Implementation and evaluation of the Heffter Method to calculate the height of the planetary boundary layer above a southern Great Plains site

Igor Pesenson
November 30th, 2002

Abstract

This paper explores the Heffter Method -- an algorithm for finding the height of the Planetary Boundary Layer (PBL). The algorithm is applied to the Balloon Borne Sounding System (BBSS) data collected over the Southern Great Plains (SGP) Site of the Atmospheric Radiation Measurement (ARM) Program. After discussing the successes and shortcomings of the algorithm, the resulting PBL height estimates for dates in May of 2002 are related to CO_2 concentration and wind data. The CO_2 data used is from the Precision Gas System (PGS) while the wind data is a combination of data from the Portable CO_2 Flux System on the SGP site and BBSS.

1. Introduction

Although all of the troposphere (the lower ~10km of the atmosphere) is effected by surface conditions, most of it has a relatively slow response time. The lower part of the troposphere that is effected on a shorter time scale is commonly defined as the Planetary Boundary Layer (PBL). According to Stull (1), one can describe the planetary boundary layer as "that part of the troposphere that is directly influenced by the presence of the earth's surface, and responds to surface forcings with a timescale of about an hour or less."

Surface temperature has a strong relationship with height of the PBL. As the surface cycles between daytime radiation and nighttime cooling the amount of convection taking place changes. When the temperature gradient is steep, more convection takes place to dissipate thermal energy in the most efficient manner. In other words, the greater the temperature difference between the surface and the lower troposphere, the higher convective eddies must reach to alleviate the gradient. Relating this to Stull's definition of turbulence, it can be concluded that the height of the PBL varies with surface temperature. In fact, the spatial range of the PBL can vary from less than one hundred meters to several kilometers.

The strong relationship between convective turbulence and height of the PBL is sometimes used to define the boundary layer (1) and call it the Convective Boundary Layer (CBL). Analogous to the Stull's definition but focusing on turbulence, Lloyd et all (3) describe CBL as "a layer of air typically of order 1km in depth, well mixed by turbulence maintained by buoyancy due to heating at the ground. It is bounded above by stably stratified, nonturbulent air and grows through the day".

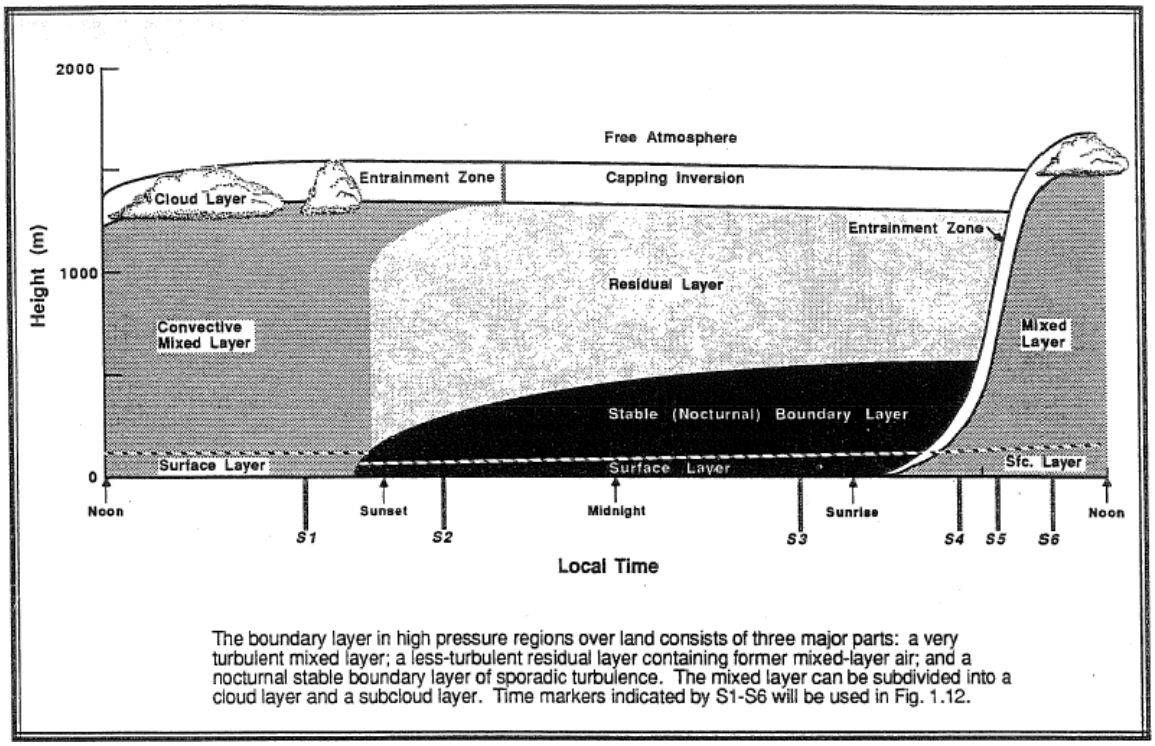

Figure 1. Taken from R. B. Stull, An Introduction to Boundary Layer Meteorology.

The PBL is also broken down into several layers (Figure 1). The lowest part of the PBL is the Surface Boundary Layer and it is directly in contact with the ground. It is characterized by "strong vertical gradients in temperature, humidity, wind and scalars" and it is roughly 10% of the PBL (1,2). Immediately above the Surface Boundary Layer is the Mixing Layer (ML). This

layer is dominated by convective mixing and any existing wind is almost constant in profile (1) (Figure 2). The top of the ML, as described by Lloyd, is bounded by stratified wind shear. This "layer at the top of the ML acts as a lid to the rising thermals, thus restraining the domain of turbulence" (1). This is called the Entrainment Zone and an absolute temperature inversion is frequently present here. Because of this, it can be also be referred to as the Inversion Layer.

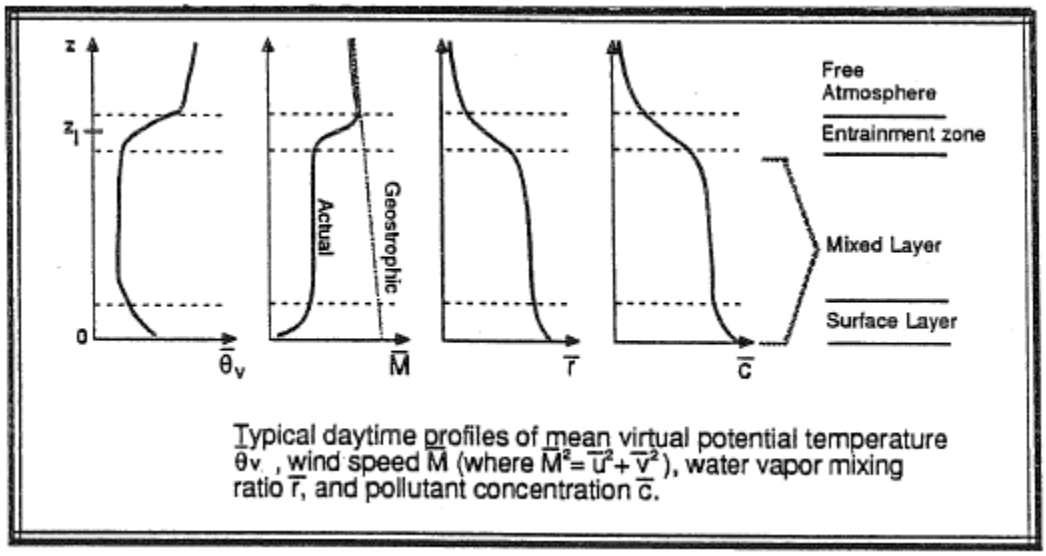

Figure 2. Taken from R. B. Stull, An Introduction to Boundary Layer Meteorology.

2. Heffter Method

The Heffter Method relies on the existence of the Inversion Layer described above. It consists of two mathematical conditions that attempt to identify a "critical inversion" in the temperature profile. Although "inversion" in the term "Inversion Layer" refers to increase of absolute temperature with height, the technique uses potential temperature to find the top of the PBL. The two conditions are

$$\Delta\theta / \Delta z > 0.005 \ \ ^\circ\text{K m}^{-1} \qquad (1)$$

$$\theta_t - \theta_b > 2 \ \ ^\circ\text{K} \qquad (2)$$

where $\Delta\theta / \Delta z$ is the potential temperature lapse rate in the inversion layer and θ_t, θ_b refer to the potential temperatures at the top and bottom of the critical inversion layer, respectively.

The first condition (Eq.1) looks for a significant change in potential temperature. Potential temperature θ is the temperature a packet of air would have if it were brought down to sea level.

$$\theta = T \left(P_0/P \right)^{R/(mCp)} \qquad (3)$$

T is the dry air temperature, P_0 is the standard pressure, and R/(mCp) evaluates to .286 . Potential temperature is more or less constant inside the mixed layer due to adiabatic mixing. According to the Ideal Gas Law

$$PV=nRT \qquad (4)$$

the decrease in pressure with height leads to the increase in volume, which, in turn, decreases the absolute temperature of the air packet. Potential temperature (Eq. 3) takes the effect of pressure out so as to let us to focus on the energy of an air packet. As the height increases and the entrainment zone is entered, the air is no longer well mixed, the adiabatic assumption is no longer as effective, and so potential temperature starts increasing with height (Figure 4).

The second condition (Eq.2) attempts to ensure the height of the PBL is set at the top of the Inversion Layer, not merely the top of the Mixing Layer (Figure 2).

2.1 Algorithm Implementation

The Heffter algorithm was implemented in C programming language and (see Appendix for full text of program). The code stepped through the data set and checked the validity of Equation (1). As soon as Equation (1) was satisfied, equations (1) and (2) would be checked for each step through the data. If both were simultaneously true, the PBL height for the data set was output. If condition (1) became, at first, satisfied but then failed before (2) was satisfied, the program continued to check for condition (1) at the current height and above.

It was found that this straightforward implementation of the method failed rather frequently. Many of the data sets had trouble satisfying exactly these conditions, despite coming close. It became clear that one or both of the conditions had to be somewhat more flexible. Condition two (Eq. 2) was chosen as the one to be iteratively relaxed on algorithm failure below 3000 meters. There was no reason for why condition (2) should have been picked over conidition (1), the iterative relaxation could have been applied to either. And so the code, upon having failed to satisfy both conditions on all data below 3000 meters, iteratively lowered the 2.0 °K condition by 0.1 °K until both Eq.1 and modified Eq.2 were satisfied.

2.2 Algorithm Performance

For each sonde balloon launch both absolute humidity and potential temperature were plotted against height. Absolute humidity decreases slowly within the Mixed Layer and the top is marked by a sharp moisture decrease. This fact is often used together with potential temperature to identify top of PBL (1). The performance of the Heffter Method was evaluated via visual inspection of plots of theta and absolute humidity for each BBSS data set.

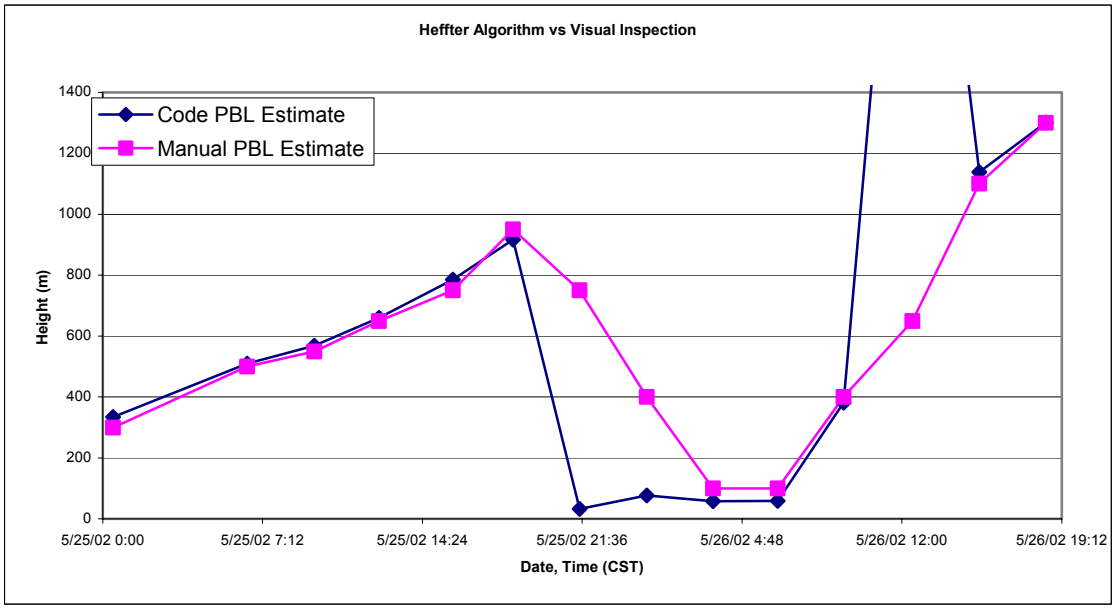

Figure 3. The dark line is the evolution of the PBL as estimated by the slightly modified Heffter Method. The lighter line is the evolution of the PBL as determined by individually inspecting each data set. Note that manual PBL height on May 26th at 03:29 AM and 06:24 AM (CST) was artificially placed at 100m due lack of definition in the theta and humidity plots for this time.

The algorithm performed quite well under reasonable weather conditions. Figure 3 shows fourteen balloon launches in two days. Out of the fourteen PBL estimates, the program was significantly off three times. The first two points the program stumbled upon, were May 25th and May 26th at 21:29 and 00:30 local time (CST), respectively. These two points, as well as the two following, occurred during nighttime when the PBL is not well defined. Figure 5 shows the profiles of theta and absolute humidity for May 25th, 21:29 local time. The graph shows a well-defined mixed layer, much as in Figure 4, however, the base of the graph had already started to change. Ground cooling had begun to alter the lower Surface Boundary Layer. The code correctly found the Heffter conditions to be satisfied at 33m and never bothered to check that most of the PBL was still in tact this early in the evening. According to Figure 1 this well-defined layer could be considered part of the Residual Zone and so not part of the PBL. It is unclear exactly what can be called the PBL in this case and visual analysis, relying on the water profile, placed it close to 400 meters. The same problem was responsible for the May 26th, 00:30 difference between calculated and estimated values. It can be concluded that one shortcoming of the algorithm might be underestimation of the PBL in the evening, when there is a strong residual layer.

The following two graphs of potential temperature and absolute humidity, on May 26th at 03:29 and 06:24, demonstrate a different issue. Figure 6 shows the lack of definition for the 03:29 (CST). The Heffter conditions were satisfied at 58 meters, and this is where the program placed the PBL. As a convenient minimum, the visual analysis placed the PBL at 100m under such ambiguous circumstances.

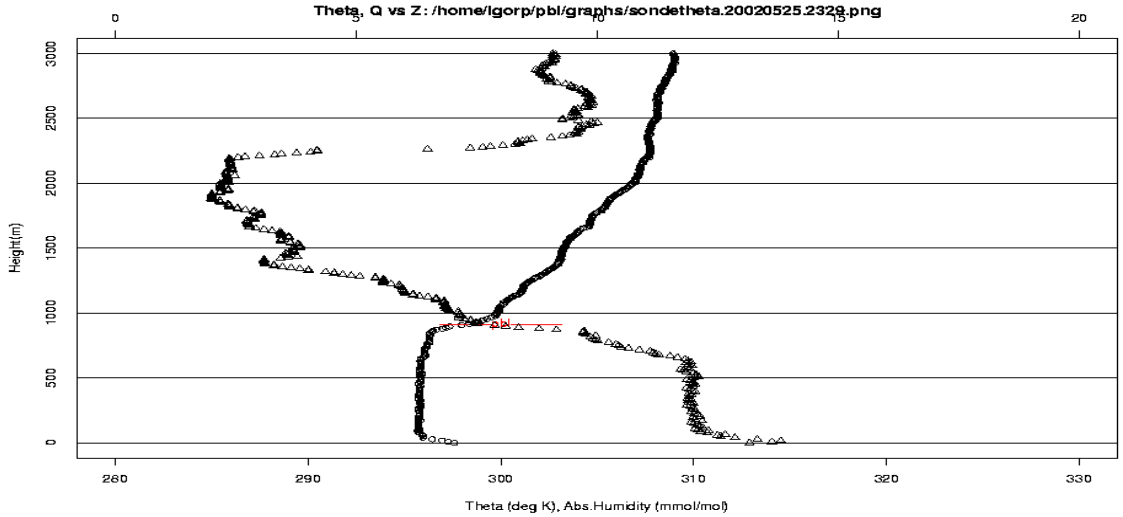

Figure 4. Potential temperature and absolute humidity at 06:30PM local time (CST) on May 25th. The surface boundary layer, the mixed layer, and the inversion layer are all well defined.

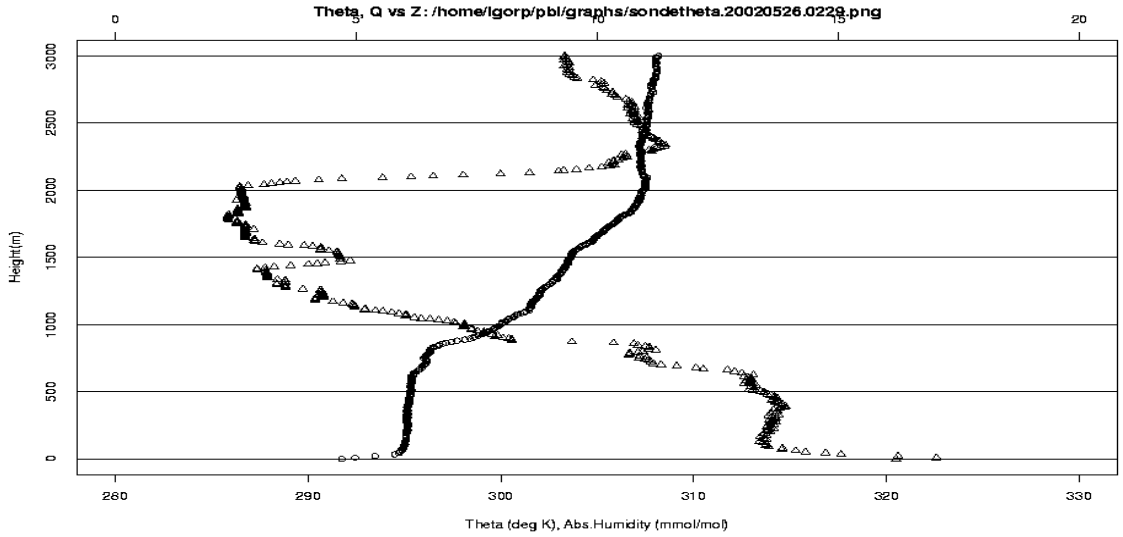

Figure 5. There is a strong residual layer up to 9:29PM local time (CST) on May 25th. The temperature profile reversal is seen in the first 100m, but not above. This makes it tempting to place the PBL height at around 750m, despite the change in water profile close to 400m. Visual analysis settled on 400m as the PBL height.

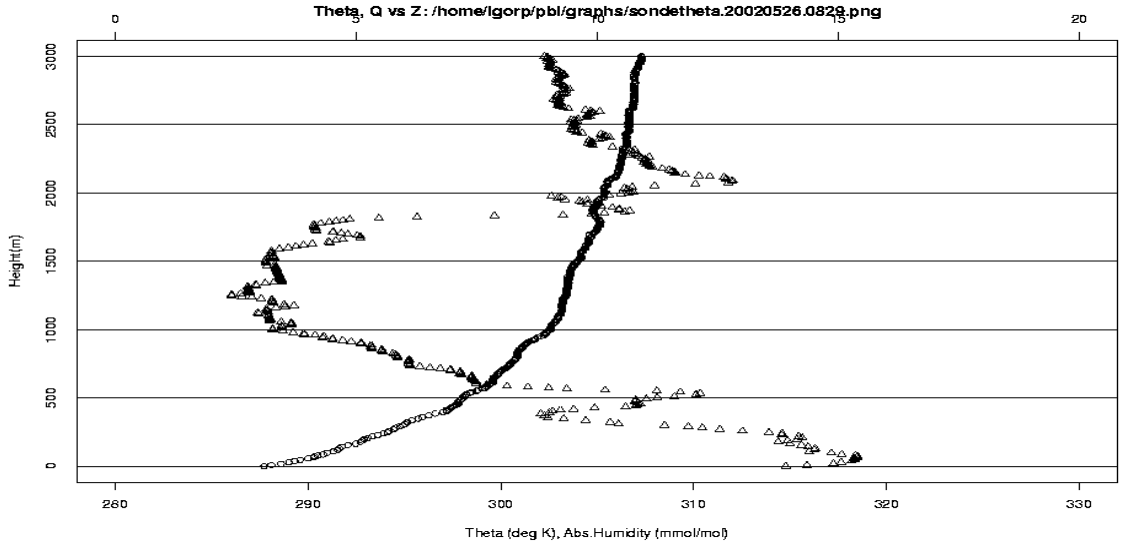

Figure 6. Potential temperature profile lacks definition and the Heffter method finds PBL to be at 58m.

The May 26th, 03:29 and 06:24 (CST) points reflect the ambiguity of the PBL late into the night. It is badly defined so that even visual analysis cannot place the PBL better than the algorithm. However, even if the PBL is badly defined at such times, the program still calculates a height with consistent conditions. Although this might not be identified as the height of the PBL, it is still arguably a useful point due to the consistency with which it is calculated.

3. Correlation of PBL and CO2

Precision Gas System is a data collection system mounted on a sixty-meter tower located at the Cloud and Radiation Testbed (CART) of the ARM Southern Great Plains Site (SGP). The sensors are mounted at 2m, 4m, 25m, and 60m and CO2 concentration is measured as well as

some other variables. Analysis of the relationship between CO_2 measurements and PBL height demonstrates some predictable, yet interesting, results.

Strong convective mixing during the day raises the height of the PBL up to 1.8 kilometers. Daytime radiation that is responsible for increase in convective activity also stimulates photosynthesis. This causes the CO_2 concentrations to drop at sunrise and decrease through to sunset (Figure 6). During this time there seems to be little discrepancy between concentrations at 2 and 60 meters and so both heights can be assumed to be in the well-mixed layer.

At sunset, the surface temperature drops and convection is no longer the dominant energy transport mechanism. Photosynthesis has stopped and CO_2 begins accumulating due to ground respiration. The surface boundary layer is the first to be affected and a temperature inversion can be detected there early in the evening (Figures 4 and 5). Figure 7 also points out that 2 meter CO_2 response time to sunset is much quicker than at 60 meters. The atmosphere is no longer well mixed by convection and the concentration gradients become steeper. As the evening progresses, the PBL height drops to much lower levels, often leaving a residual layer higher up (Figure 1).

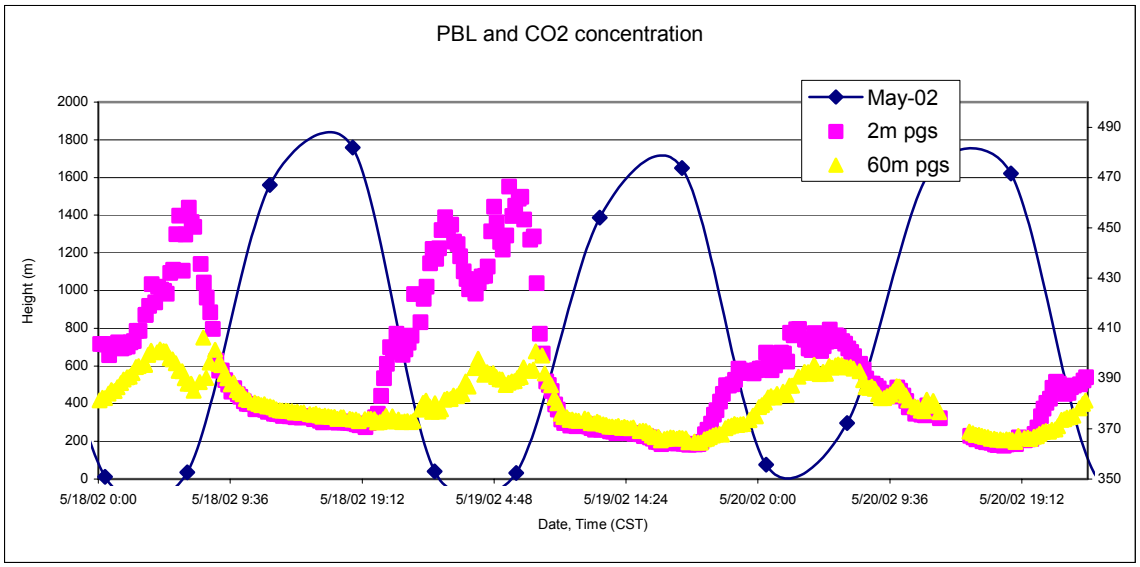

Figure 7. There is a strong nighttime decoupling between 2m and 60m CO2 concentrations.

Another issue behind the increase in nighttime CO_2 concentrations is the rectifier effect (5). One can think of the PBL as the top of a "box" of air. This box has a certain concentration of some gas. During the nighttime the top of the box, or the height of the PBL, will lower. Ignoring any photosynthesis or ecosystem respiration effects, the concentration of the gas will increase following the Ideal Gas Law. Although the PBL is not a perfectly sealed box, and the situation is affected by photosynthesis and ecosystem respiration, the effect should still be noticed on trace gas concentrations. The nighttime PBL height on May 20th is one to two hundred meters higher than that for previous night of May 19th (Figure 6). The coupling between 2m and 60m is clearly much stronger during night of May 20th than either May 18th or 19th. As there could not have been any photosynthesis on either night, the change in readings must have originated from different atmospheric conditions. It is not certain whether a higher PBL was solely responsible for lower concentrations on this night, but the two are correlated.

4. Wind and PBL

Since the height of the PBL depends on air temperature, it is logical to postulate a relationship between wind and PBL. The vertical velocity and temperature of wind could affect

the air temperature and pattern of mixing. The wind temperature should be related to its general direction: warmer wind should generally come from the south.

Wind speeds are much higher during the day and die down at night. This is, of course, attributed to temperature gradients established by daytime radiation. By this reasoning, higher daytime PBL should correspond to faster winds, as both are caused by high heat gradients. This is roughly the relationship observed in Figure 8. However, if the horizontal length scale of the temperature gradients is larger than that of the BBSS data set, the relationship can be expected to be more complex.

The data under scrutiny points to this later case of a complex relationship. For example, the PBL height close to 00:00 on May 20th is 100-200 meters higher than that on May 19th. However, the wind speeds for both nights are comparable. This is in contrast to the night of May 18th when a much higher gas concentration at 2m seems to correlate well with almost zero wind speed at 4m.

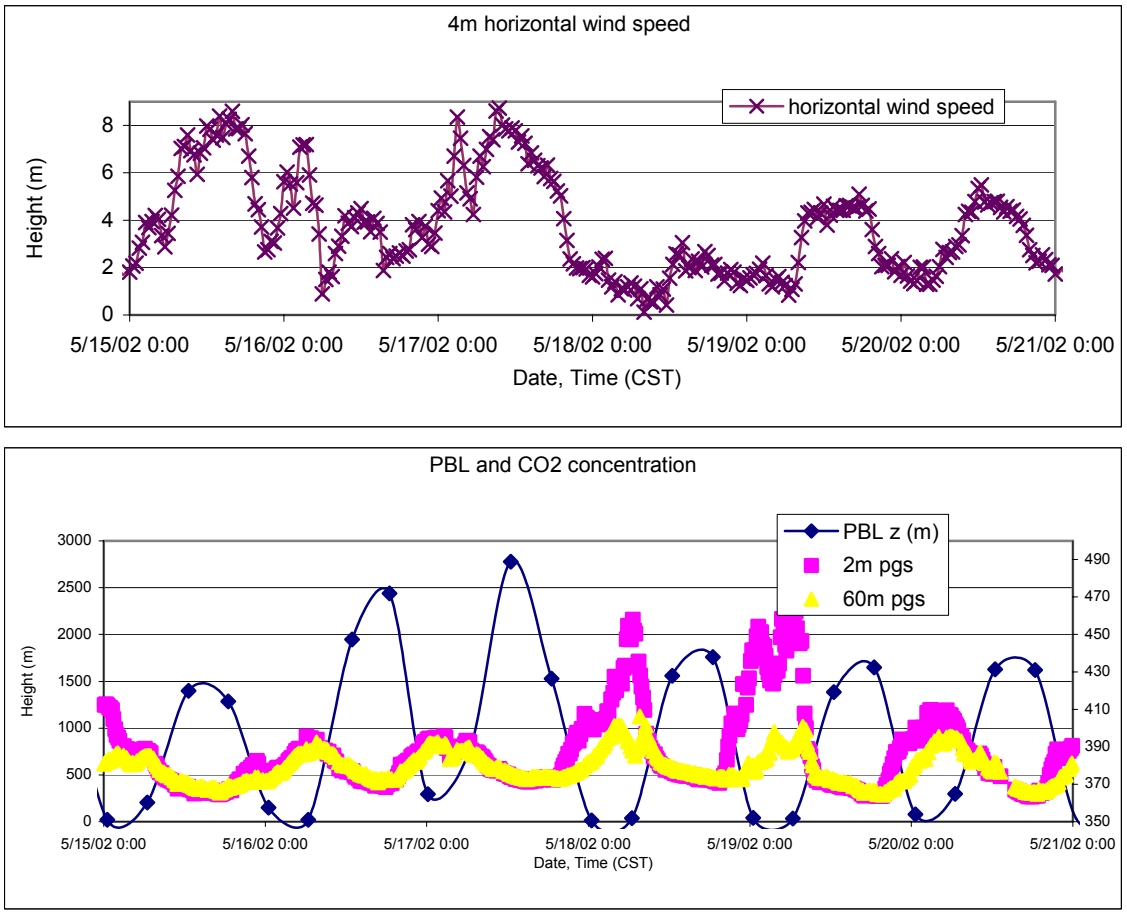

Figure 8. The correlation between horizontal wind speeds at 4m and 2m CO_2 concentration is not very consistent.

To explore the inconsistency between the nighttime PBL height and wind speed during the nights of May 18th, 19th, and 20th, we can look at wind direction. Southern wind, generally warmer, should result in higher PBL heights, whether as northern wind might bring in cold air and drop the PBL. Figure 9 shows directions of the wind at the end of these three nights. May 18th, with a rather low calculated PBL height is dominated by wind coming from the north. May 19th, also with a low calculated PBL height, has consistent wind from the south. This contradiction is further underlined by wind direction of May 20th. The PBL height was calculated

higher then both May 18th and 19th, but the wind was more eastwardly than the May 19th. The conclusion, at least for these particular dates, is that the relationship between wind and nighttime PBL is more complex than postulated.

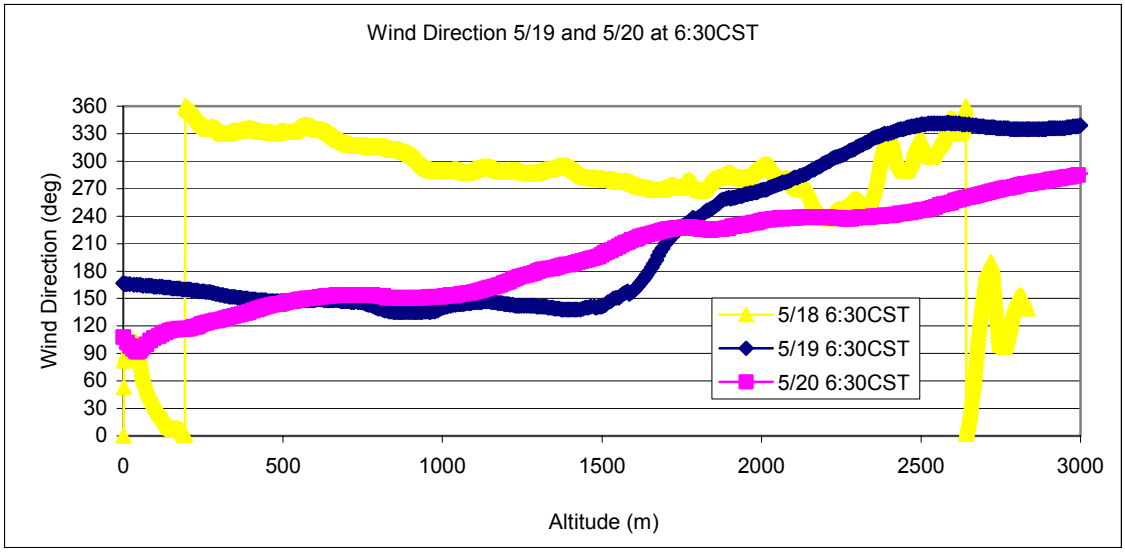

Figure 9. Wind direction at 6:30 CST on 5/18, 5/19, 5/20. Zero degrees denotes wind coming from the north, while 90 degrees is wind coming from the east. Despite having a north dominated wind on 5/18, the nighttime PBL was estimated higher than on 5/20.

5.0 Conclusions

Although there are difficulties in applying the Heffter Method to balloon sonde data, it is a good starting point. There are several modifications one might want to consider in implementing the algorithm:

a) Just as the condition two is iteratively relaxed on failure, have condition one relaxed on failure. Whether to do this before, after, or intermixed with relaxing of condition two is an issue requiring more analysis.
b) Add another technique to attempt to detect a sharp change in d(Θ)/dz. Bill Riley suggested fitting a parabola to small, "mirrored", snippet of Θ vs. height plot. As soon as the curvature becomes significant, note. This could be weighed together with other eligible points as determined by Heffter Technique.
c) Instead of picking the first point that you find as PBL, create an array of several eligible ones by running through the data several times. Then pick the best one by applying pre-determined criteria.

Bibliography

(1) Stull, R. B. An Introduction to Boundary Layer Meteorology, Kluwer Academic Publishers, 1993.

(2) Lecture notes, Earth and Planetary Science 129, Biomeorology, Fall 2002, Dennis Baldocchi.

(3) Lloyd, J., Francey, R.J., Mollicone, D., et all, *Vertical Profiles, Boundary Layer Budgets, and Regiounal Flux Estimates for CO2 and its C-13/C-12 ratios and for water vapor above a forest/bog mosaic in central Siberia*, Global Biogeochemical Cycles, Jun 2001, V15(N2):267-284.

(4) Heffter, J. L., 1980: Transport layer depth calculations. *Second Joint Conference on Applications of Air Pollution Meteorology*, New Orleans, LA (1980).

(5) Denning, A.S., Takahashi, T., Friedlingstein, P., *Can a strong atmospheric CO_2 rectifier effect be reconciled with a "reasonable" carbon budget?*, Tellus Series B – Chemical and Physical Meteorology, Apr 1999, V51(N2):249-253.

Acknowledgement

This work was supported by the Atmospheric Radiation Measurement Program, Biological and Environmental Research, Office of Science, U.S. Department of Energy under Contract No. DE-AC03-76SF00098. The author would also like to thank Margaret Torn and Marc Fischer.

Appendix

The complete text of the program used to implement the Heffter Method is given below:

```
/*********************************************************
 Igor Pesenson                                  sgppbl.c


 Given sonic data from the arm archives, this program uses
 the Heffter Technique to calculate the PBL height of the data.

 Heffter Technique:
       Potential temperature profiles are computed from the vertical
profile of temp and pres. The profiles are analyzed for the existence
of a "critical inversion", which is assumed to mark the top of the
mixed layer. A critical inversion is defined as the lowest inversion
that meets the following two criteria:
       (1)     del(theta)/del z > .005 Km-1
       (2)     (theta top) - (theta bottom) > 2 K

 Specific Algorithm:
       Within the framework of the Heffter Technique, a few
modifications have been added to aid the program:
          a) Data is analyzed only up to height PBLMAX (currently
3000m) that is set as a constant. This is to avoid unnecessary
calculations.
          b) Data is analyzed through using the original Heffter
constraints. If no point is found to satisfy them below PBLMAX, the
second constraint, theta difference of 2 K is relaxed by
heffter_delt_inc (currently .1). The data is then analyzed up to PBLMAX
again. The second condition keeps getting iteratively reduced as low as
possible (>0). If still no point is found, a failure message is
displayed.

Input:
       None. The program read the file "/home/igorp/pbl/pbl-input-
files.txt" This file has to list the file names of the sonde files to
be analyzed, in order. The program finds the files in /data/arm-
archive/sonde/a1 opens them one by one and reads the cdf data.
       In the future, it might be easier to change the program to so
that a date range can be input on command line and the files would be
pulled automatically.

Output:
       Output is to a file currently named "/home/igorp/pbl/pbl-
output.csv". For each input file, the program outputs a line into the
output file containing the following:

year,jday,hhmm,z PBL (m),tdry at pbl (deg C),pres at pbl (mbar),ave q
from 0 to pbl (g/m3),ave q from (pbl+500) to (pbl+1000) (g/m3),ave q
from (pbl+500) to (pbl+2500) (g/m3),ave rh from 0 to pbl (%),ave rh
```

```
from (pbl+500) to (pbl+1000) (%),ave rh from (pbl+500) to (pbl+2500)
(%),ave tdry from 0 to pbl (deg C),ave tdry from (pbl+500) to
(pbl+1000) (deg C),ave tdry from (pbl+500) to (pbl+2500) (deg C)),ave
pres from 0 to pbl (mbar),ave pres from (pbl+500) to (pbl+1000)
(mbar),ave pres from (pbl+500) to (pbl+2500) (mbar)

Revisions:
      r4 - standardized input and output and cleaned up the code some.

*********************************************************/

#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <float.h>
#include <string.h>
#include <netcdf.h>

/* define constants */

const int QMOLS = 1;              /* variable that determines units of
q: 1=mmol/mol, 0=g/m3  */
const float SITE_HEIGHT = 315.0;  /* elevation of CART site */
const float Pnot = 1013;          /* std pressure in mbar */
const int PBLMAX = 3000;          /* max pbl height in meters */
const int PBLMIN = 0;             /* min pbl height in meters */
const float RCP = .286;           /* (R/mCp) used as exponed to get
theta */
const float FILL = -9999;         /* bad value fill */


/* declare functions */

static void setupIO();       /* get date and time from file name and
set up input/output */
static void setDateTime();   /* set up the date and time based on input
file */
static void openInFile();    /* define and open file for input */
static void readInFile();    /* read in the b1 file */
static void calcTheta();     /* calc potential temp from input */
static void calcSlope();     /* calc delta theta/delta z */
//static void calc2ndDir();    /* calc second d(d(theta)/dz)/dz */
static void findPbl();       /* calc pbl */
static void findQbar();      /* calc q and find, printout qbar, rh bar
*/
static void handle_error();  /* descipher and print netcdf errors */
static int day_of_year();    /* set day of year from month and day */


/* declare global variables */

int nc_status, ncid_in;         /* netCDF variables */
int dateint;                    /* date of file looked at */
int timeint;
int yrint;
int dayint;
```

```
int monint;
int jday;
int pbl_index=0;
float slope[5000];                /* array holding the slopes of theta
with z */
//float secdir[5000];             /* array holding the second
derivatives of theta */
char file[38];

/* declare the structures */

/* in_variables: this structure is used to hold
   input variables from the cdf file */
struct in_variables {
  int base_time;                                        /* base time
of the data file */
  double time_offset[5000];                             /* time offset
of data line from base time */
  int time_offset_id, pres_id, tdry_id, alt_id, rh_id;   /* variable
identification numbers in cdf file */
  /* note these will be used starting with index 0, not 1 */
  float pres[5000], tdry[5000], alt[5000];              /* variable
values */
  /* potential temp will be calculated from pres and tdry */
  float theta[5000];
  float rh[5000], q[5000];
  size_t nrecords;                                      /* nrecords in
cdf file */
}in;

/* out_variables: this structure is used to
   hold output variables */
struct out_variables {
  int year, jday, sondetime;
  float zpbl,tpbl,prespbl;
  float qBelowPbl,qAbovePbl1,qAbovePbl2;
  float rhBelowPbl,rhAbovePbl1,rhAbovePbl2;
  float tBelowPbl,tAbovePbl1,tAbovePbl2;
  float pBelowPbl,pAbovePbl1,pAbovePbl2;
}out;


/* define file pointers */

FILE *fptr1;          /* output file */
FILE *fptr2;          /* file listing input file names */



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

  /* get date and time from the argument file and set appropriate
global variables */
  setupIO();

  /* while the input file lists files to look at */
  while( fscanf(fptr2,"%s",file)>0 ) {
```

```
    printf("Input file is %s\n",file);
    setDateTime();

    /* create and open the input file */
    openInFile();

    readInFile();

    calcTheta();

    calcSlope();

    //calc2ndDir();

    findPbl();

    findQbar();

    nc_close(ncid_in);
  }

  /* close the file pointers */
  fclose(fptr1);
  fclose(fptr2);

  return 0;
  /* end of main() */
}



/* setupIO(): sets up some IO variables such as the input file name
   and the output file name */
static void setupIO() {
  char outfile[34],infile[48];

  /* create and open the output file */
  strcpy(outfile,"/home/igorp/pbl/pbl-output.csv");
  if ( (fptr1=fopen(outfile,"w")) == NULL) {
    fprintf(stderr,"Can not open output file: %s\n",outfile);
    exit(-1);
  }

  /* open the file listing input file names */
  strcpy(infile,"/home/igorp/pbl/pbl-input-files.txt");
  if ( (fptr2=fopen(infile,"rt")) == NULL) {
    fprintf(stderr,"Can not open input file: %s\n",infile);
    exit(-1);
  }

  return;
  /* end of setup() */
}


/* setDateTime(): get date and time from the argument file
```

```
   and set appropriate global variables */
static void setDateTime() {
  char datestr[8], timestr[8], yrstr[5], monstr[3], daystr[3];
  char *ptr;
  int i,j, length, start;
  int ch='.';

  /* find pointer to first "." */
  if ( (ptr = strchr(file, ch)) == NULL ) {
    fprintf(stderr,"Bad input file name, aborting program.\n");
    exit(1);
  }
  /* result if lenght of name up to '.' */
  start = ptr-file;
  /* find length of filename */
  length = strlen(file);
  /* get the date from the filename */
  j=0;
  for(i=(start+4); i<(start+12); i++) {
    datestr[j] = file[i];
    j++;
  }
  dateint = atoi(datestr);
  /* now get the time in a similar manner */
  j=0;
  for(i=(start+13); i<(start+17); i++) {
    timestr[j] = file[i];
    j++;
  }
  timeint = atoi(timestr);
  //printf("timeint is %d, timestr is %s\n", timeint, timestr);
  /* now get the year, month and day */
  yrstr[0] = datestr[0];
  yrstr[1] = datestr[1];
  yrstr[2] = datestr[2];
  yrstr[3] = datestr[3];
  yrint = atoi(yrstr);
  monstr[0] = datestr[4];
  monstr[1] = datestr[5];
  monint = atoi(monstr);
  daystr[0] = datestr[6];
  daystr[1] = datestr[7];
  dayint = atoi(daystr);

  /* find julian day of year */
  jday = day_of_year(yrint, monint, dayint);
  //printf("Date is %d, time is %d, mon is %d, day is %d, jday is
%d\n", dateint, timeint, monint, dayint, jday);

  return;
  /* end setDateTime() */
}



/* openInFile(): opens the sonde file you're looking at
   note it currently looks into a1 directory. this should be changed
```

```
   later to look into /yyyy/a1 directory for neatness */
static void openInFile() {
  char infile[52];

  strcpy(infile, "/data/arm-archive/sonde/a1/");
  strcat(infile, file);
  //printf("opening cdf %s for reading\n",infile);

  if ((nc_status = nc_open(infile, NC_NOWRITE, &ncid_in)) != NC_NOERR)
{
    fprintf(stderr,"openInFile(): NetCDF error #%i %s  \n NetCDF-
Opening file '%s'\n",
         nc_status, nc_strerror(nc_status), infile);
    exit(-1);
  }

  return;
  /* end openInFile() */
}



/* readInFile(): read in the sonde cdf file */
static void readInFile() {
  int i, unlimdimid, varid;

  /* ask the netcdf input file how many records there are
     note the sonde files seem to count number of records starting with
0, so
     0,1,2,3 records, not 1,2,3,4 records */
  if ((nc_status = nc_inq_unlimdim(ncid_in, &unlimdimid)) != NC_NOERR)
{
    fprintf(stderr,"main: NetCDF error #%i %s\n", nc_status,
nc_strerror(nc_status));
    exit(-1);
  }
  if ((nc_status = nc_inq_dimlen(ncid_in, unlimdimid, &in.nrecords)) !=
NC_NOERR) {
    fprintf(stderr,"main: NetCDF error #%i %s\n", nc_status,
nc_strerror(nc_status));
    exit(-1);
  }

  /* check the number of records */
  if (in.nrecords < 10) {
    fprintf(stderr, "Too few records in file, unable to calculate
PBL\n");
    exit(1);
  }

  //printf("The input file has %d records\n",in.nrecords);

  nc_status=nc_inq_varid(ncid_in, "base_time", &varid);
  if (nc_status != NC_NOERR) handle_error(nc_status);
  nc_status=nc_get_var1_int(ncid_in, varid, &i, &in.base_time);

  //printf("The base time is %d\n", in.base_time);
```

```
  /* find the varid's for p,tdry,time_offset,alt */
  nc_status=nc_inq_varid(ncid_in, "time_offset", &in.time_offset_id);
  if (nc_status != NC_NOERR) handle_error(nc_status);
  //printf("The time offset id is %d\n", in.time_offset_id);

  nc_status=nc_inq_varid(ncid_in, "pres", &in.pres_id);
  if (nc_status != NC_NOERR) handle_error(nc_status);
  //printf("The pres id is %d\n", in.pres_id);

  nc_status=nc_inq_varid(ncid_in, "tdry", &in.tdry_id);
  if (nc_status != NC_NOERR) handle_error(nc_status);
  //printf("The tdry id is %d\n", in.tdry_id);

  nc_status=nc_inq_varid(ncid_in, "alt", &in.alt_id);
  if (nc_status != NC_NOERR) handle_error(nc_status);
  //printf("The alt id is %d\n", in.alt_id);

  nc_status=nc_inq_varid(ncid_in, "rh", &in.rh_id);
  if (nc_status != NC_NOERR) handle_error(nc_status);

  /* note the input arrays are used starting with index 0, not 1,
     and defined no bigger than 2999 */
  for(i=0; i<in.nrecords; i++) {
    nc_status=nc_get_var1_double(ncid_in, in.time_offset_id, &i,
&in.time_offset[i]);
    if (nc_status != NC_NOERR) handle_error(nc_status);
    nc_status=nc_get_var1_float(ncid_in, in.pres_id, &i, &in.pres[i]);
    if (nc_status != NC_NOERR) handle_error(nc_status);
    nc_status=nc_get_var1_float(ncid_in, in.tdry_id, &i, &in.tdry[i]);
    if (nc_status != NC_NOERR) handle_error(nc_status);
    nc_status=nc_get_var1_float(ncid_in, in.alt_id, &i, &in.alt[i]);
    if (nc_status != NC_NOERR) handle_error(nc_status);
    /* correct altitude from above sea level to above surface */
    in.alt[i] = in.alt[i] - SITE_HEIGHT;

    nc_status=nc_get_var1_float(ncid_in, in.rh_id, &i, &in.rh[i]);
    if (nc_status != NC_NOERR) handle_error(nc_status);

    //printf("at i=%d, height is %.0f\n",i,in.alt[i]);
  }

  return;
  /* end of readInFile() */
}


/* calcTheta(): calculate virtual temperature */
static void calcTheta() {
  int i, unlimdimid;
  size_t nrecords;

  /* note that tdry is originally in C so need to convert to K */
  for(i=0; i<in.nrecords; i++) {
    in.theta[i] = (in.tdry[i]+273.16) * pow((Pnot / in.pres[i]), RCP);
  }
```

```
  return;
  /* end calcTheta() */
}


/* calcSlope(): calculate the slope */
static void calcSlope() {
  int i;

  for(i=0; i<in.nrecords-2; i++) {
    slope[i] =(in.theta[i+1]-in.theta[i])/(in.alt[i+1]-in.alt[i]);
    /* bad attempt at de-spiking */
    //if(slope[i] > 30) slope[i]=0;
    //printf("Slope at height %f is %f, theta is %f\n",in.alt[i],
slope[i], in.theta[i]);
    //printf("at i %d, height %f,  slope is %f\n", i,
in.alt[i],slope[i]);
  }

}

/*
static void calc2ndDir() {
  int i;

  for(i=0; i<in.nrecords-2; i++) {
    secdir[i] =(slope[i+1]-slope[i])/(in.alt[i+1]-in.alt[i]);
    printf("At height %f, second dir is %f, slope is %f, theta is
%f\n",in.alt[i], secdir[i], slope[i], in.theta[i]);
  }
}
*/


/* findPbl(): actually applies the Heffter Method */
static void findPbl() {
  int i, cond1_index, pblfound;
  float theta_b, theta_t;
  /* delta T in Heffter Technique and the increment it will be
decreased by
     should the initial value be too big. To avoid trouble, have delt
divisible
     by delt_inc. */
  float heffter_delt, heffter_delt_inc;


  /* initialize heffter delta t as 2.0 This will get decremented if it
happens
     to be too big after the first run through */
  heffter_delt = 2.0;
  heffter_delt_inc = .1;

  /* here I need to install some tests that the i chosen is reasonable
     so i>such, meters[i]<such, otherwise bad data */

  /* initialize array searching index */
  i=0;
```

```
  pblfound = 0;
  while (pblfound == 0) {

    /* find first point that satisfies Heffter Condition 1 */
    while ( (slope[i] <= .005) ) {
      //printf("Slope at height %f is %f, theta is %f\n",in.alt[i],
slope[i], in.theta[i]);
      //printf("Height %f, Slope %f, Theta %f\n",in.alt[i], slope[i],
in.theta[i]);
      if ( (in.alt[i] > PBLMAX) &&
         (heffter_delt <= heffter_delt_inc) ) {
      fprintf(stderr, "No point satisfies Heffter Condition 1 below
%d\n", PBLMAX);
      fprintf(stderr, "Unable to calculate PBL for this file.\n");
      exit(1);
      }
      /* decrement heffter delta T */
      else if ( (in.alt[i] > PBLMAX) ) {
      fprintf(stderr, "(a)Heffter Condition 2: del_T of %f too large,
decreasing by %f\n",
            heffter_delt, heffter_delt_inc);
      heffter_delt = heffter_delt - heffter_delt_inc;
      /* reset the index to start from lowest point again */
      i = 0;
      pblfound = 0;
      }
      else {
      i++;
      }
    }

    /* mark theta at bottom of critical inversion layer */
    theta_b = in.theta[i];
    theta_t = in.theta[i];
    cond1_index = i;
    //printf("Slope at height %f is %f, theta is %f\n",in.alt[i],
slope[i], in.theta[i]);
    //printf("Height %f, Slope %f, Theta %f\n",in.alt[i], slope[i],
in.theta[i]);
    //printf("Set theta_b at Height %f, theta %f\n", in.alt[i],
in.theta[i]);

    /* also, what to do when theta is increasing steadily with height?
(ie 20020501.1128)
       PBL is badly defined - mark first qualifying point, note that
data is unrealiable */

    /* now keep climbing until you find pbl, or conditions fail */
    i = cond1_index;
    while( ((theta_t-theta_b) < heffter_delt)
         && (slope[i] > .005)
         && (in.alt[i] <= PBLMAX) ) {
      /* increment to index of next point */
      i++;
      theta_t = in.theta[i];
      //printf("Going to Height %f, Slope %f, Theta %f\n",in.alt[i],
slope[i], in.theta[i]);
```

```
      //printf("(theta_t-theta_b), %f > %.2f, slope[i] is %f, in.alt[i]
is %f\n", (theta_t-theta_b), heffter_delt, slope[i], in.alt[i]);
    }

    /* now that you're out of the loop, need to find out why */

    /* check if you ran out of data and if so change cond 2 before
looping */
    if (in.alt[i] >= PBLMAX) {
      /* check if heffter delta is too small to decrement */
      if (heffter_delt <= heffter_delt_inc) {
      fprintf(stderr, "No point satisfies Heffter Condition 2 below
%d\n", PBLMAX);
      fprintf(stderr, "Unable to calculate PBL for this file.\n");
      exit(1);
      }
      /* decrement heffter delta T */
      else {
      fprintf(stderr, "(b)Heffter Condition 2: del_T of %f too large,
decreasing by %f\n",
            heffter_delt, heffter_delt_inc);
      heffter_delt = heffter_delt - heffter_delt_inc;
      }
      /* reset the index to start from lowest point again */
      i = 0;
      pblfound = 0;
    }
    /* check if while loop exited because slope went bad */
    else if ( slope[i] <= .005 ) {
      //      printf("Slope(%.4f) dropped below threshold, start
searching again\n", slope[i]);
      //printf("Slope at height %f is %f, theta is %f\n",in.alt[i],
slope[i], in.theta[i]);
      //printf("Height %f, Slope %f, Theta %f\n",in.alt[i], slope[i],
in.theta[i]);
      /* slope dropped below threshold, start over from the last point
*/
      //      i = cond1_index + 1;
      /* check if point that failed slope is the next one. If yes,
       then start search just beyond it. If not, then start
       search from the following one. */
      i = i + 1;
      pblfound = 0;
    }
    /* or loop exited because we found our point */
    else {
      pblfound = 1;
      pbl_index = i;
    }
    /* end of while(pblfound==0) */
  }

  /* stupid bug!! it will not let you set pblheight here so I'm doing
it above.
     I don't know why and it's frustrating */
  if (pblfound == 1) {
    out.year = yrint;
```

```
    out.jday = jday;
    out.sondetime = timeint;
    out.zpbl = in.alt[i];
    out.tpbl = in.tdry[i];
    out.prespbl = in.pres[i];

fprintf(fptr1,"%d,%d,%04d,%.2f,%.3f,%.3f,",out.year,out.jday,out.sondet
ime,out.zpbl,out.tpbl,out.prespbl);
 }

  return;
  /* end of findPbl() */
}


/* findQbar(): find the average q, averate t, average rh, and pressure
above and below the pbl */
static void findQbar() {
  int i, i_pbl, i_t1;
  float qbartop1=0, qbartop2=0, qbarbot=0, satpres, vappres;
  float rhtop1=0, rhtop2=0, rhbot=0;
  float tdrytop1=0, tdrytop2=0, tdrybot=0;
  float ptop1=0, ptop2=0, pbot=0;
  int badline=0;

  /* find q */
  for(i=0; i<in.nrecords; i++) {
    if (in.rh[i] != -9999) {
      satpres = 611*exp((17.502*in.tdry[i])/(in.tdry[i]+240.97)); /*
[tdry]=C, [satpres]=Pa */
      vappres = (in.rh[i]/100) * satpres;   /* [vapres]=Pa */
      in.q[i] = (2.165 * vappres)/(in.tdry[i] + 273.16);  /* [rho]=g/m3
*/
      /* now change into mmol/mol if this options was chosen */
      if (QMOLS == 1) {
      in.q[i] = (in.q[i] * 8.3143 *
(in.tdry[i]+273.16))/(in.pres[i]*100);    /* g/mol */
      in.q[i] = (in.q[i]/18 * 1000);
/* g/mol */
      }
      //printf("z %f, rh %f, q %f\n", in.alt[i], in.rh[i], in.q[i]);
    } else
      badline++;
  }

  /* check how many bad values there are - if too many, output bad
values */
  if(badline < 200) {

    badline=0;
    /* find average q from 0m to pbl */
    for(i=0; in.alt[i]<in.alt[pbl_index]; i++) {
      if (in.rh[i] != -9999) {
      qbarbot+=in.q[i];
      rhbot+=in.rh[i];
      tdrybot+=in.tdry[i];
      pbot+=in.pres[i];
```

```
      } else
      badline++;
    }
    qbarbot/=(i-badline);
    rhbot/=(i-badline);
    tdrybot/=(i-badline);
    pbot/=(i-badline);
    i_pbl=i+1;

    /* find average q from pbl+500m to pbl+1000m */

    /* 'fast forward' to pbl+500 */
    for (i=i+2; in.alt[i]<=(in.alt[pbl_index]+500); i++) {}

    /* some sondes never leave the ground, so we have to check for bad
files */
    if (in.alt[i] <= in.alt[pbl_index]) {

      badline=0;
      for (i=i; in.alt[i]<=(in.alt[pbl_index] + 1000); i++) {
      if (in.rh[i] != -9999) {
        qbartop1+=in.q[i];
        rhtop1+=in.rh[i];
        tdrytop1+=in.tdry[i];
        ptop1+=in.pres[i];
      } else
        badline++;
      }
      qbartop1/=(i-badline);
      rhtop1/=(i-badline);
      tdrytop1/=(i-badline);
      ptop1/=(i-badline);

      /* find average q from pbl+500m to pbl+2500m */
      badline=0;
      for (i=i; in.alt[i]<(in.alt[pbl_index] + 2500); i++) {
      if(in.rh[i] != -9999) {
        qbartop2+=in.q[i];
        rhtop2+=in.rh[i];
        tdrytop2+=in.tdry[i];
        ptop2+=in.pres[i];
      } else
        badline++;
      }
      qbartop2/=(i-badline);
      rhtop2/=(i-badline);
      tdrytop2/=(i-badline);
      ptop2/=(i-badline);

    } else {
      /* bad sonde */
      qbartop1=qbartop2=-9999;
      rhtop1=rhtop2=-9999;
      tdrytop1=tdrytop2=-9999;
      ptop1=ptop2=-9999;
    }
  }
```

```
  /* too many bad lines */
  else {
    qbarbot=qbartop1=qbartop2=-9999;
    rhbot=rhtop1=rhtop2=-9999;
    tdrybot=tdrytop1=tdrytop2=-9999;
    pbot=ptop1=ptop2=-9999;
  }

  out.qBelowPbl = qbarbot;
  out.qAbovePbl1 = qbartop1;
  out.qAbovePbl2 = qbartop2;
  out.rhBelowPbl = rhbot;
  out.rhAbovePbl1 = rhtop1;
  out.rhAbovePbl2 = rhtop2;
  out.tBelowPbl = tdrybot;
  out.tAbovePbl1 = tdrytop1;
  out.tAbovePbl2 = tdrytop2;
  out.pBelowPbl = pbot;
  out.pAbovePbl1 = ptop1;
  out.pAbovePbl2 = ptop2;

  /* print out the remainder of output line (started by calcPbl) */

fprintf(fptr1,"%.3f,%.3f,%.3f,%.3f,%.3f,%.3f,%.3f,%.3f,%.3f,%.3f,%.3f,%
.3f\n",qbarbot,qbartop1,qbartop2,rhbot,rhtop1,rhtop2,tdrybot,tdrytop1,t
drytop2,pbot,ptop1,ptop2);

  return;
  /* end of findQBar() */
}


/* function to handle netcdf errors */
static void handle_error(int status) {
  fprintf(stderr, "NetCDF error: %s\n", nc_strerror(status));
  exit(-1);
}


/* day_of_year: set day of year from month and day */
static int day_of_year(int year, int month, int day) {
  int i, leap;

  char daytab[2][13] = {
    {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31},
    {0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}
  };

  leap = year%4 == 0 && year%100 != 0 || year%400 == 0;
  for (i = 1; i < month; i++)
    day += daytab[leap][i];
  return day;
}
```