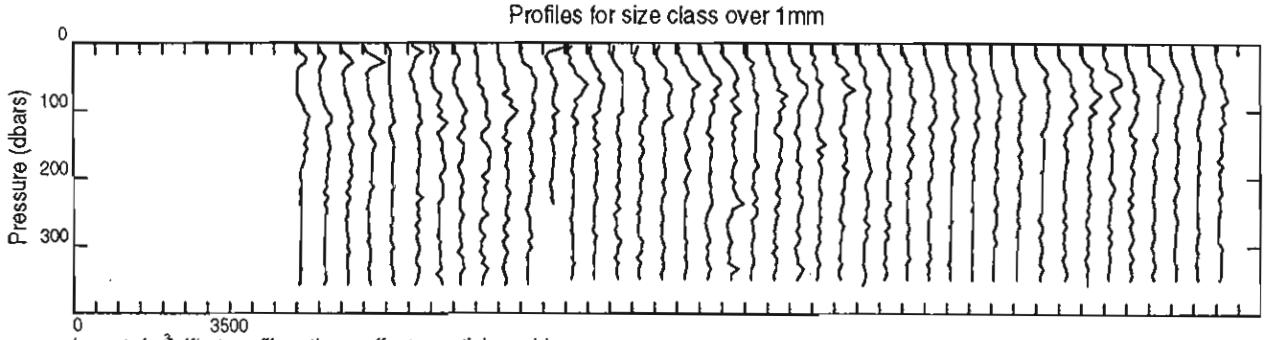
Profiles for size class 500-800μ



Profiles for size class 800-1000μ

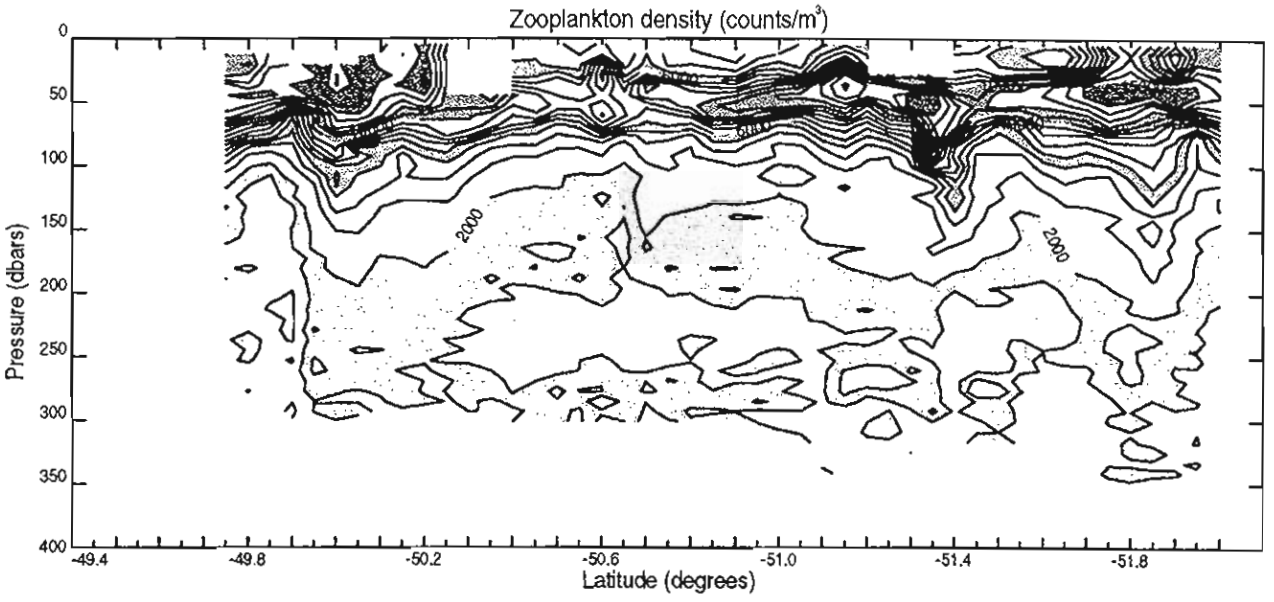


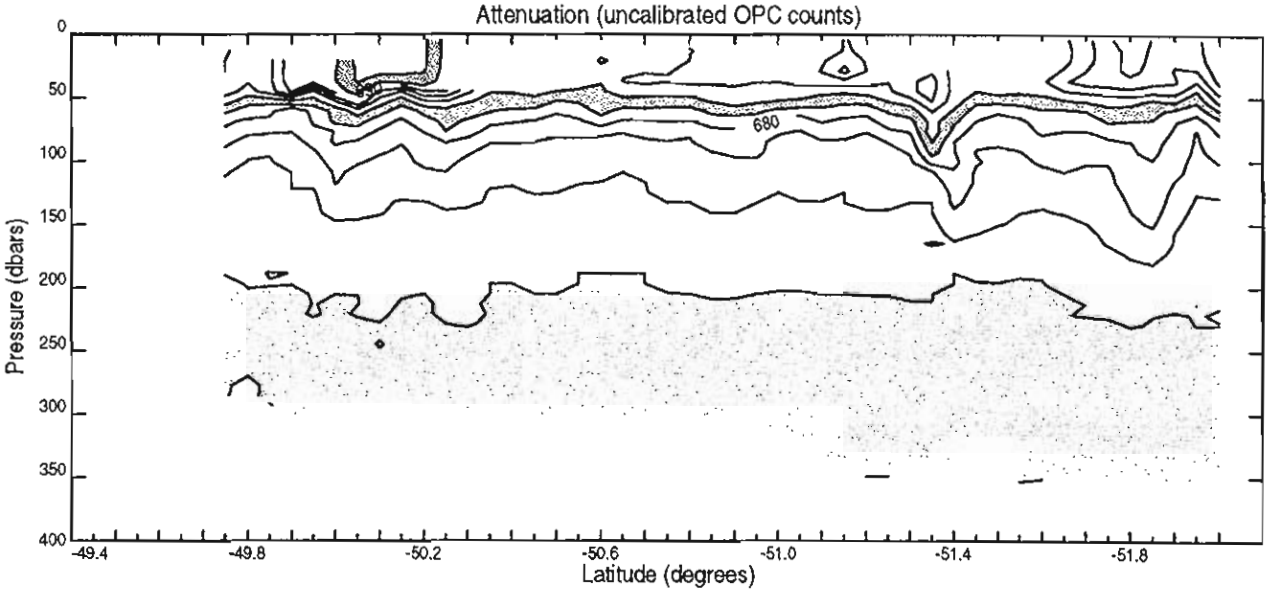
Profiles for size class over 1mm

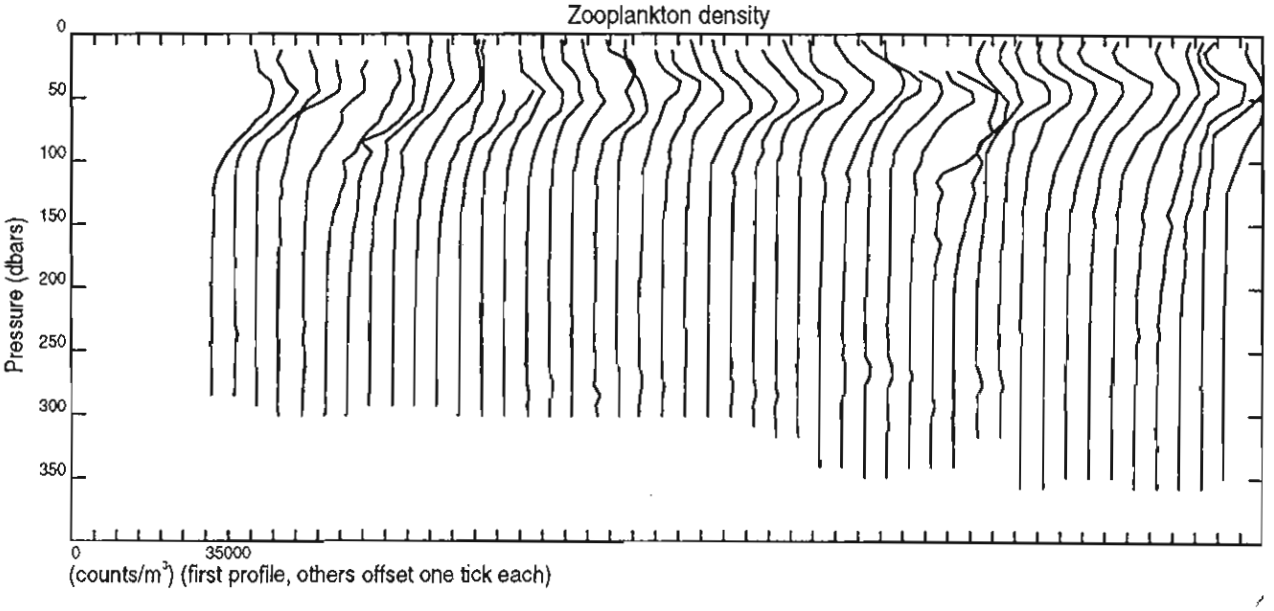


(counts/m^3) (first profile, others offset one tick each)

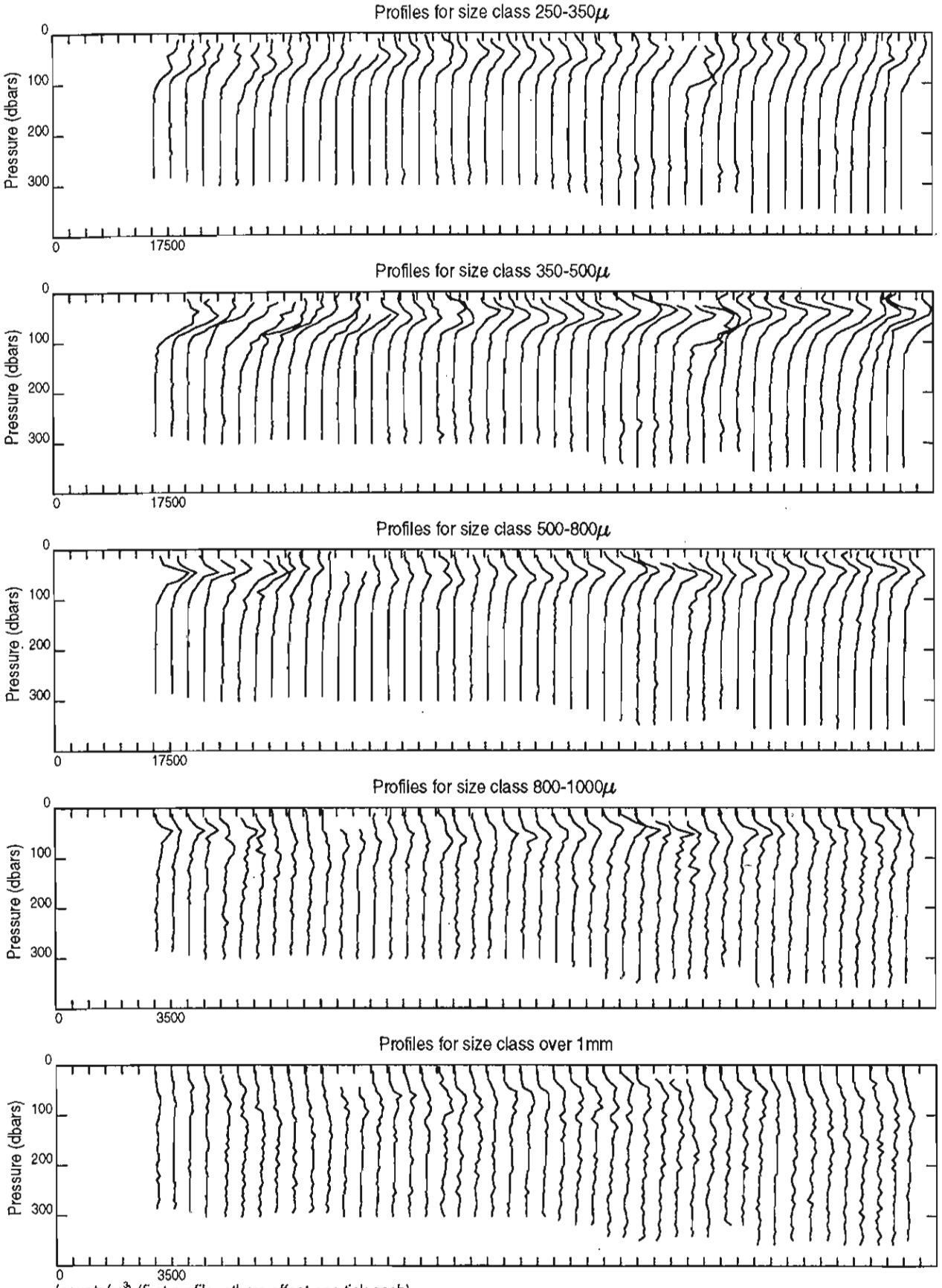
SeaSoar Run 6.5





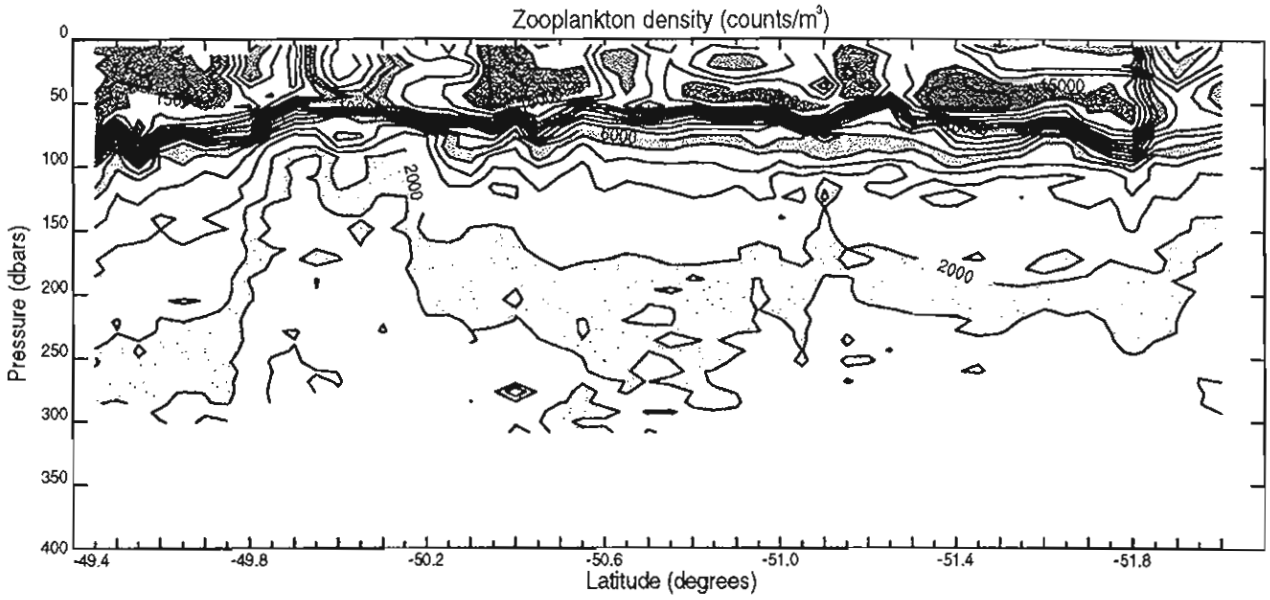


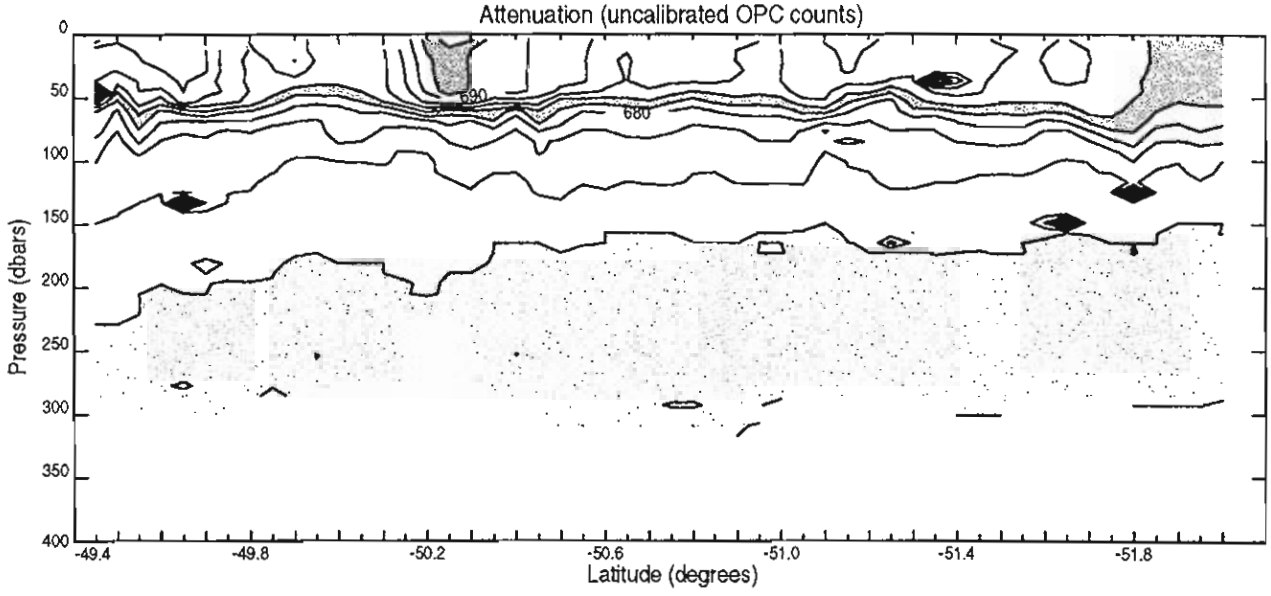
SeaSoar Run 6.5

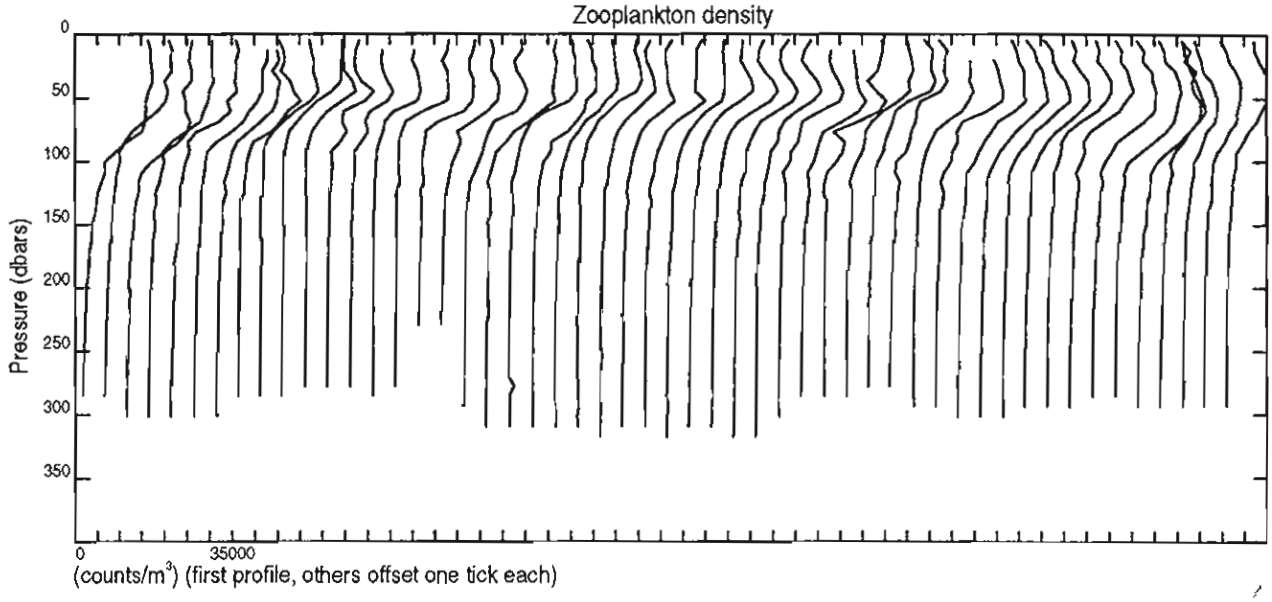


(counts/m^3) (first profile, others offset one tick each)

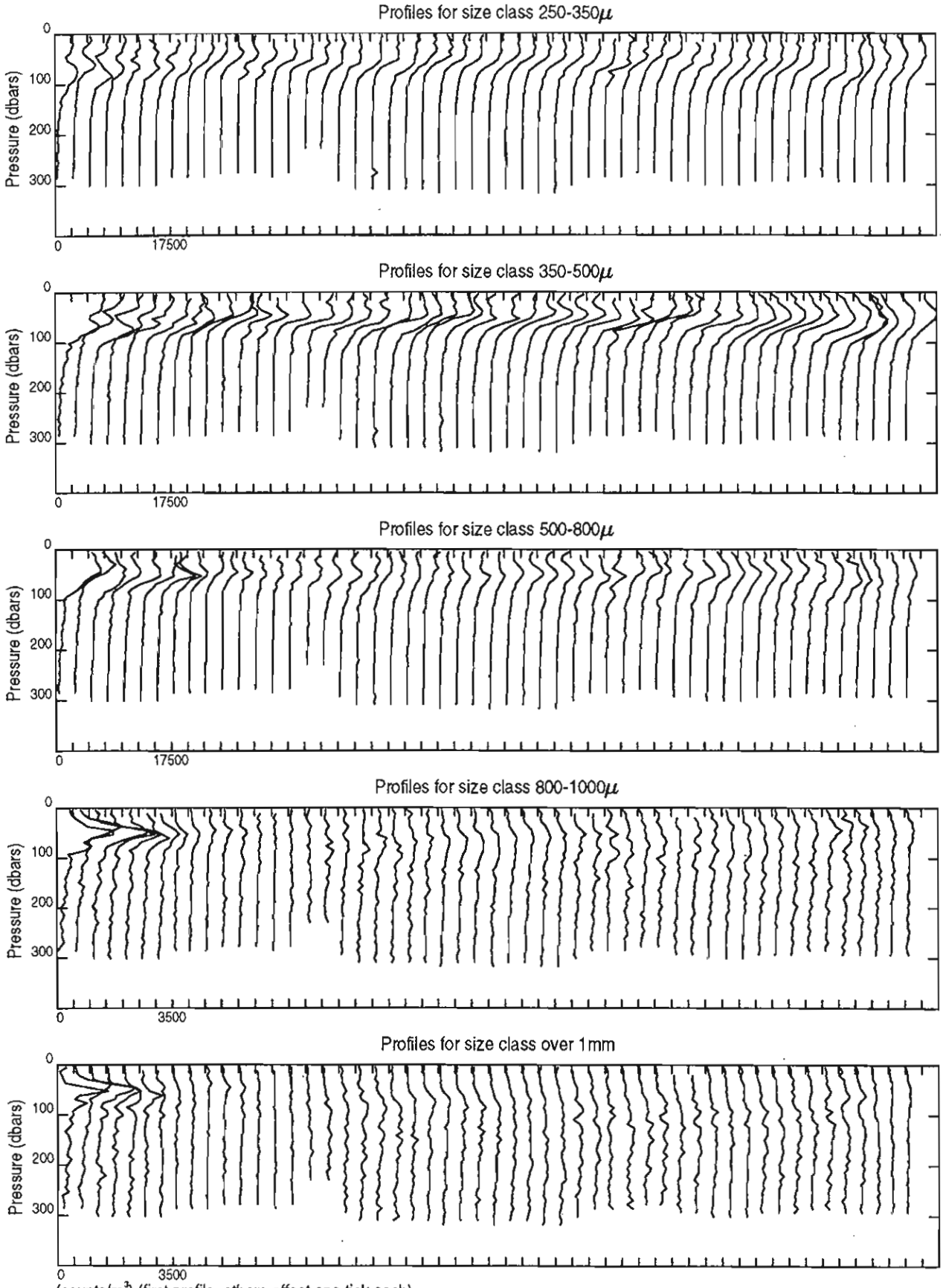
SeaSoar Run 6.6





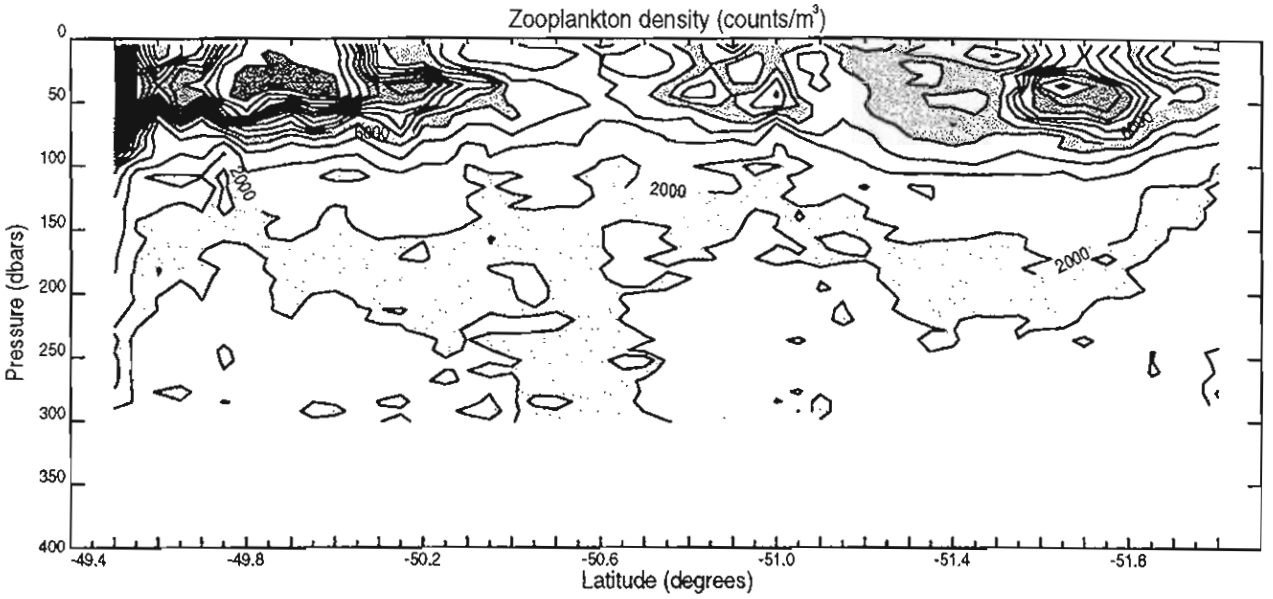


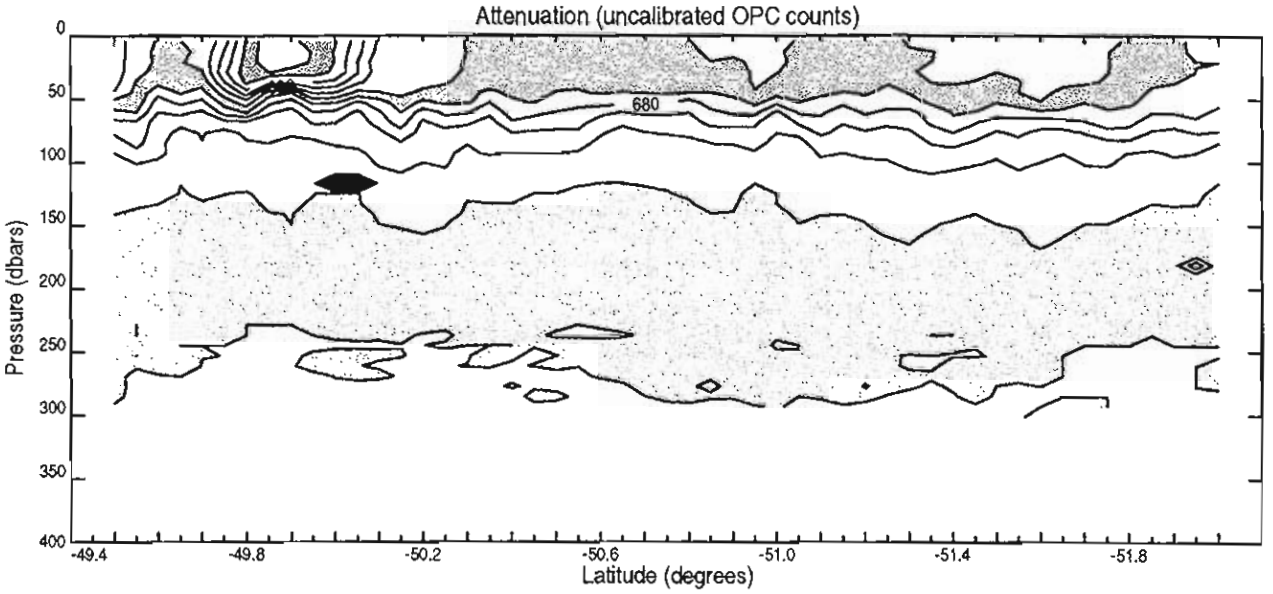
SeaSoar Run 6.6

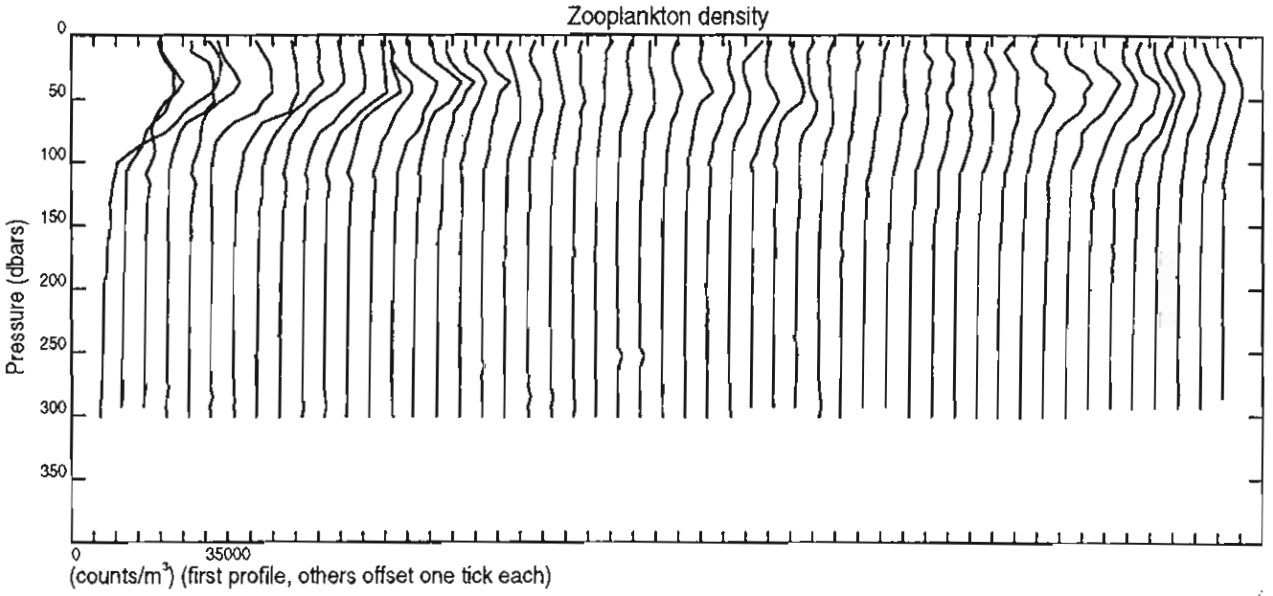


(counts/m^3) (first profile, others offset one tick each)

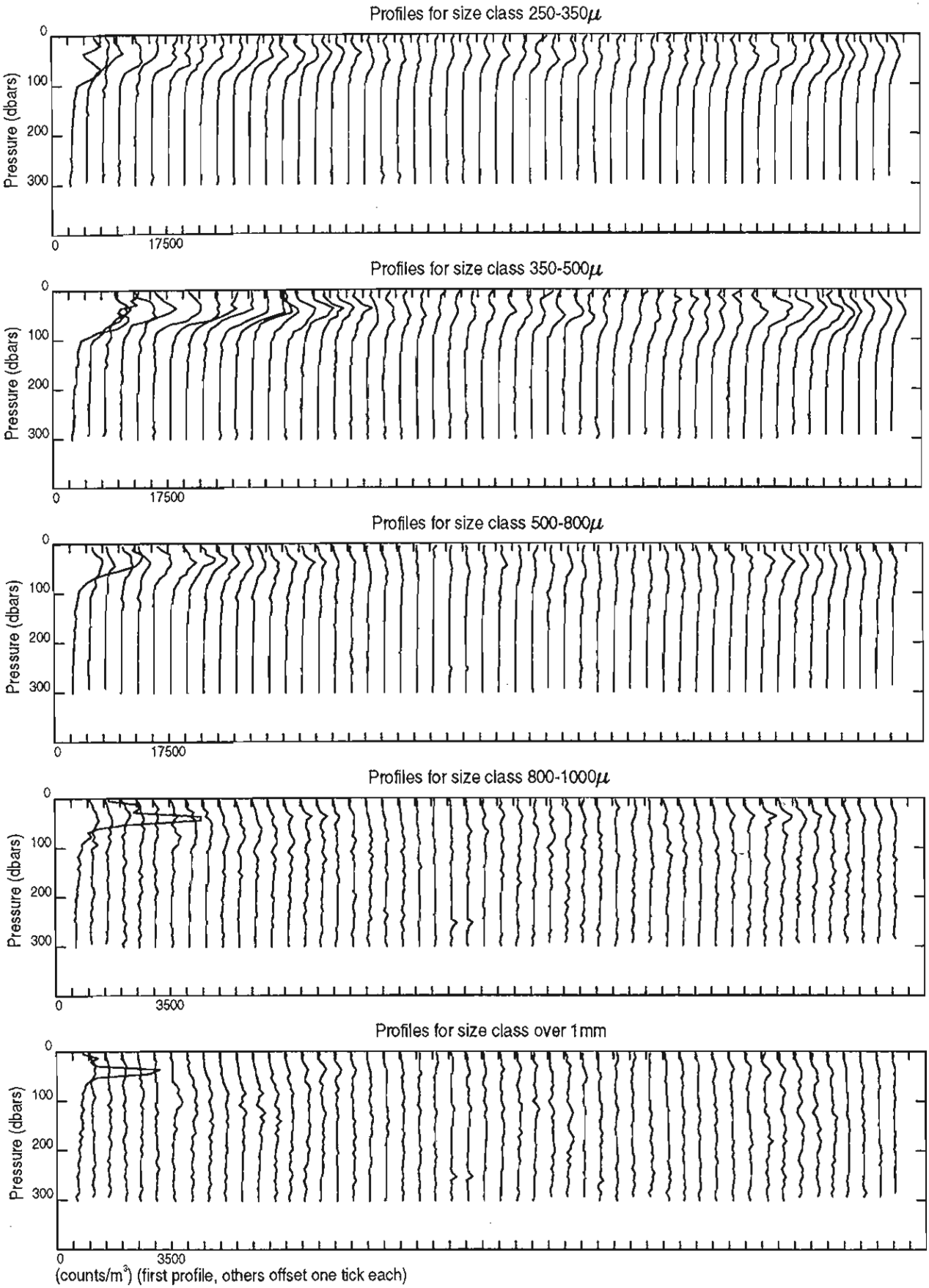
SeaSoar Run 6.7



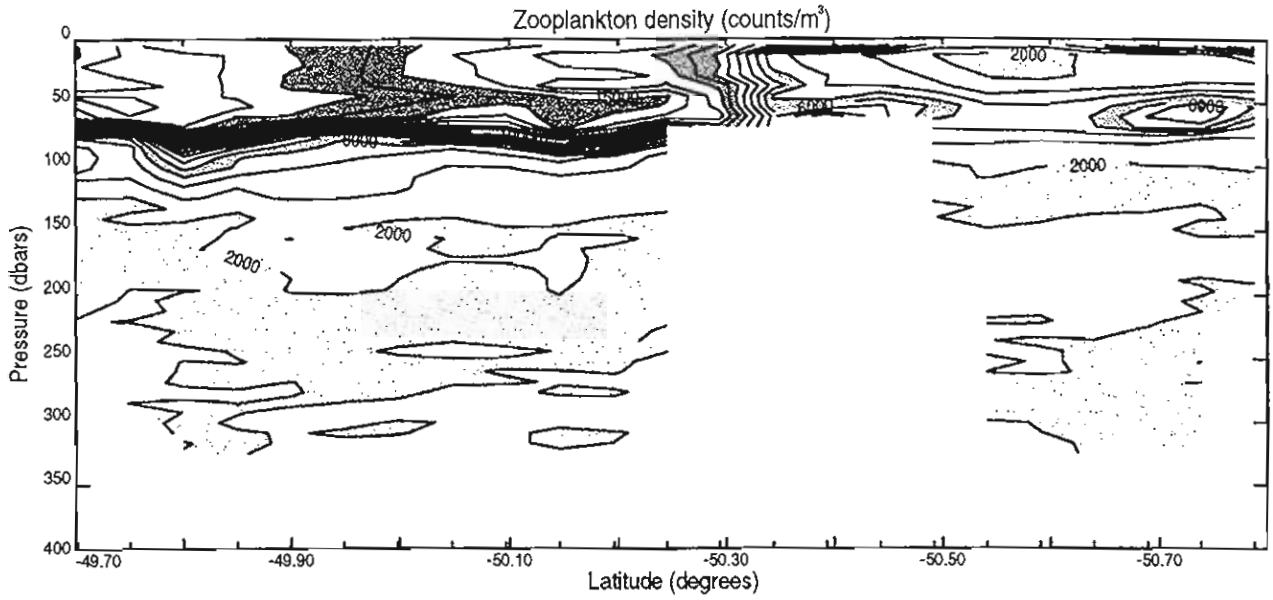


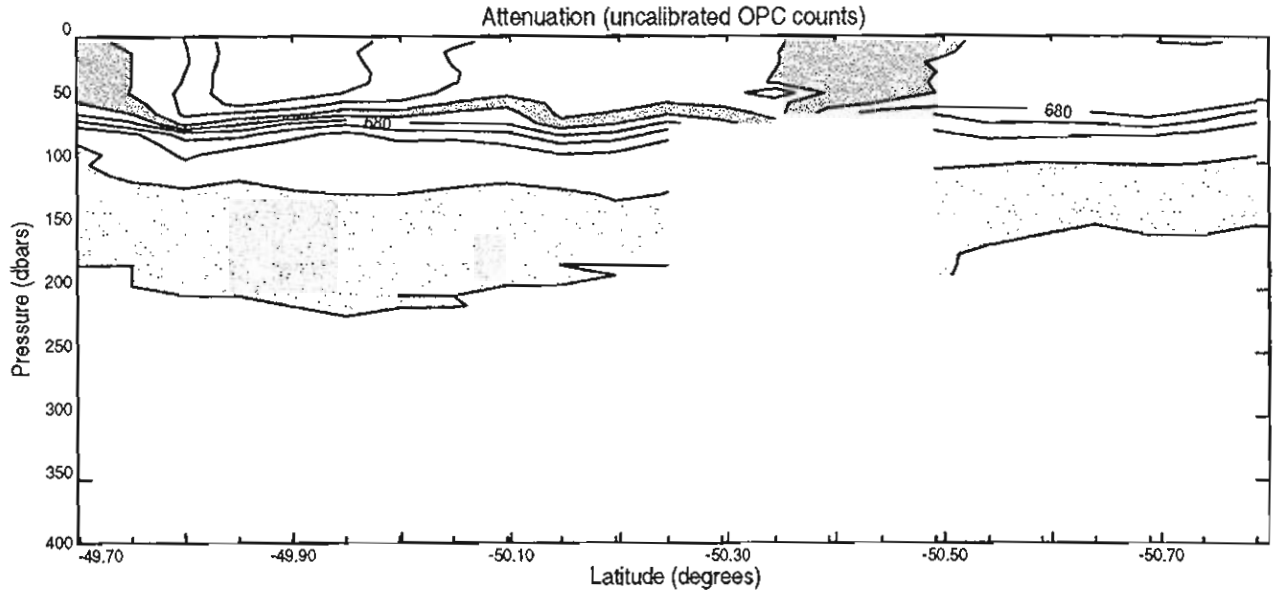


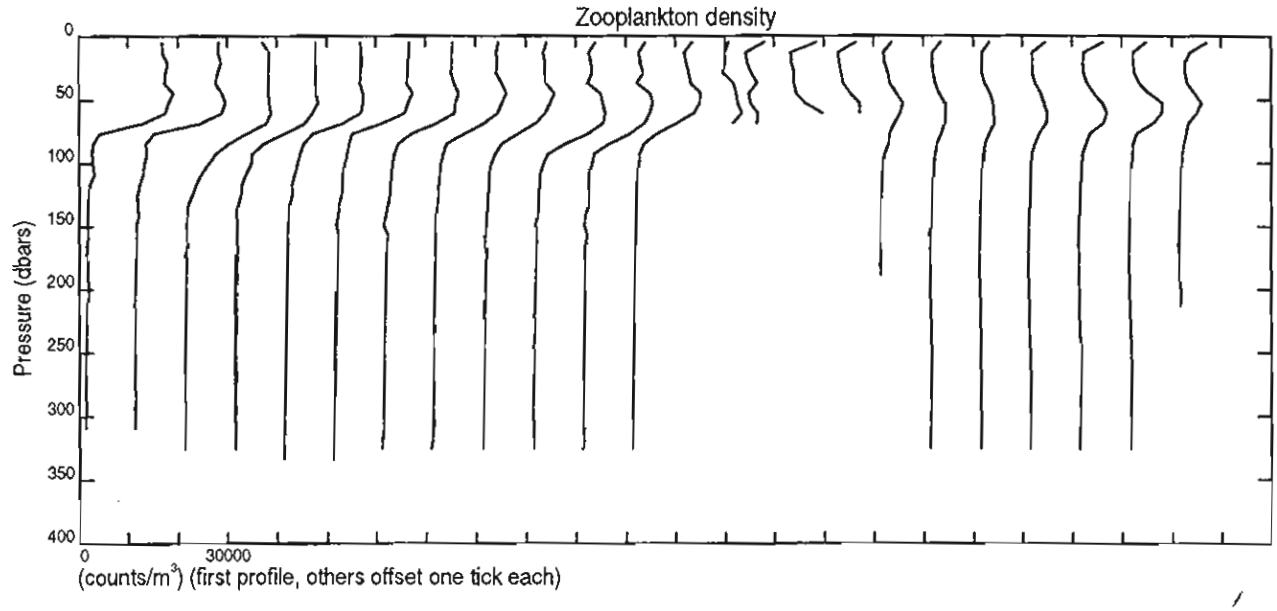
SeaSoar Run 6.7



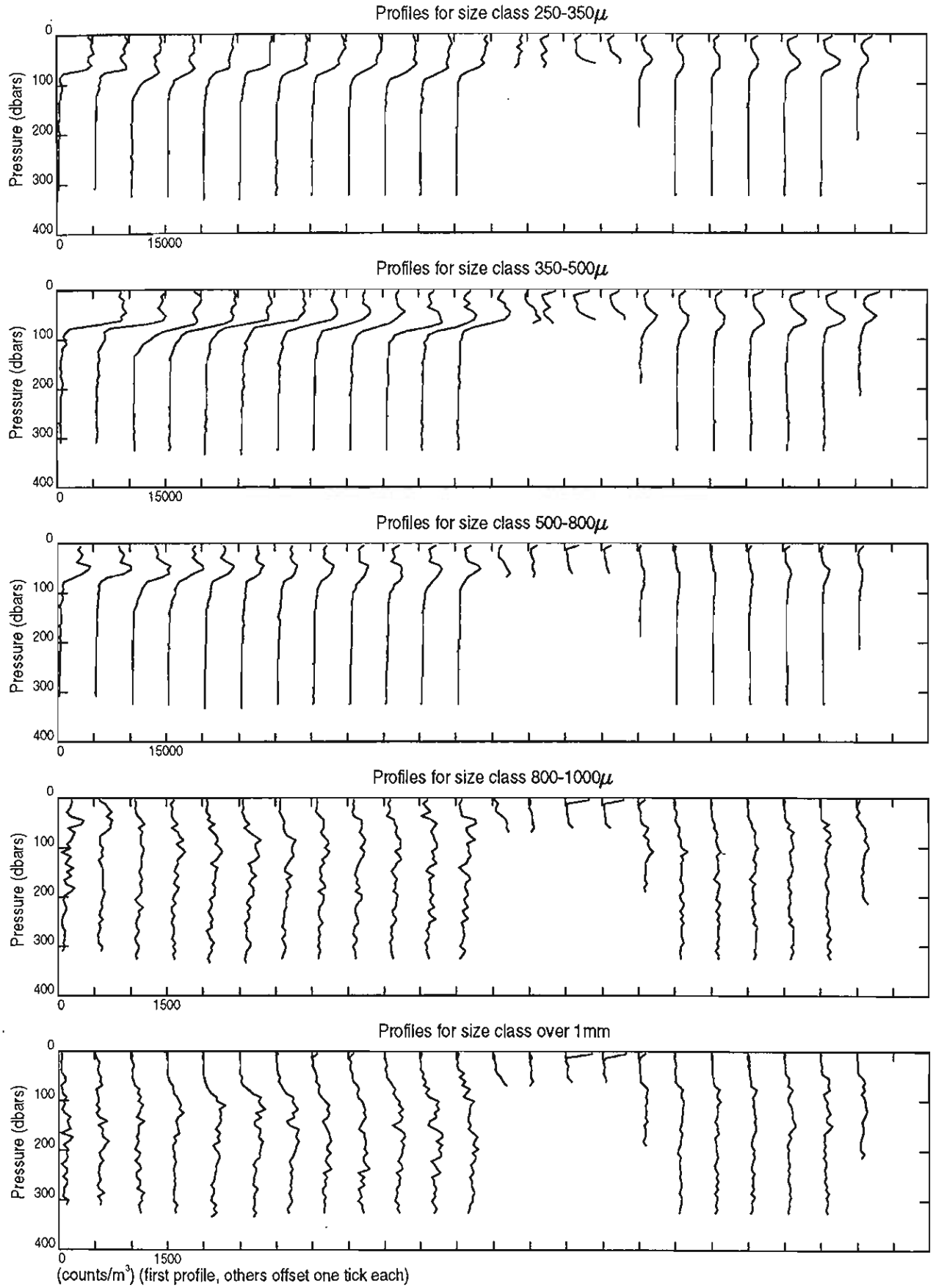
SeaSoar Fine Scale Survey - Run 8.1



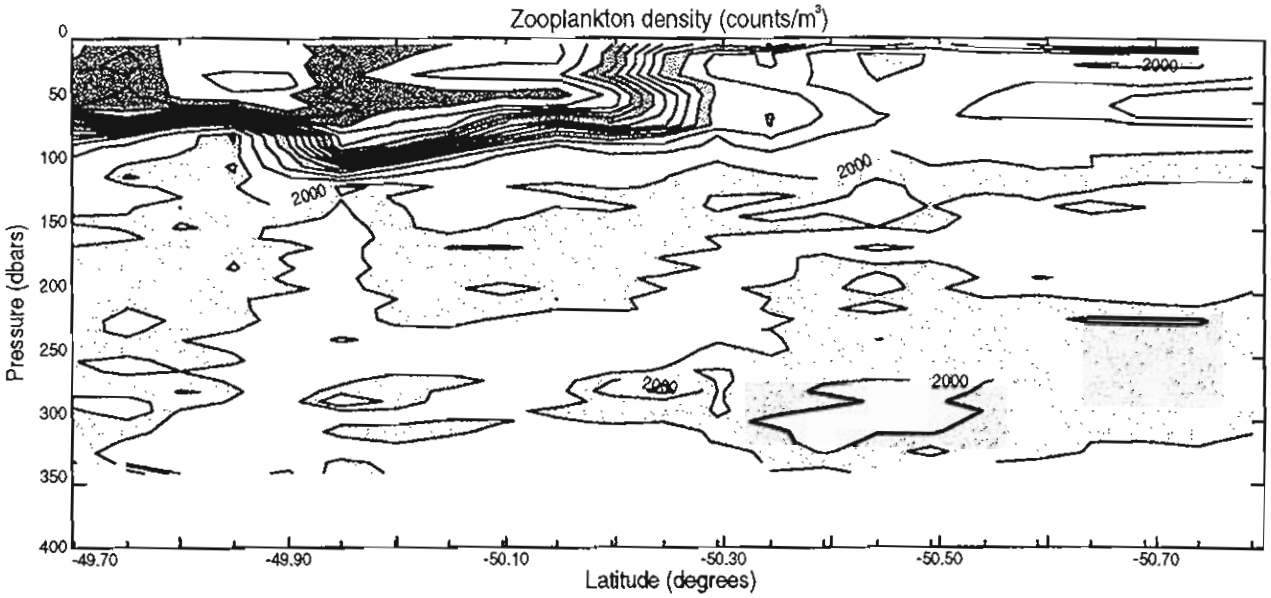


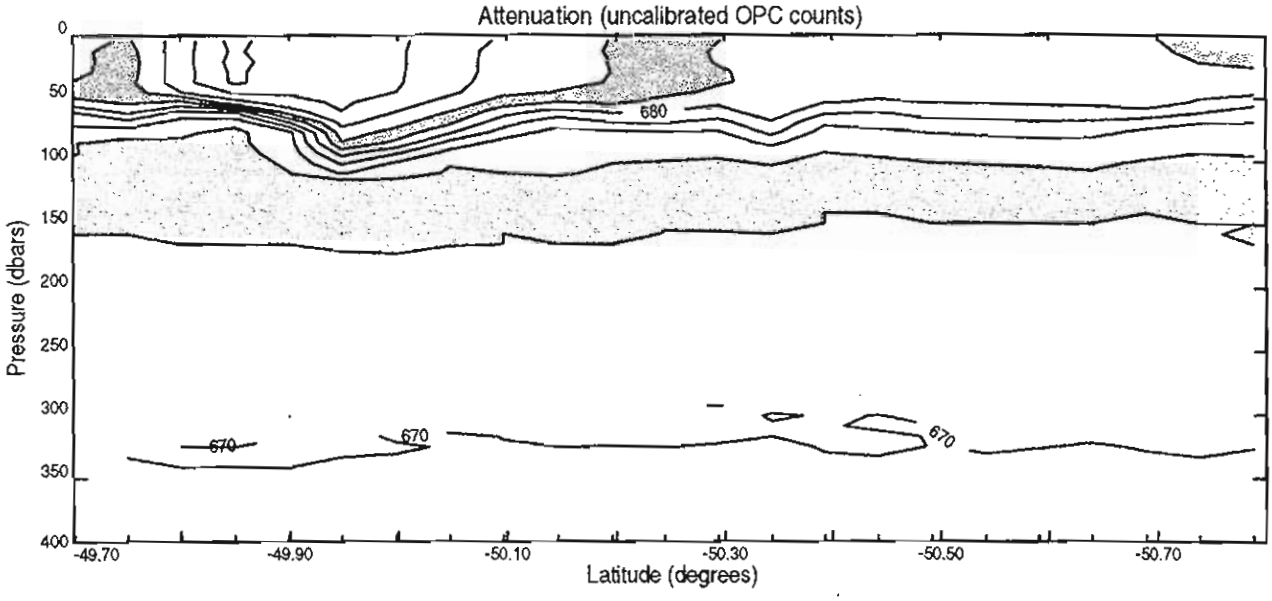


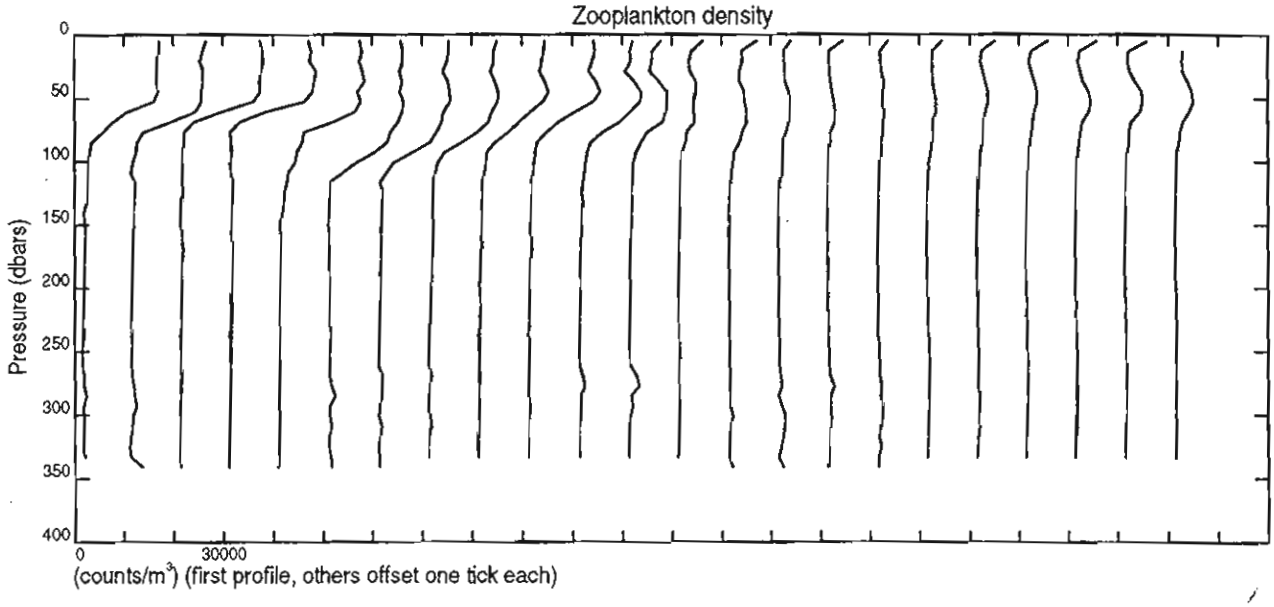
SeaSoar Fine Scale Survey - Run 8.1



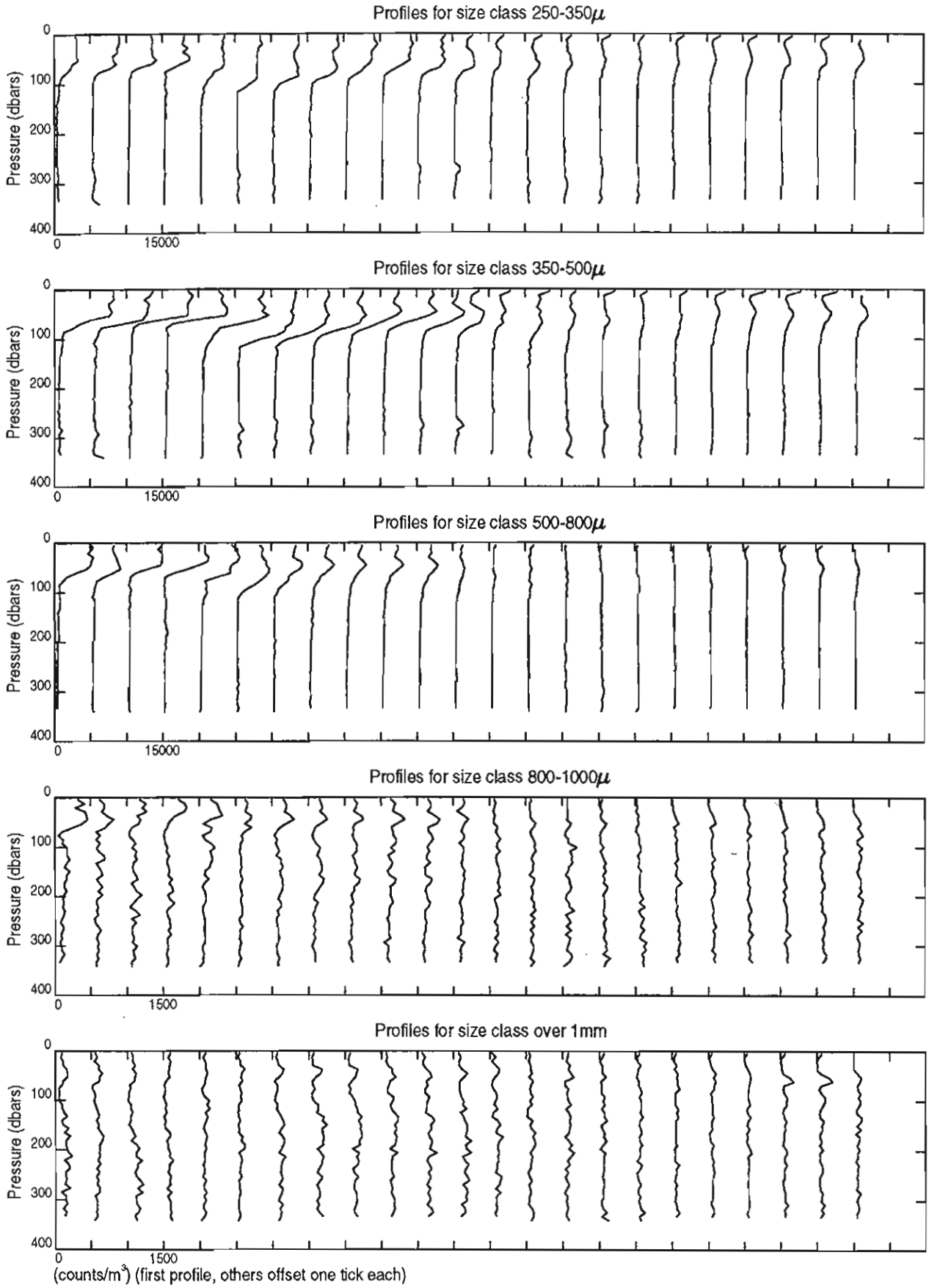
SeaSoar Fine Scale Survey - Run 8.2



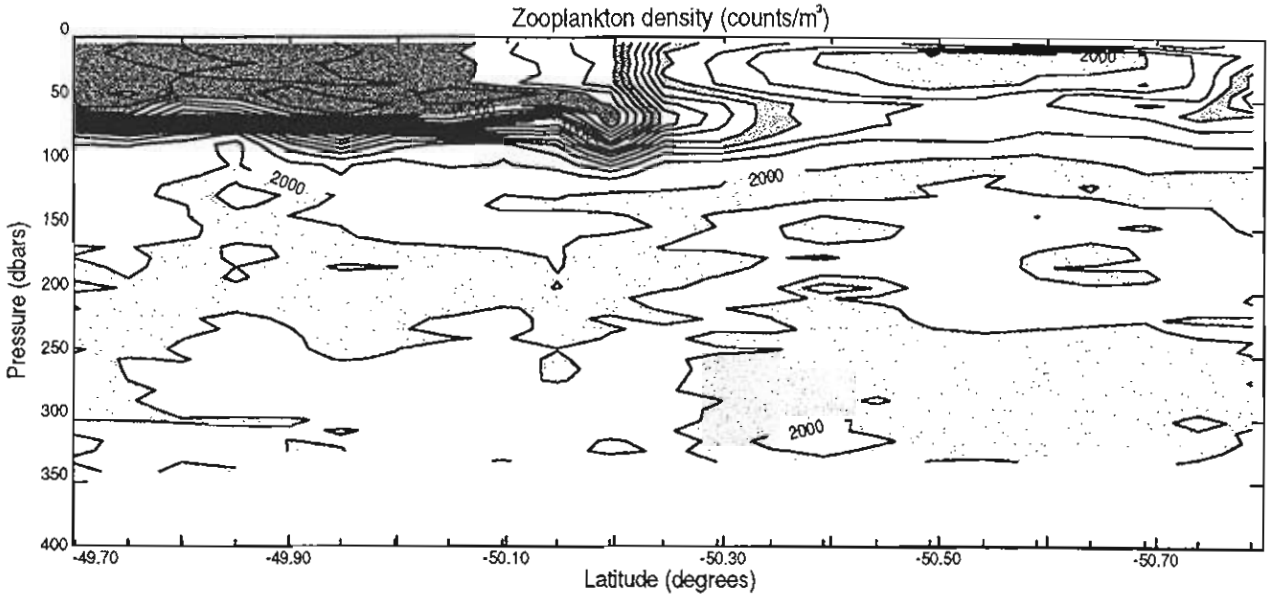


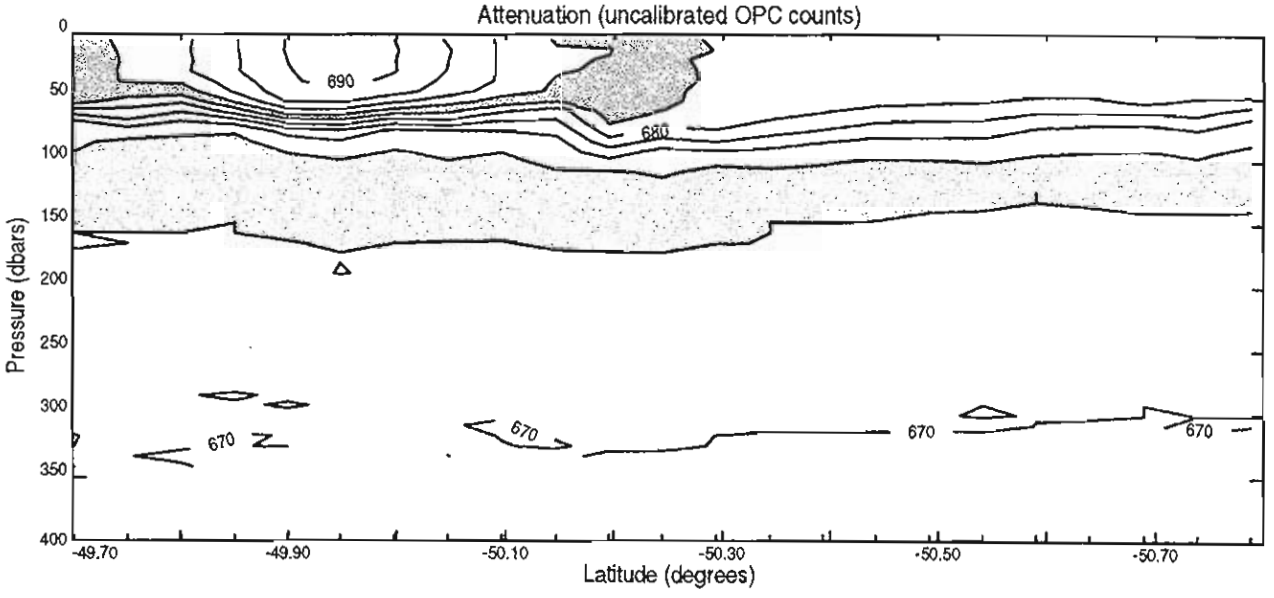


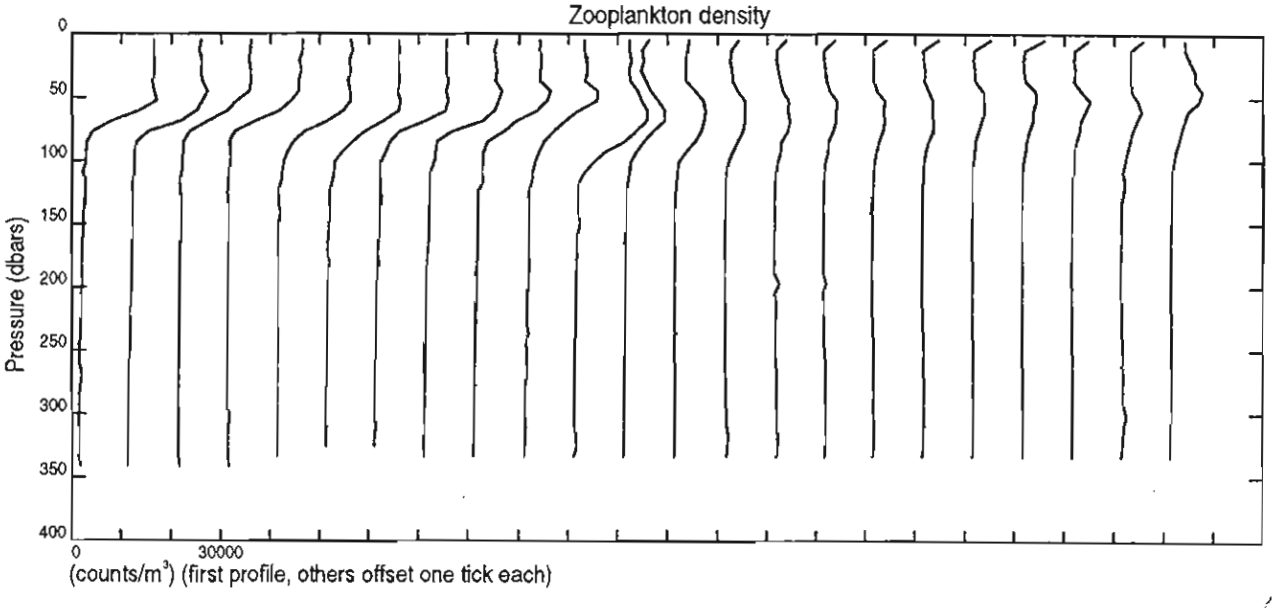
SeaSoar Fine Scale Survey - Run 8.2



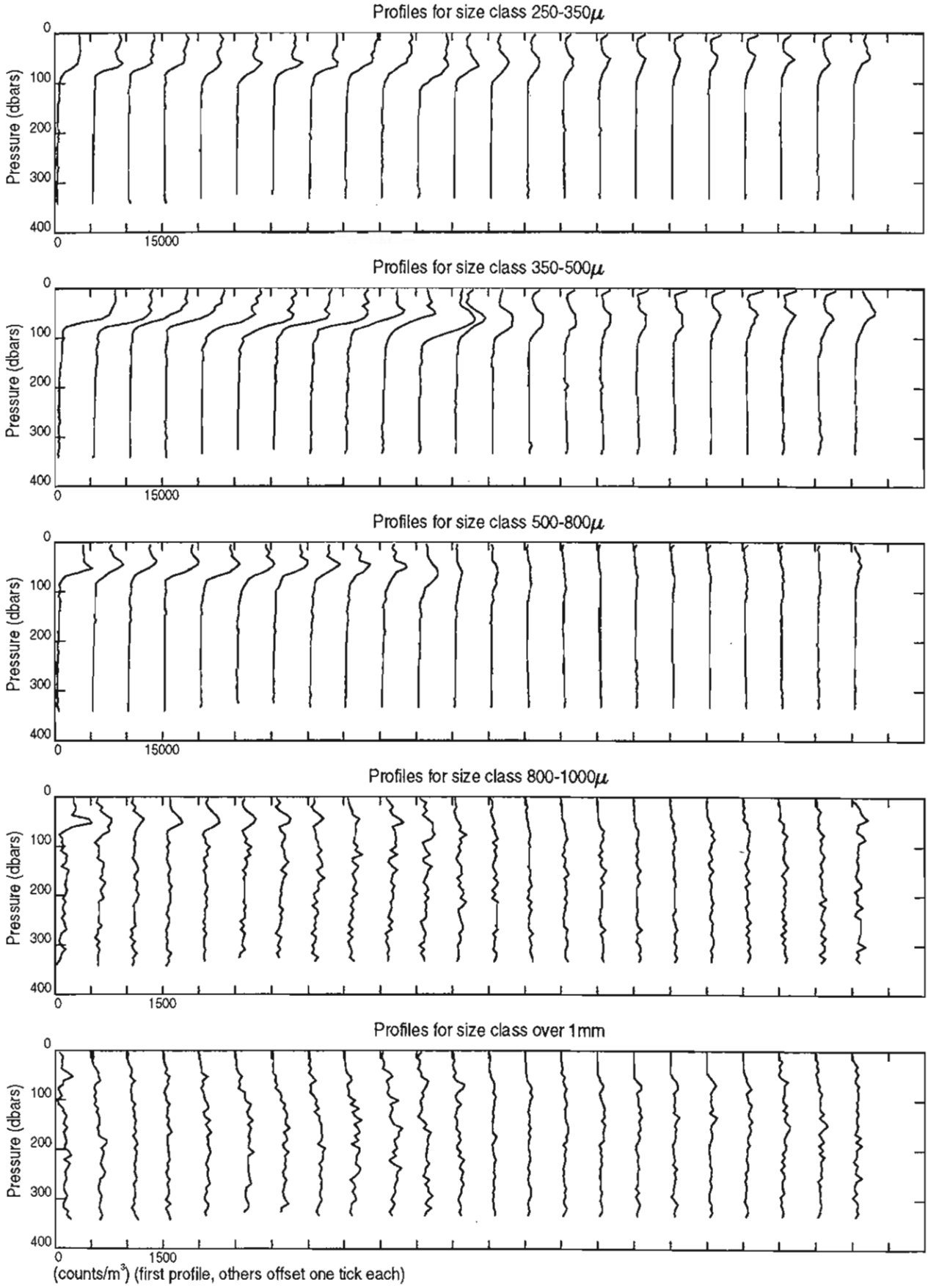
SeaSoar Fine Scale Survey - Run 8.3



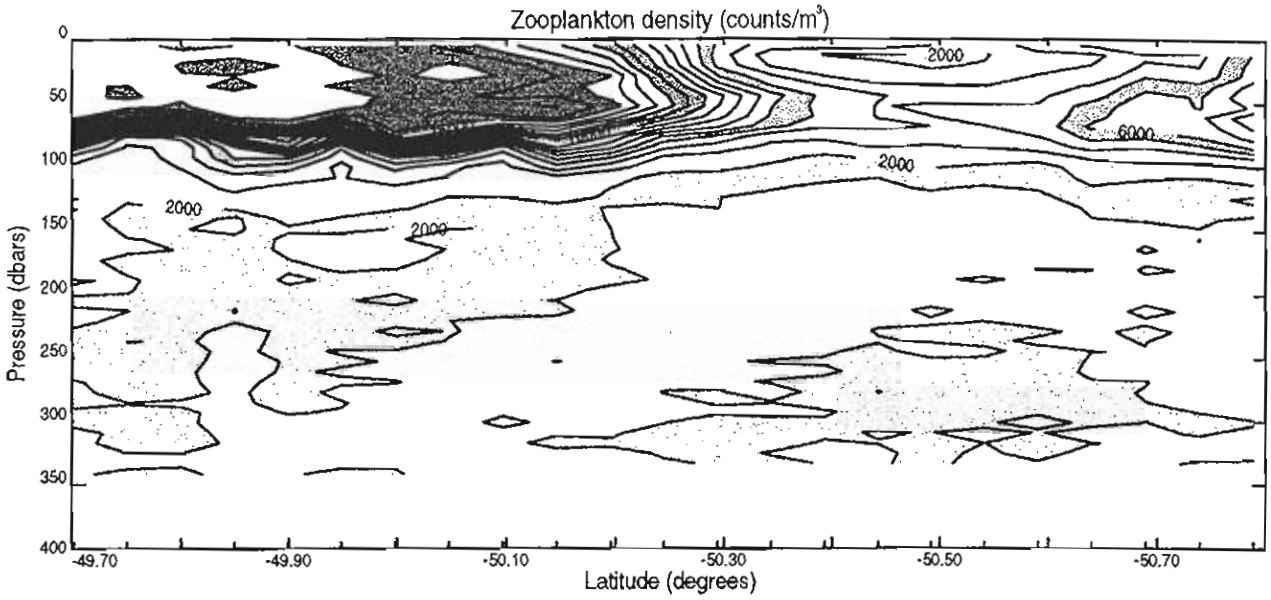


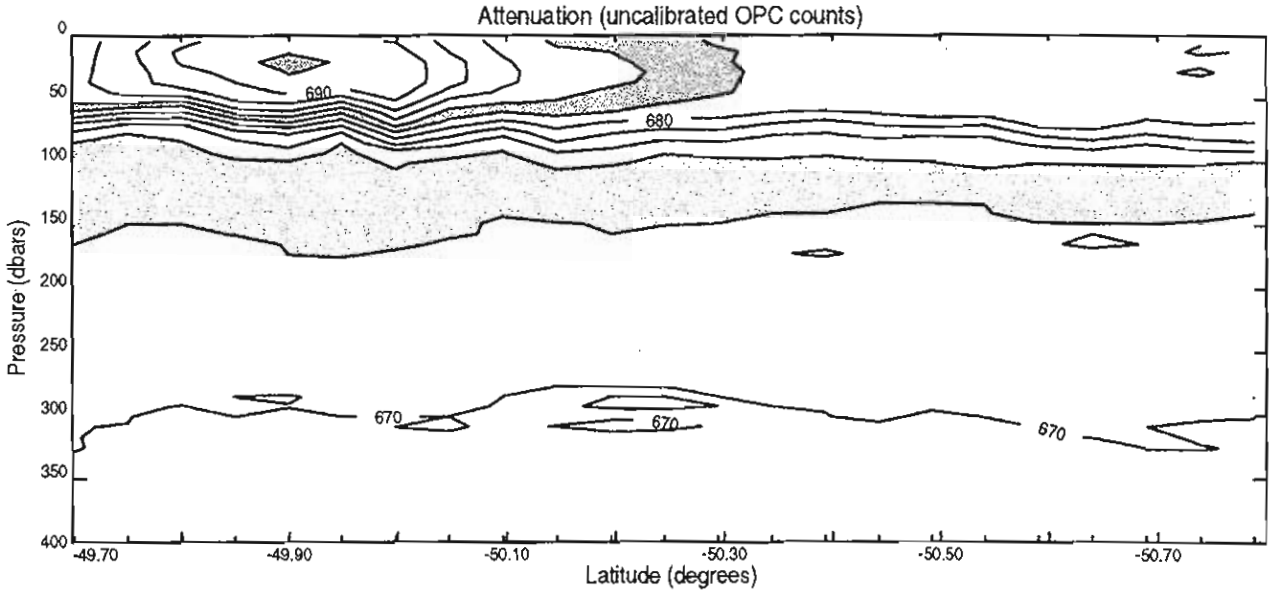


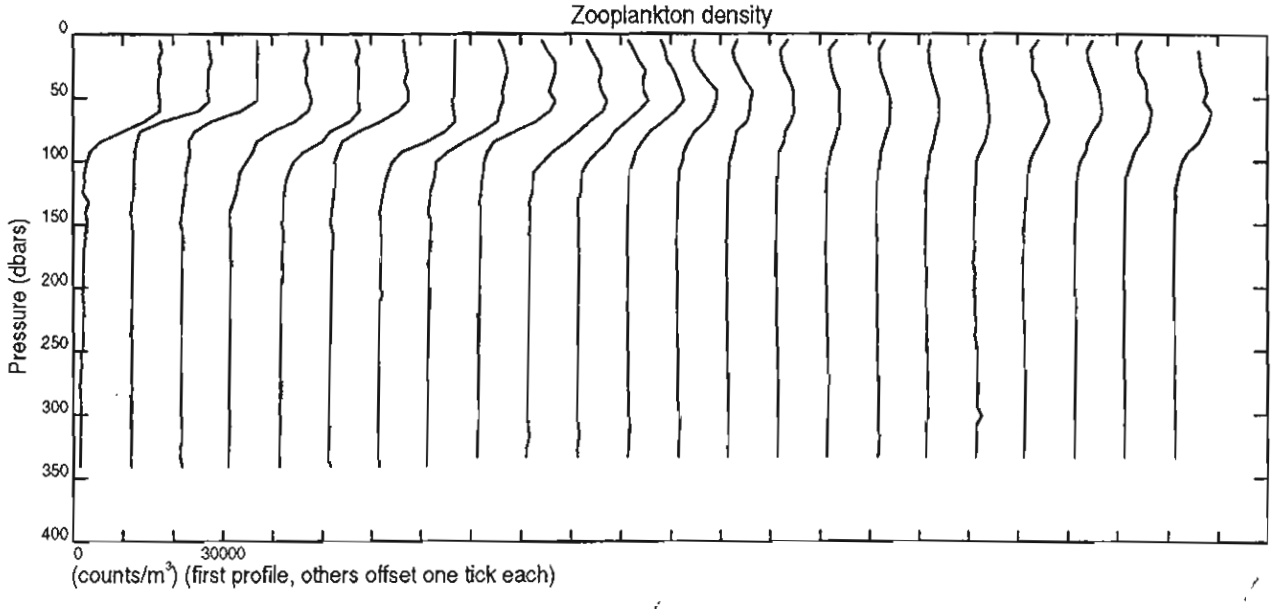
SeaSoar Fine Scale Survey - Run 8.3



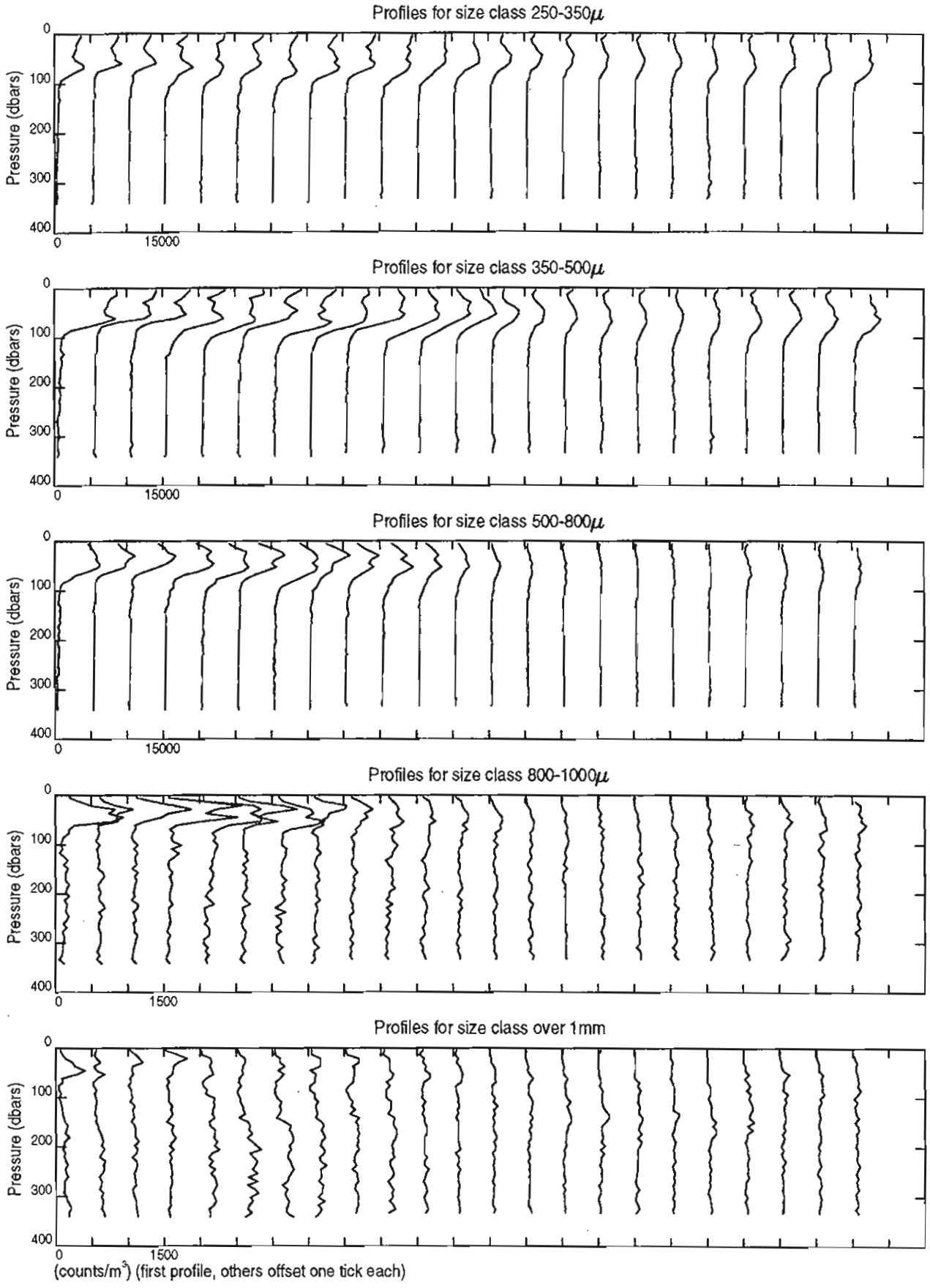
SeaSoar Fine Scale Survey - Run 8.4



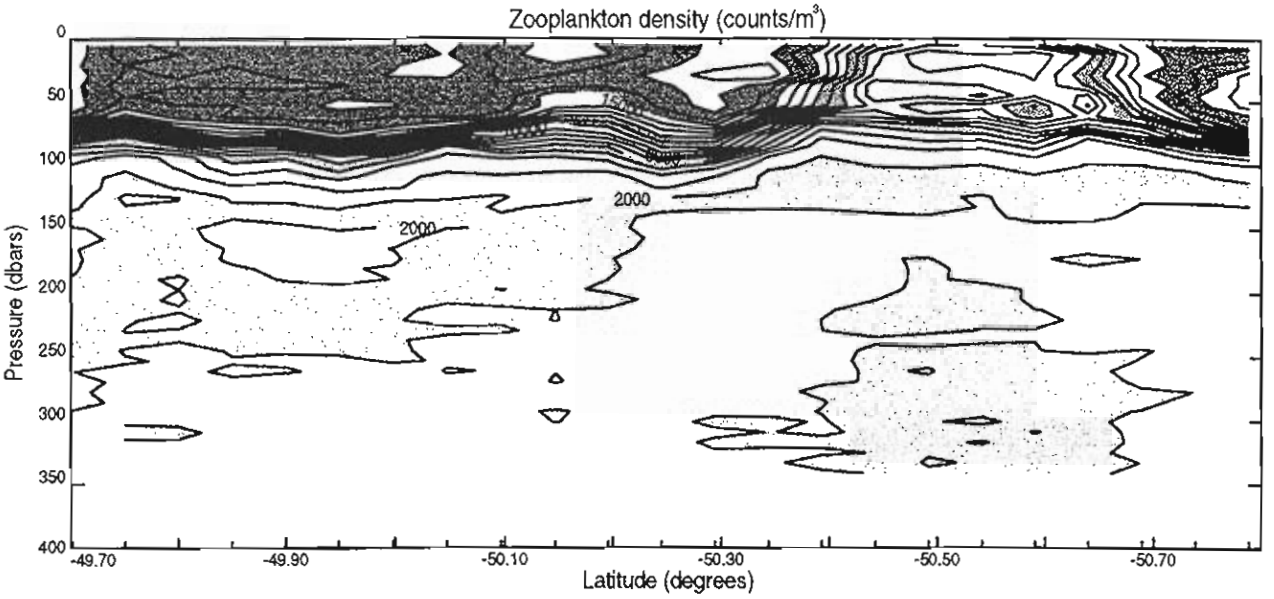


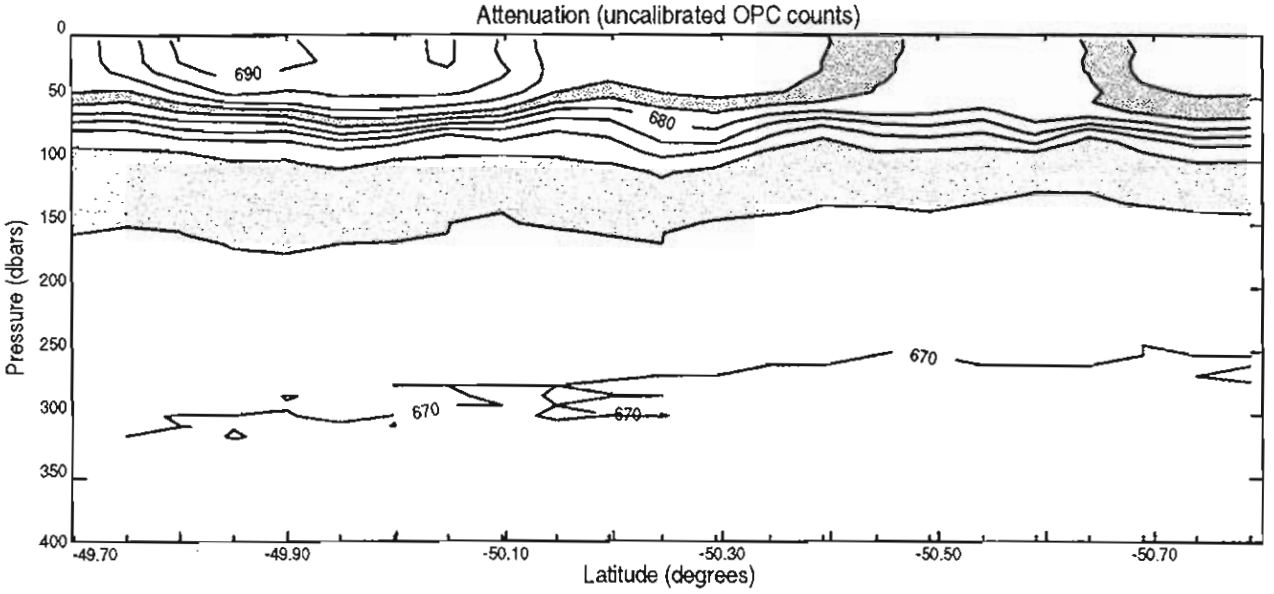


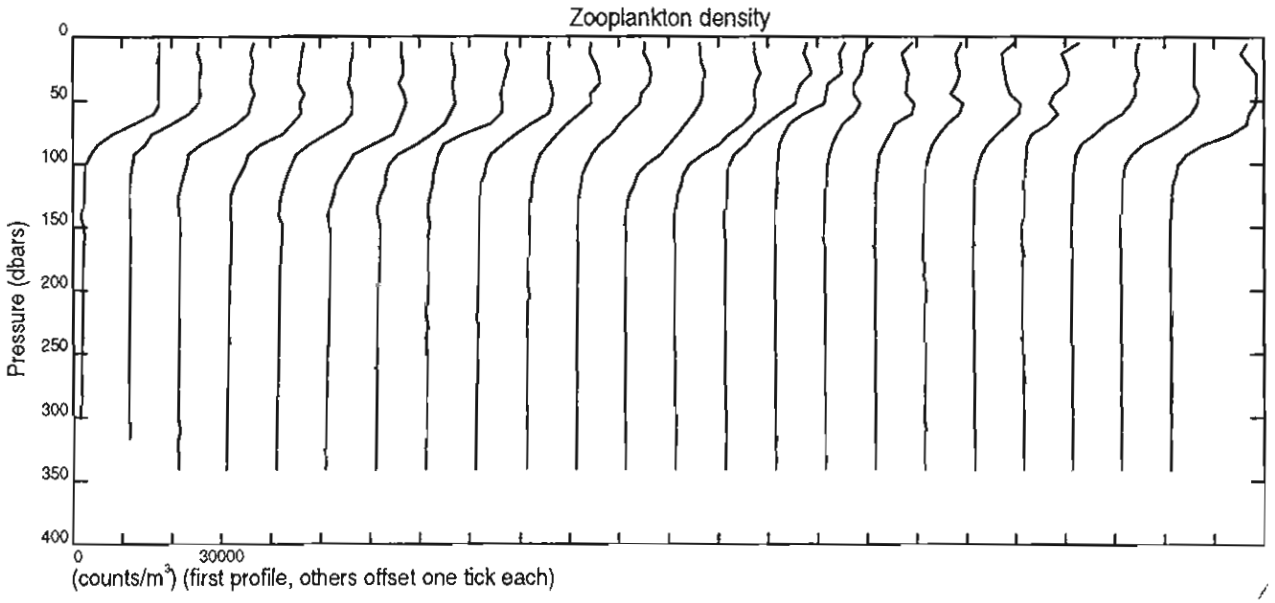
SeaSoar Fine Scale Survey - Run 8.4



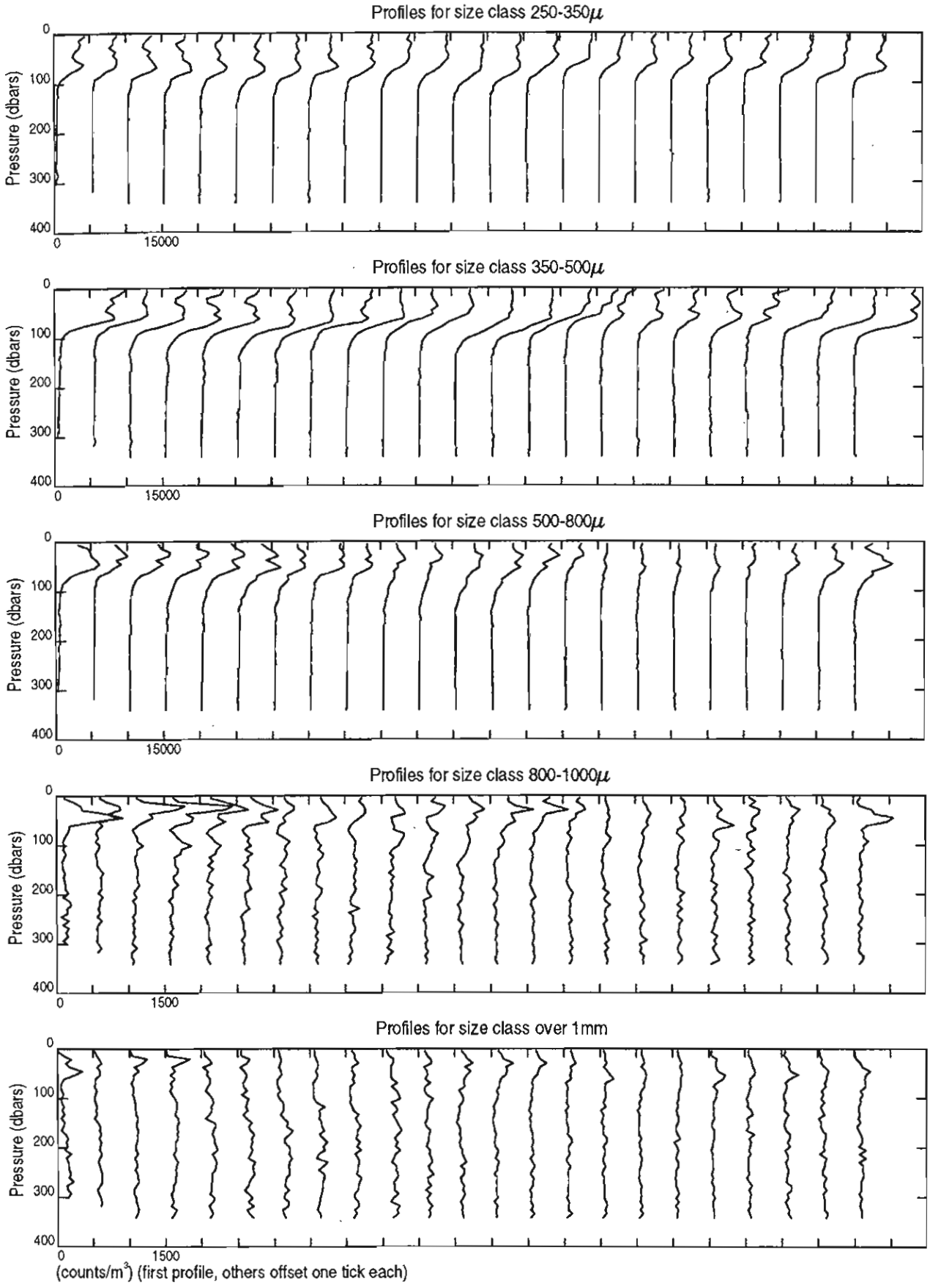
SeaSoar Fine Scale Survey - Run 8.5



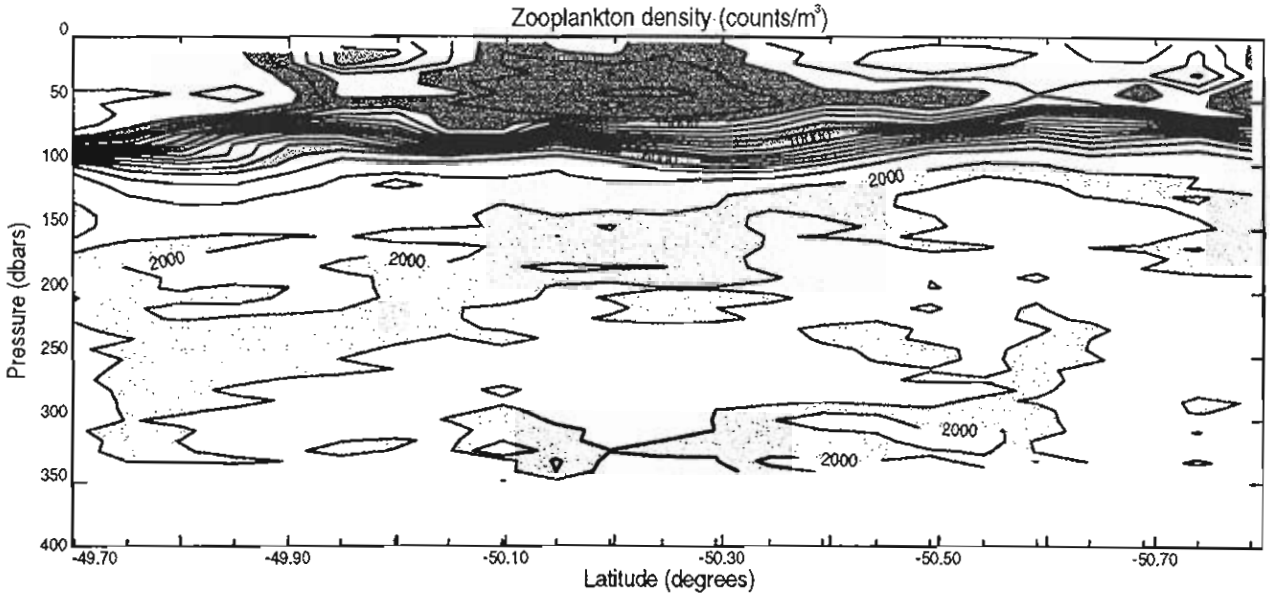


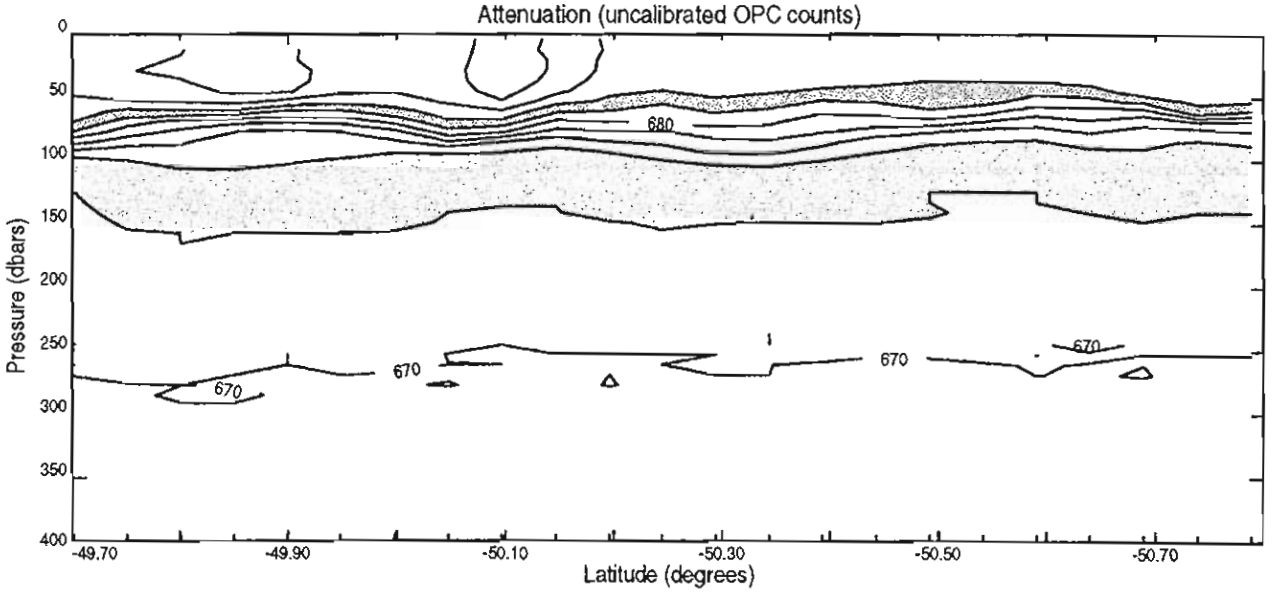


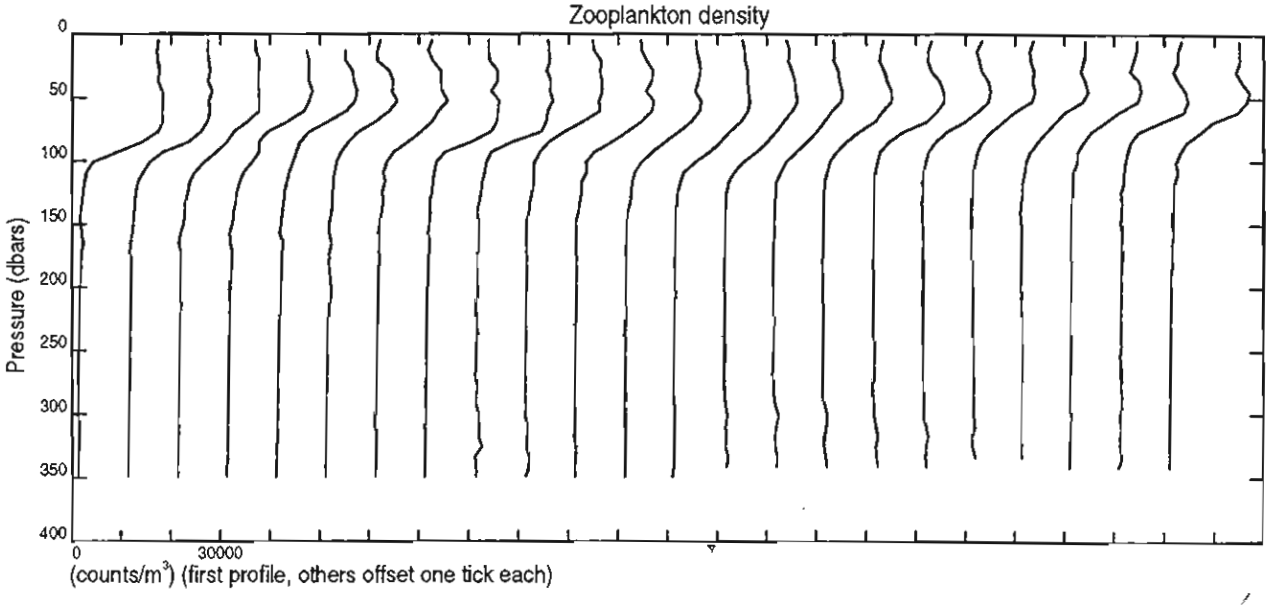
SeaSoar Fine Scale Survey - Run 8.5



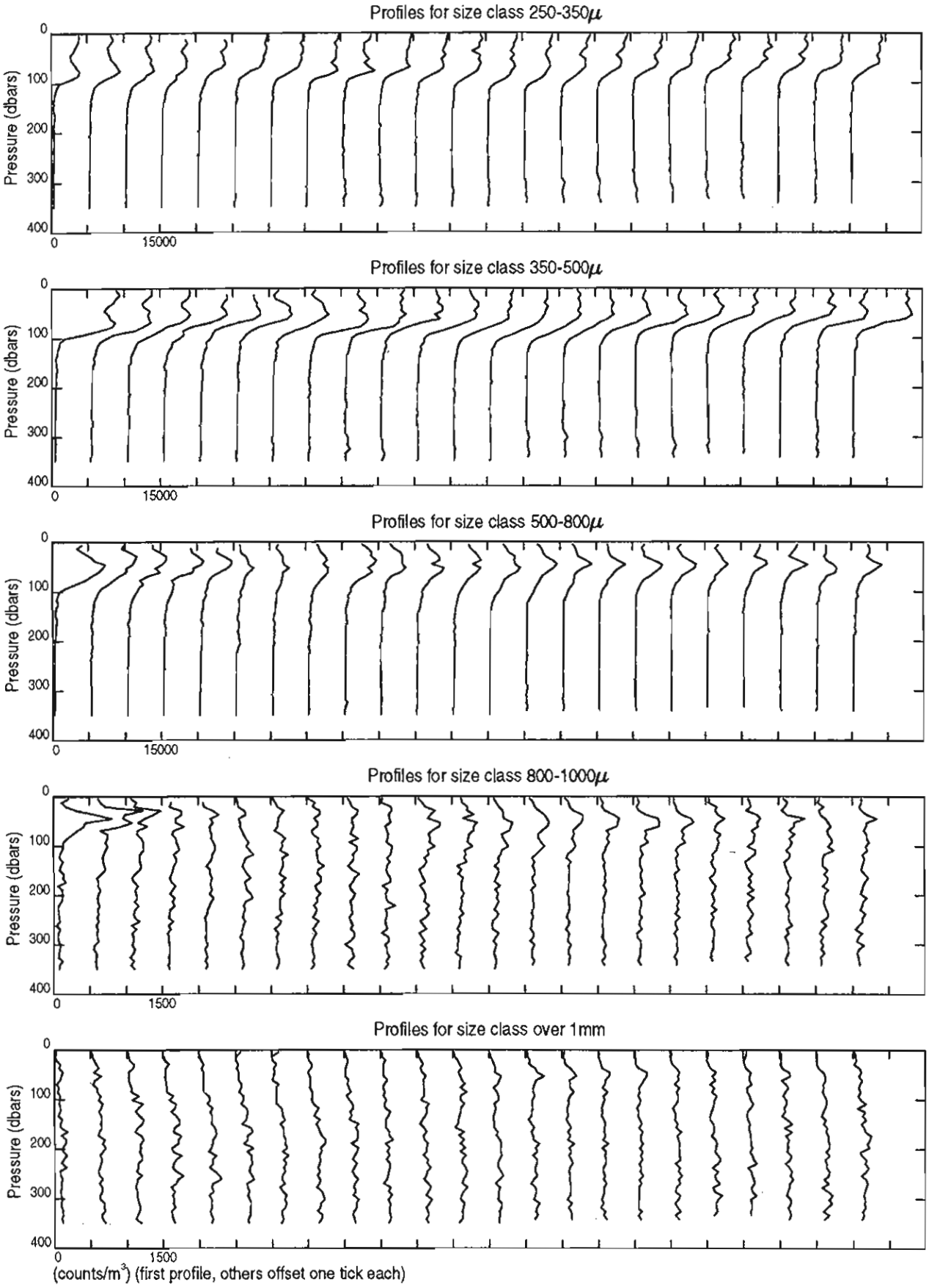
SeaSoar Fine Scale Survey - Run 8.6



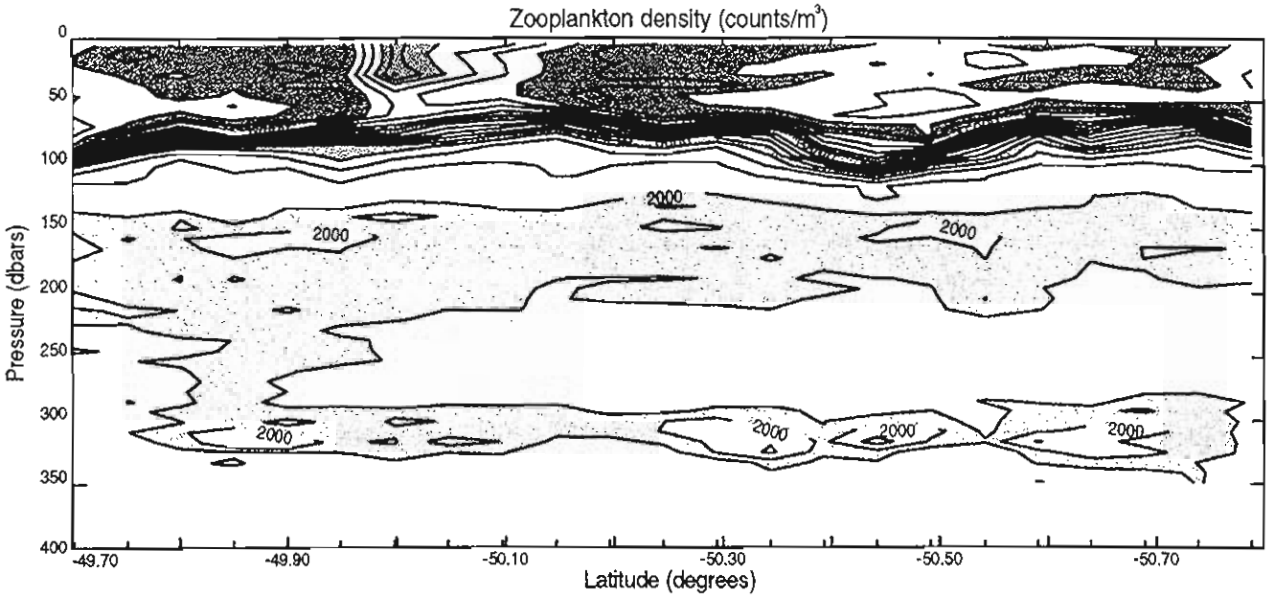


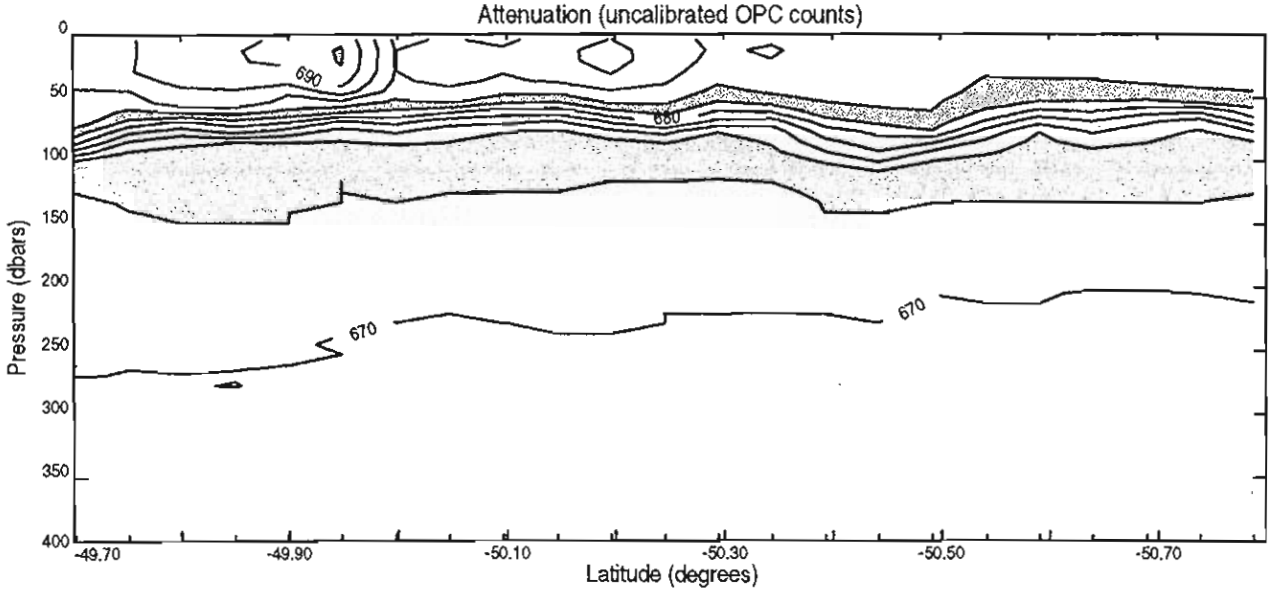


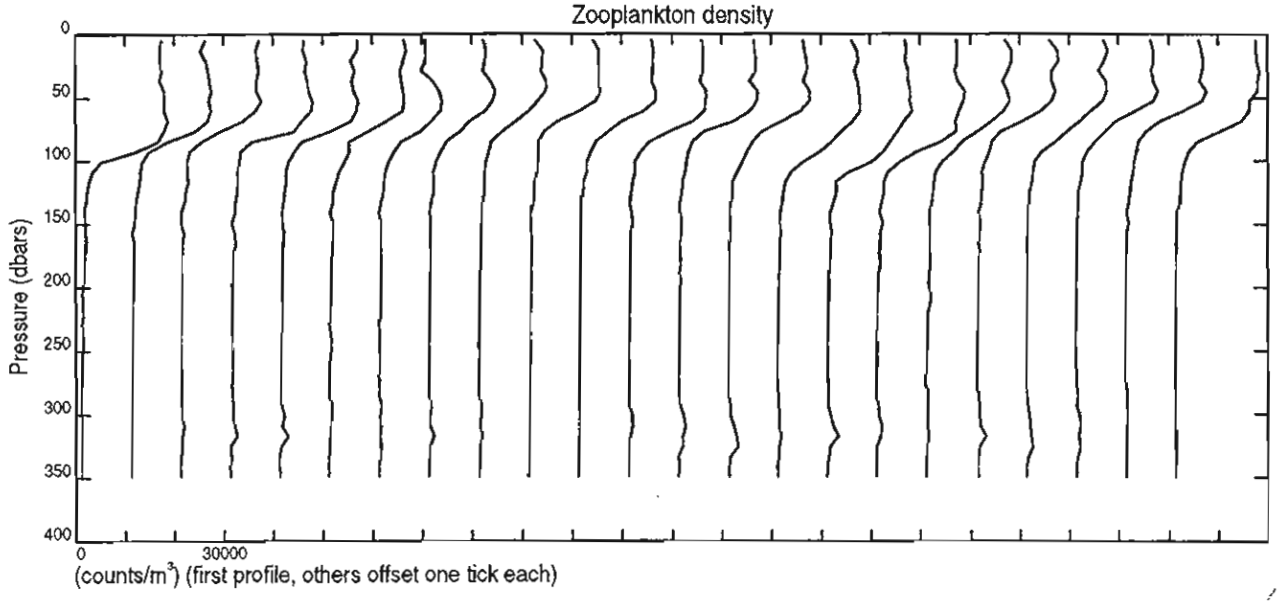
SeaSoar Fine Scale Survey - Run 8.6



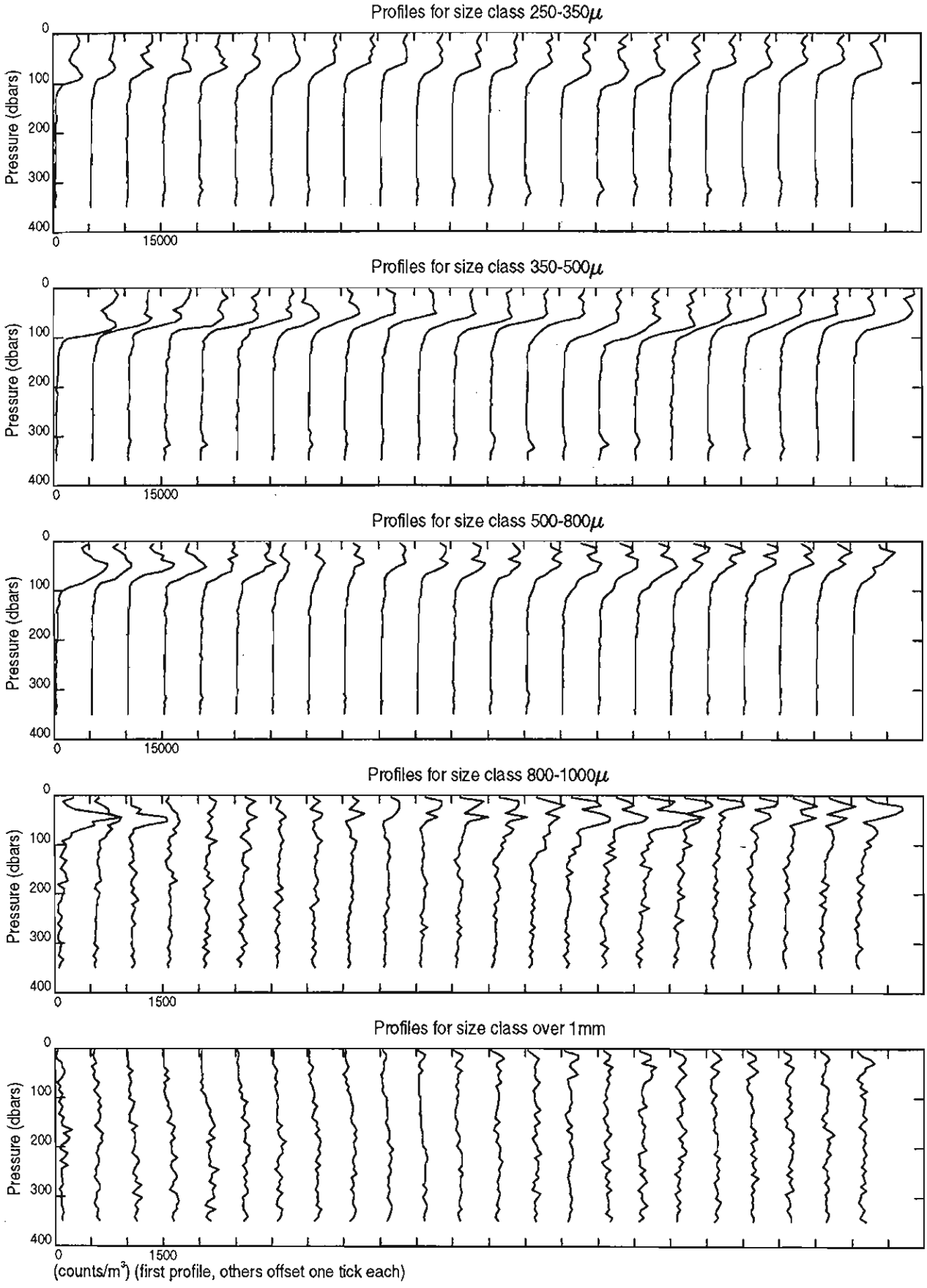
SeaSoar Fine Scale Survey - Run 8.7



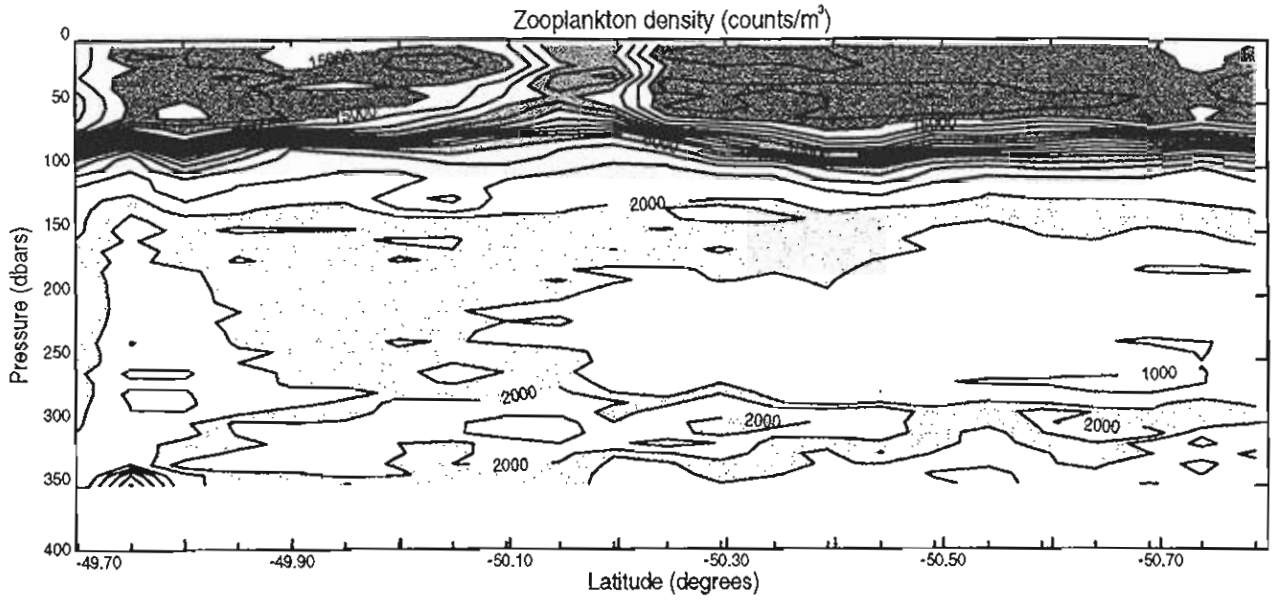


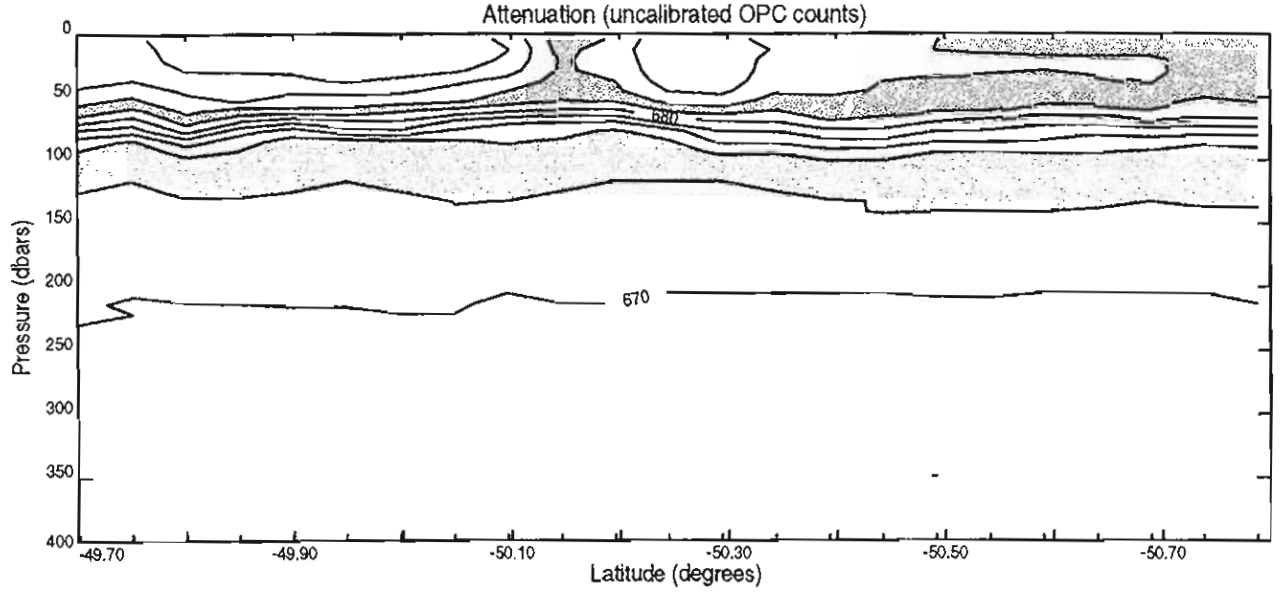


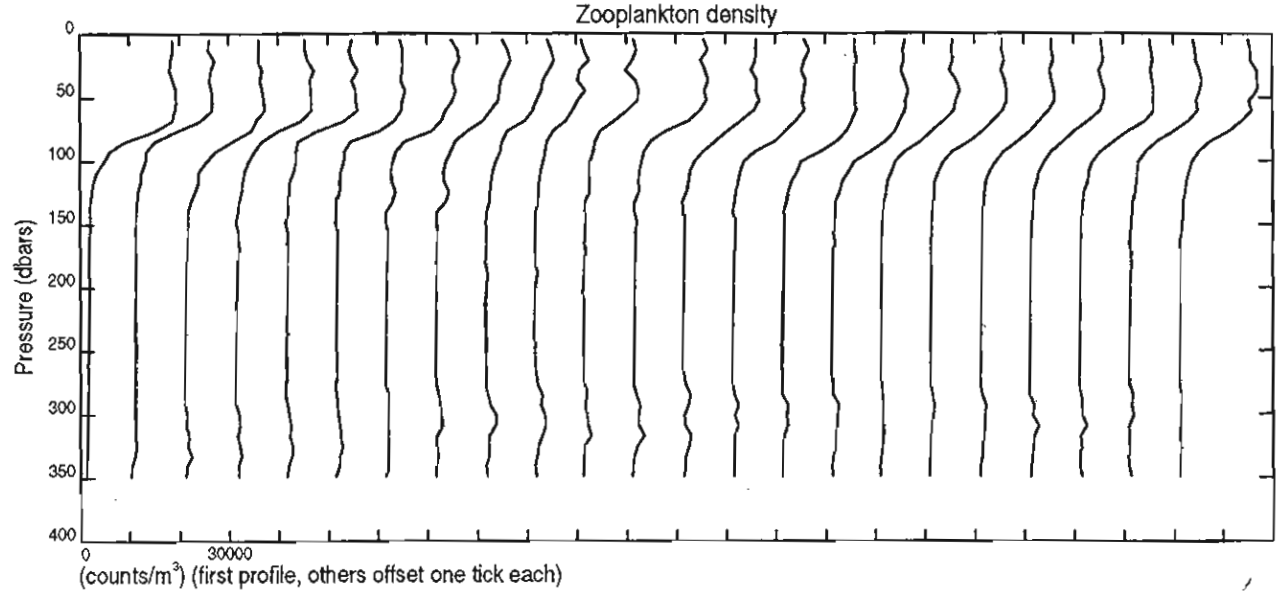
SeaSoar Fine Scale Survey - Run 8.7



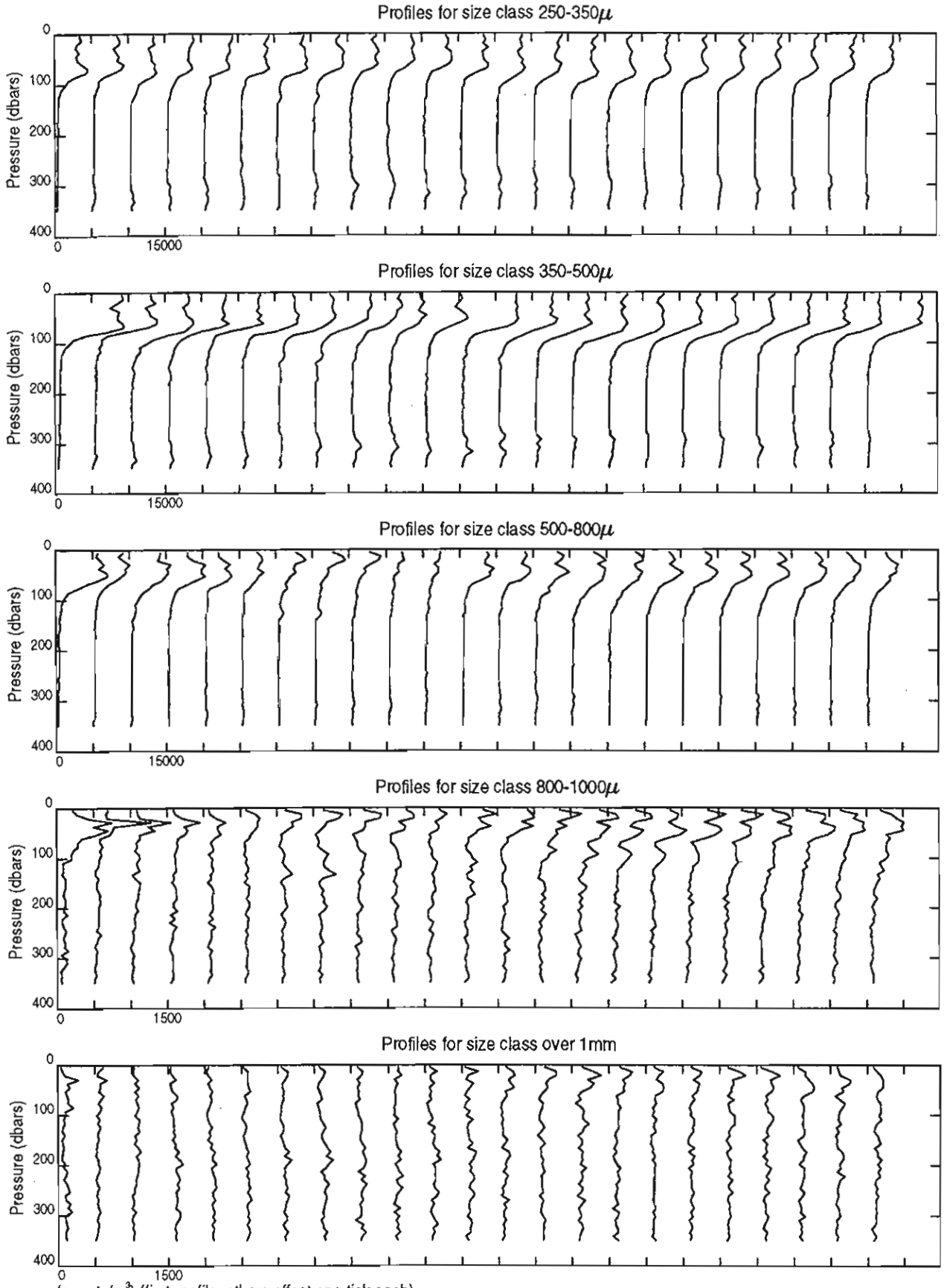
SeaSoar Fine Scale Survey - Run 8.8





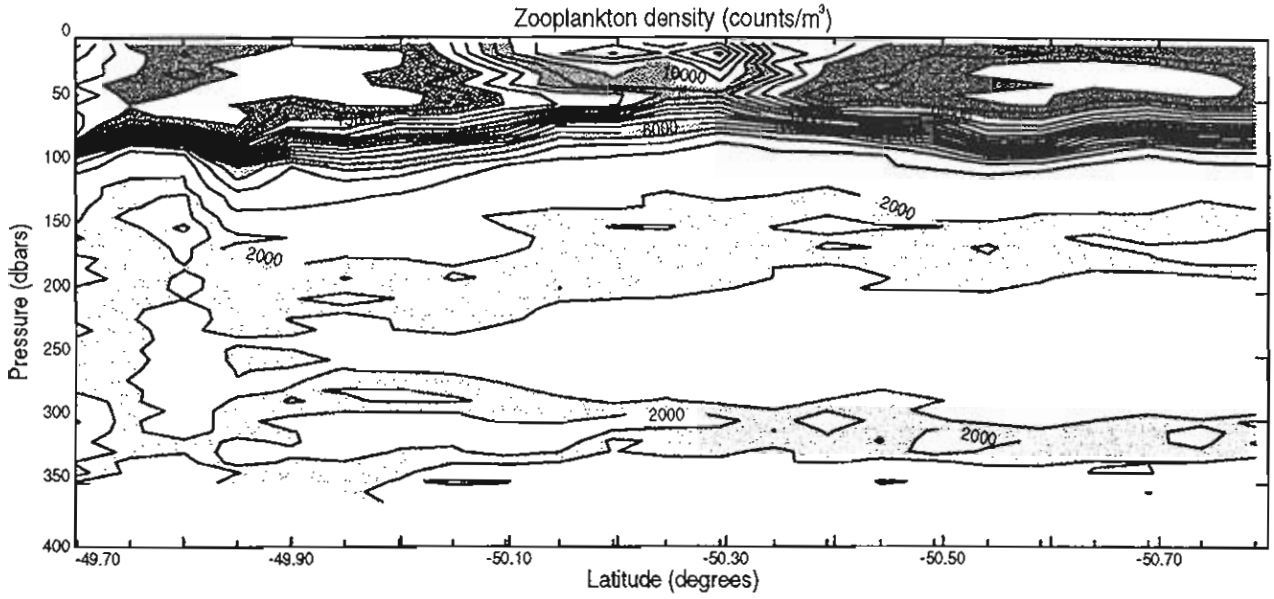


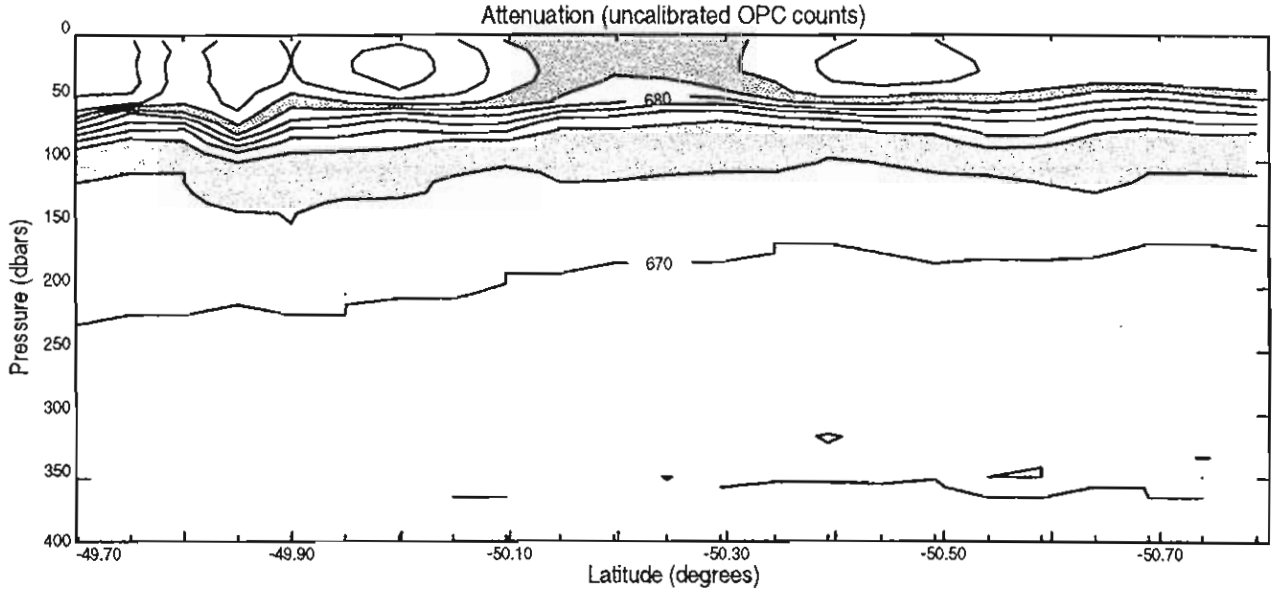
SeaSoar Fine Scale Survey - Run 8.8

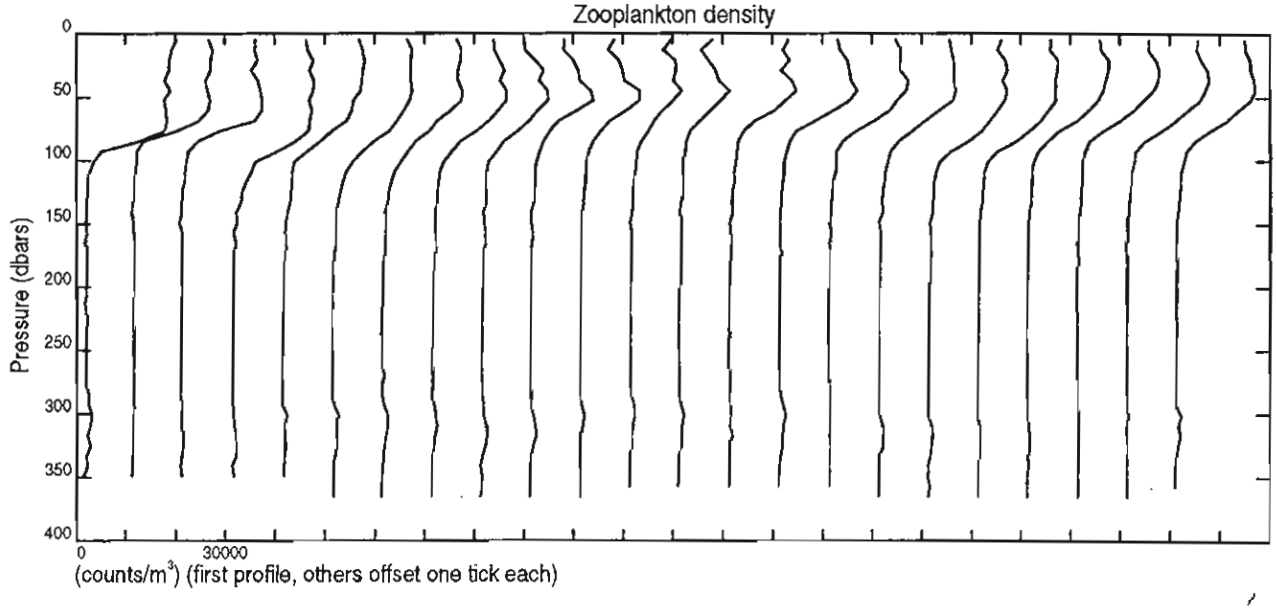


(counts/m^3) (first profile, others offset one tick each)

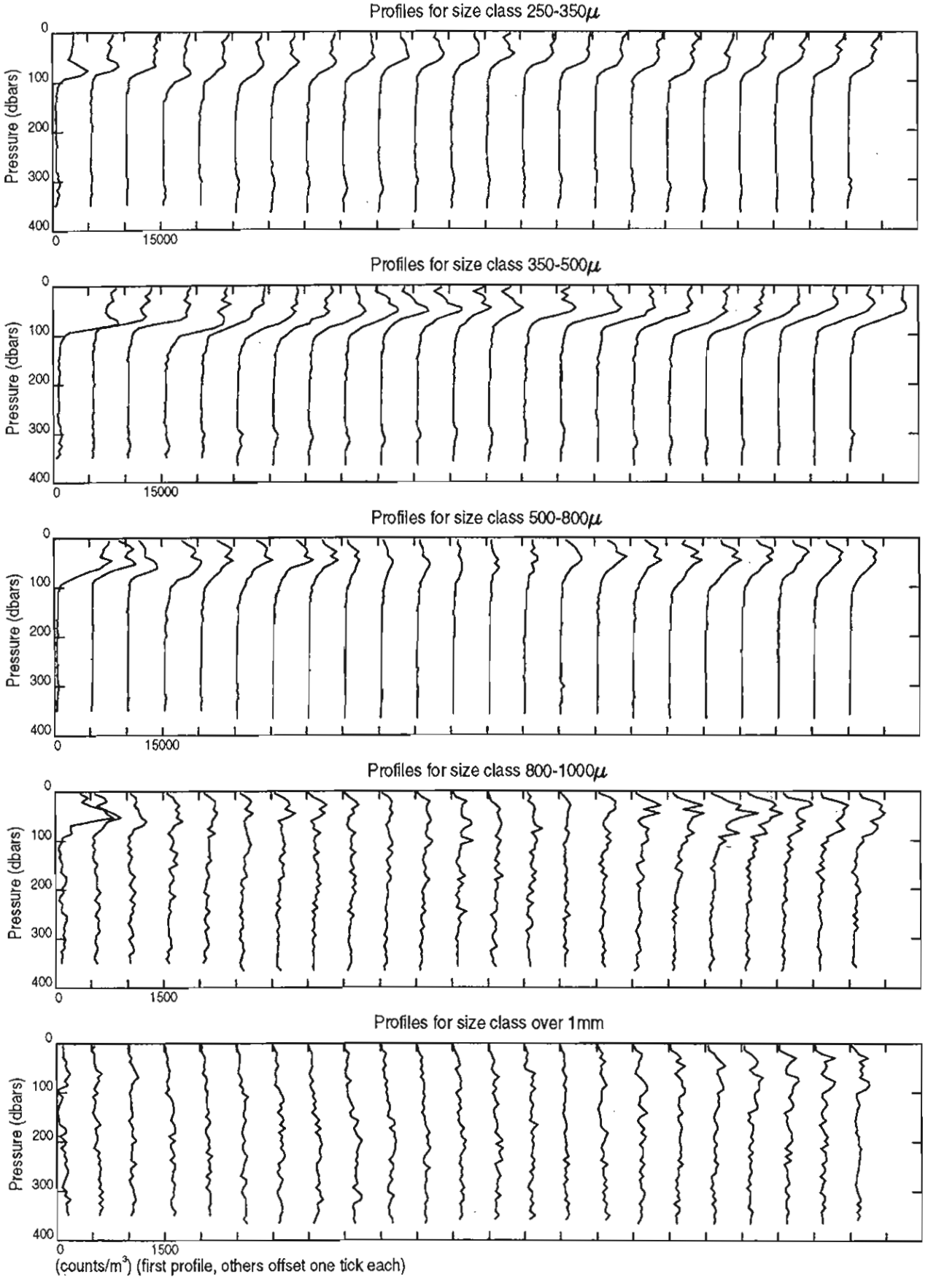
SeaSoar Fine Scale Survey - Run 8.9



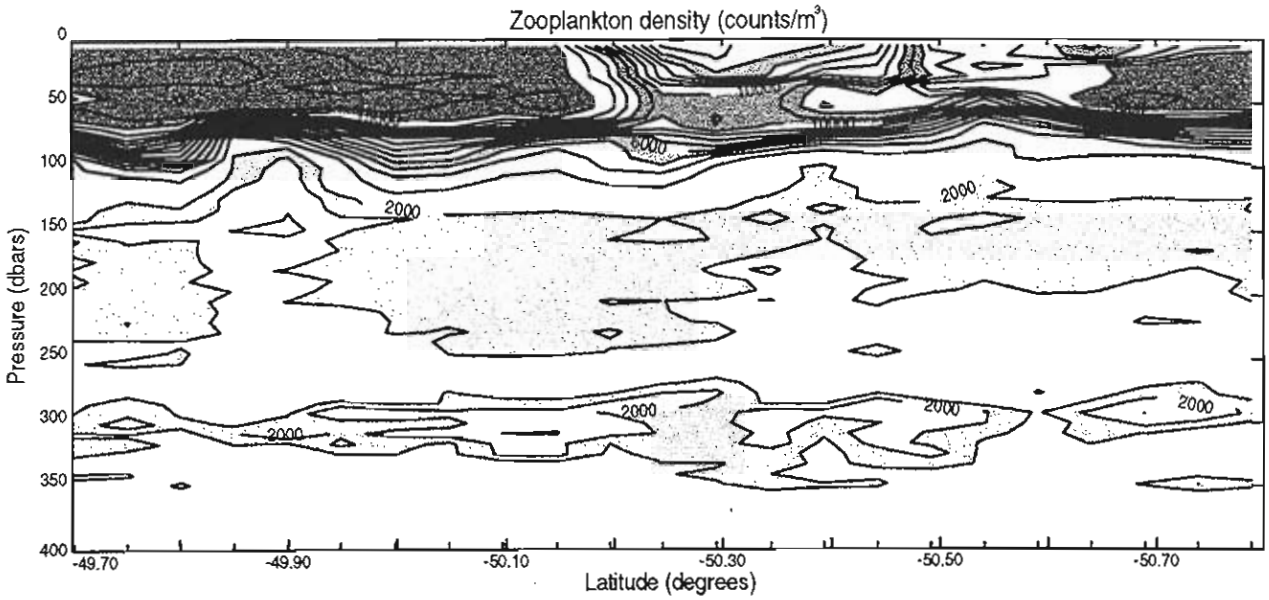


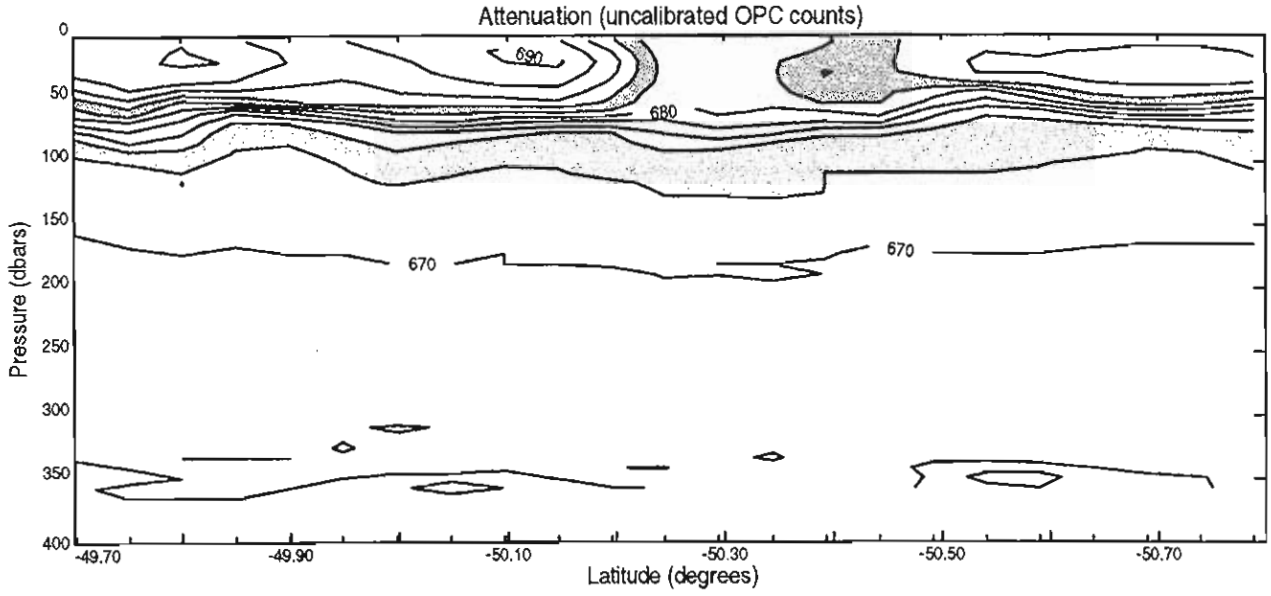


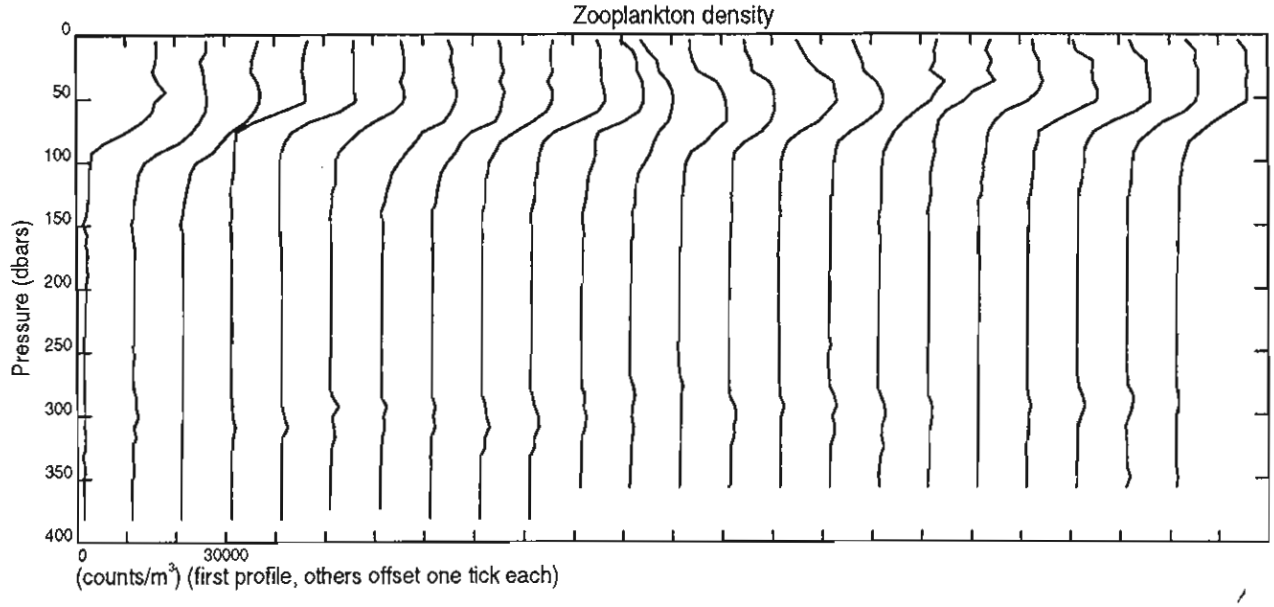
SeaSoar Fine Scale Survey - Run 8.9



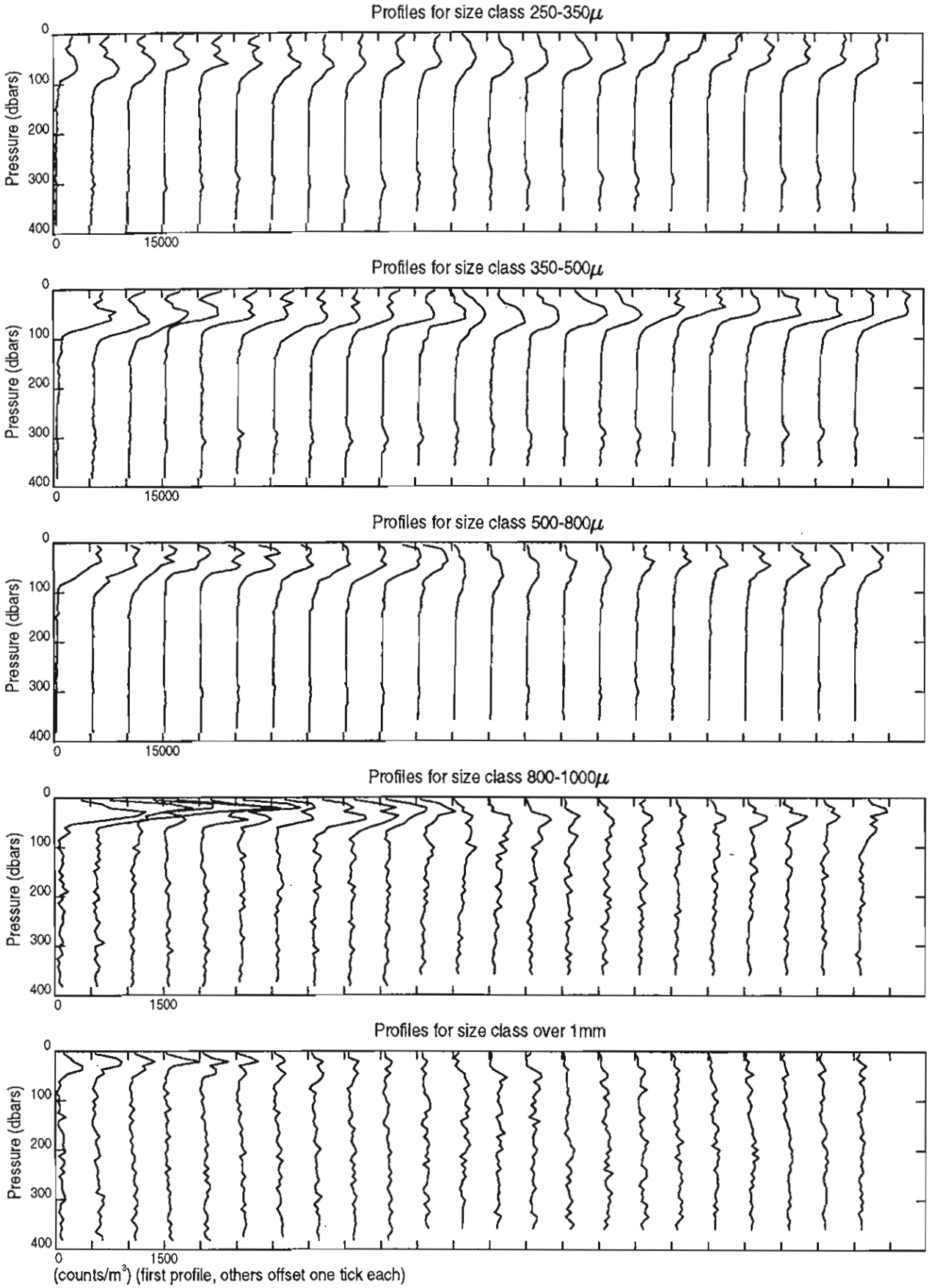
SeaSoar Fine Scale Survey - Run 8.10



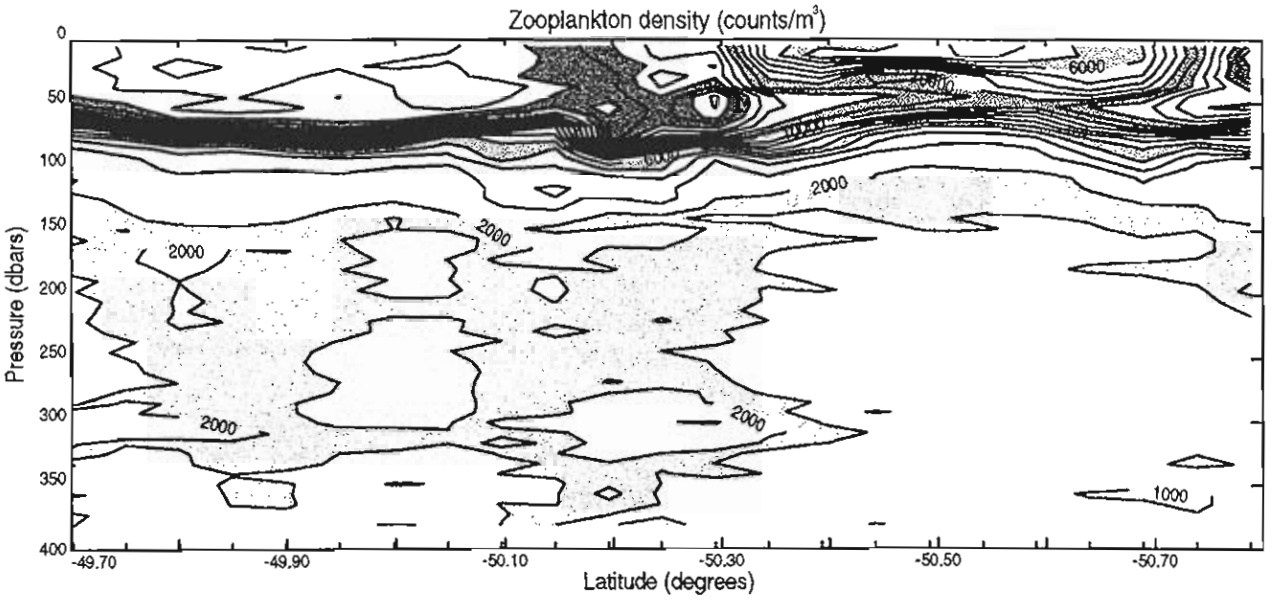


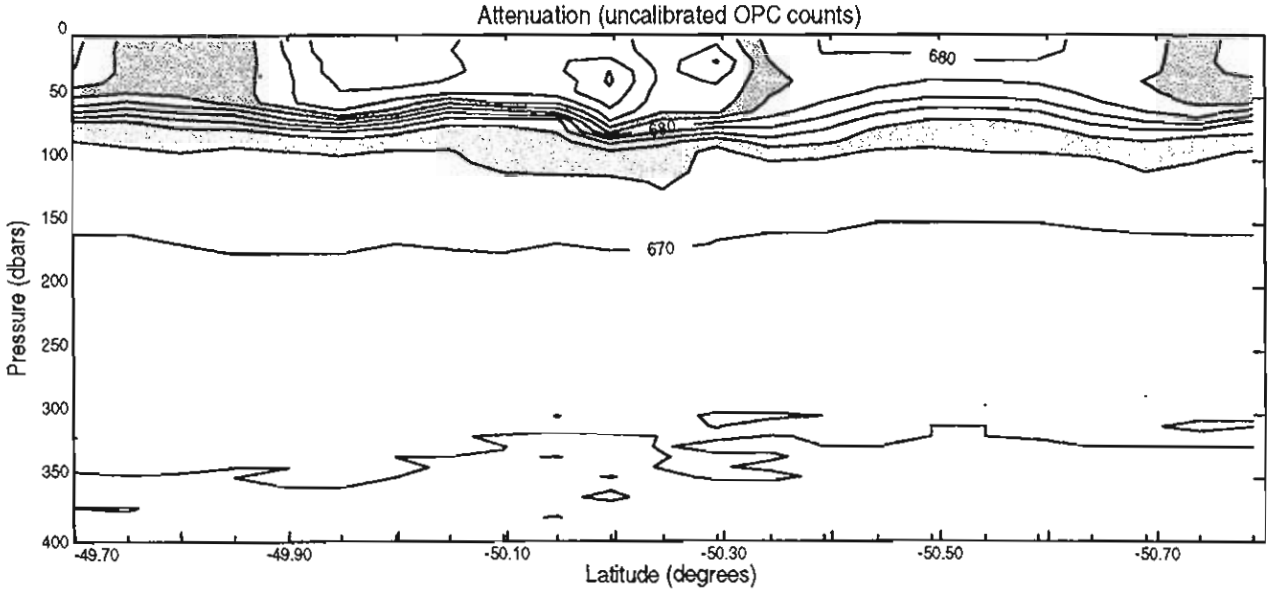


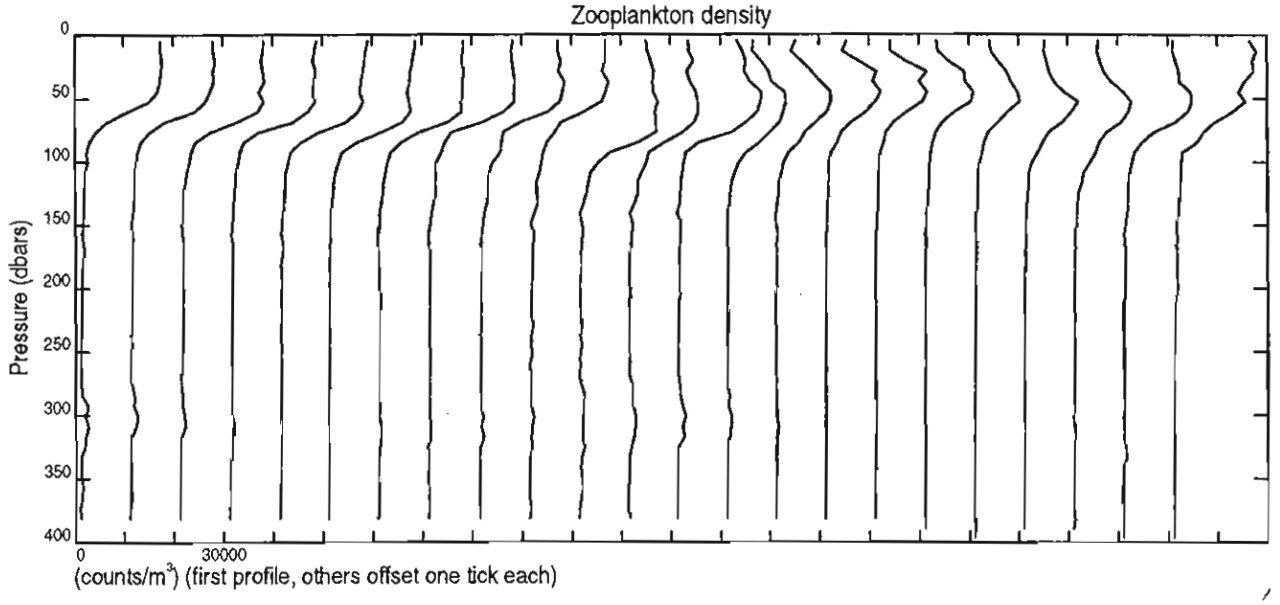
SeaSoar Fine Scale Survey - Run 8.10



SeaSoar Fine Scale Survey - Run 8.11







SeaSoar Fine Scale Survey - Run 8.11