Subject: Re: routine for GRIB data
Posted by m.hadfield on Tue, 03 Jul 2001 04:28:37 GMT
View Forum Message <> Reply to Message

Dominik Brunner's Hip-Hop program has tools for reading GRIB. See...

http://www.knmi.nl/onderzk/atmosam/hiphop/hiphop.html

I haven't used them myself.

I vaguely recall reading GRIB data some years ago by trapping output from
the wgrib command-line program. This is not the ideal way to do it!

Mark Hadfield
m.hadfield@niwa.cri.nz http://katipo.niwa.cri.nz/~hadfield
National Institute for Water and Atmospheric Research

--

Subject: Re: routine for GRIB data
Posted by Kenneth P. Bowman on Tue, 03 Jul 2001 12:42:14 GMT
View Forum Message <> Reply to Message

In article <86fe0ca1.0107022006.5d36b49a@posting.google.com>, Kwangwoo
<kwcho@kei.re.kr> wrote:

> I am looking for a IDL routine which can read the GRIB format data
> file. The data I would like to read now is the NCEP reanalysis data
> (monthly mean UFLUX and VFLUX). I appreciate for your cooperation.

The NOAA Climate Diagnostics Center provides the NCEP reanalysis data
in netCDF format, which is considerably easier to deal with than GRIB
files.

 http://www.cdc.noaa.gov/cdc/data.ncep.reanalysis.derived.htm l

Good luck,

Ken

Subject: Re: routine for GRIB data
Posted by Liam E. Gumley on Tue, 03 Jul 2001 13:36:13 GMT
View Forum Message <> Reply to Message

Kwangwoo wrote:
> I am looking for a IDL routine which can read the GRIB format data
> file. The data I would like to read now is the NCEP reanalysis data
> (monthly mean UFLUX and VFLUX). I appreciate for your cooperation.

IMHO the best utility for reading GRIB files is wgrib:

http://wesley.wwb.noaa.gov/wgrib.html

It is written in portable C code. Just dump the data from the GRIB file
to a binary file, then read the binary file in IDL.

Cheers,
Liam.
Practical IDL Programming
http://www.gumley.com/

Subject: Re: routine for GRIB data
Posted by R.G.S. on Tue, 03 Jul 2001 16:16:57 GMT
View Forum Message <> Reply to Message

Kwangwoo <kwcho@kei.re.kr> wrote in message
news:86fe0ca1.0107022006.5d36b49a@posting.google.com...
> I am looking for a IDL routine which can read the GRIB format data
> file. The data I would like to read now is the NCEP reanalysis data
> (monthly mean UFLUX and VFLUX). I appreciate for your cooperation.
>
> Kwangwoo
>
> ----------
> Kwangwoo Cho
> Global Environment Research Center
> Korea Environment Institute (KEI)
> 613-2 Pulgwang-Dong Eunpyong-Gu
> Seoul 122-706 Korea
> Tel: 82-2-380-7615
> Fax: 82-2-380-7688
> Email: kwcho@kei.re.kr
> ----------

I use the NetCDF ncep files. They are large, but IDL handles the NetCDF
nicely.

Below is an example piece of code that will print out file information from
a NCEp NetCDF file.

The ftp site is:
archive.cdc.noaa.gov

and data on pressure levels can be found at:
/Datasets/archive0/ncep.reanalysis/pressure/

Note: archive0 is for a certain time period, also check archive1,archive2
etc...

Cheers,
bob stockwell
colorado research associates

```
; open and read the (HUGE) netcdf file


Filename =' put file name here!!'


Cdfid = NCDF_OPEN( Filename )

glob = NCDF_INQUIRE(Cdfid)
print
print,'_____ glob  _____________'
help,glob,/st
print
print
print,'_____ info _____________'
print,'     ',' i ',' : ','name',' : ','size'
names = strarr(glob.ndims)
sizes = lonarr(glob.ndims)
for i = 0,glob.ndims-1 do begin
 NCDF_DIMINQ, Cdfid, i, Name, Size
 names(i) = name
 sizes(i) = size
 print,i,' : ',name,' : ',size
endfor

help,Cdfid
```

```
print
print, ' _________  Variables  ____________'
 for i=0,glob.nvars-1 do begin

  ; Get information about the variable
  info = ncdf_varinq(cdfid, i)
  FmtStr = '(A," (",A," ) Dimension Ids = [ ", 10(I0," "),$)'
  print, FORMAT=FmtStr, info.name,info.datatype, info.dim[*]
  print, ']'

  ; Get attributes associated with the variable
  for j=0,info.natts-1 do begin
   attname = ncdf_attname(cdfid,i,j)
   ncdf_attget,cdfid,i,attname,attvalue
   print,' Attribute ', attname, '=', string(attvalue)
  endfor
 endfor


levelid = NCDF_VARID(Cdfid, 'level')
NCDF_VARGET, Cdfid,levelid, level
help,level

levelid = NCDF_VARID(Cdfid, 'lon')
NCDF_VARGET, Cdfid,levelid, lon
help,lon

levelid = NCDF_VARID(Cdfid, 'lat')
NCDF_VARGET, Cdfid,levelid, lat
help,lat

levelid = NCDF_VARID(Cdfid, 'time')
NCDF_VARGET, Cdfid,levelid, time
help,time

NCDF_CLOSE, Cdfid


end
```

Subject: Re: routine for GRIB data
Posted by wmconnolley on Tue, 03 Jul 2001 21:10:02 GMT
View Forum Message <> Reply to Message

Mark Hadfield <m.hadfield@niwa.cri.nz> wrote:
> I vaguely recall reading GRIB data some years ago by trapping output from
> the wgrib command-line program. This is not the ideal way to do it!

Oh dear. I still do this. It works, if you're going to convert all
your files to another format. Too slow for reading lots in real time,
though.

-W

--
William M Connolley | wmc@bas.ac.uk | http://www.nerc-bas.ac.uk/icd/wmc/
Climate Modeller, British Antarctic Survey | Disclaimer: I speak for myself
I'm a .signature virus! copy me into your .signature file & help me spread!

Subject: Re: routine for GRIB data
Posted by Martin Schultz on Fri, 06 Jul 2001 16:44:10 GMT
View Forum Message <> Reply to Message

wmc@bas.ac.uk writes:

> Mark Hadfield <m.hadfield@niwa.cri.nz> wrote:
>> I vaguely recall reading GRIB data some years ago by trapping output from
>> the wgrib command-line program. This is not the ideal way to do it!
>
> Oh dear. I still do this. It works, if you're going to convert all
> your files to another format. Too slow for reading lots in real time,
> though.
>
> -W
>
> --
> William M Connolley | wmc@bas.ac.uk | http://www.nerc-bas.ac.uk/icd/wmc/
> Climate Modeller, British Antarctic Survey | Disclaimer: I speak for myself
> I'm a .signature virus! copy me into your .signature file & help me spread!
>

Hi you fellow GRIBBERS ;-)

As I am repeatedly faced with the same problem, I started out on a
read_wgrib routine. As the name says, it also uses wgrib to analyse the
GRIB file, but I tried to optimise it a little bit here and there, so
it's already working for some applications. There is a lot of timing
output in the current version of the program and you can quickly locate
where it spends all it's time: it is the analysis of the header information
provided by wgrib that takes up most of the time. This immediately leads
to 3 suggestions:
(1) analyse only the part of the header that you really need
(2) make read_wgrib an object so that at least you scan the header only
    once for an individual file

(3) ask the wgrib developper for a binary or more formatted header output.
Currently it is necessary to do some string parsing, and if this could be avoided and instead a formatted read could be used, that would probably save a lot of time.

The program is attached below. Current limitation: it can only read gridded data as I would need to find a way to assess the dimensions of spectral data. (I believe this is possible if one converts the data with header, but I didn't have time to test).

Interface:

```
PRO read_wgrib, data, filename=filename, $
          code=code, date=date, timestep=timestep, level=level, $
          metadata=metadata, dims=dims, $
          header_only=header_only
```

data will be a structure with tags like CODE100, CODE234, etc. Names should be added eventually, but I had trouble with getting my own code table to work properly and in a flexible way, so I decided to go for code number rather than a wrong name.

The code, date, timestep, level keywords are input, metadata and dims are output, and header_only is a switch. Timestep is computed by counting the unique date values, and is thus equivalent to date, but you don't need to know the date value.

I'd be happy to hear if this works for you, and I'd be even happier if someone takes it from here and makes it a real library tool.

Best regards,

Martin

--
[[[[[[[[[[
[[Dr. Martin Schultz   Max-Planck-Institut fuer Meteorologie   [[
[[Bundesstr. 55, 20146 Hamburg   [[
[[phone: +49 40 41173-308   [[
[[fax: +49 40 41173-298   [[
[[martin.schultz@dkrz.de   [[
[[[[[[[[[[

```
;+
;
```

```
; NAME:
;    read_wgrib
;
; PURPOSE:
;    Read GRIB formatted data files using the wgrib utility
;
; NOTES:
;    (1) Currently supports only grid point fields (record size of
;    spectral fields is a little harder to determine). Use h option
;    with wgrib to get record size!!!!
;    (2) Analysing header is the slowest process: need to improve!!
;    (3) Add code names!! Use gribtab file ....
;    (4) Add dimensional information from grid object ....
;    (5) Rewrite as object ....
;
; MODIFICATION HISTORY:
;    mgs, 05 Jul 2001 : VERSION 0.1
;
;-


FUNCTION parse_wgrib_line, line

   ;; Extract the information from one wgrib output line

   ;; Define result structure
   ;;  The inventory consists of several fields separated by colons. The contents
   ;;  of the fields are:
   ;;   1. Record number
   ;;   2. Position in bytes
   ;;   3. Date (YYMMDDHH).
   ;;   4. Parameter name (LAND=land/sea mask)  [not used]
   ;;   5. Indicator of parameter and units (grib PDS octet 9)
   ;;   6. Type of level/layer (grib PDS octet 10)
   ;;   7. Height, pressure, etc (grib PDS octets 11-12)
   ;;   8. Time Range (grib PDS octet 21)
   ;;   9. Period of time 1, (grib PDS octet 19)
   ;;  10. Period of time 2, (grib PDS octet 20)
   ;;  11. Forecast time unit (grib PDS octet 18)
   ;;  12. level
   ;;  13. anl=analysis, fcst=forecast
   ;;  14. NAve (number of grids used to make average)
   ;;
   ;; Value-added information:
   ;;      timestep : n-th different date value
   ;;      nx, ny   : the grid dimensions for gridded data
   ;;      gridtype : the grid type (4=gaussian)
   ;;      text     : the original string as read from wgrib output
```

```
  template = { grib_record, $
               recno:0L, pos:0L, date:'', code:-1L, $
               levtype:0L, type:0L, $
;;               TR:0L, p1:0L, p2:0L, TU:0L, $
               level:'', ana_fcst:'', gridav:0L, $
               timestep:-1L, nx:0L, ny:0L, gridtype:-1L, text:'' }

  tags = StrTok(line,':',/extract)
  template.recno    = Long(tags[0])
  template.pos      = Long(tags[1])
  template.date     = StrMid(tags[2],2)
;;  template.codename = Long(tags[3])   ; code name (requires proper
;;                                      ; code table)
  template.code     = Long(StrMid(tags[4],6))
  template.levtype  = Long(StrMid(tags[5],6))
  template.type     = Long(StrMid(tags[6],6))
;  template.tr       = Long(StrMid(tags[7],3))
;  template.p1       = Long(StrMid(tags[8],3))
;  template.p2       = Long(StrMid(tags[9],3))
;  template.tu       = Long(StrMid(tags[10],6))
  template.level    = StrTrim(tags[11],2)
  template.ana_fcst = StrTrim(tags[12],2)
  template.gridav   = Long(StrMid(tags[13],5))
  template.text     = line

  RETURN, template

END



PRO get_wgrib_extrainfo, metadata, filename, inventoryfile

  ;; Get more specific grid information for individual codes
  ;; Use position parameter from previous scan to avoid lengthy scans
  ucodes = get_wgrib_info(metadata, /CODES)

  FOR i=0L, N_Elements(ucodes)-1 DO BEGIN
    first = ( where(metadata.code EQ ucodes[i]) )[0]
;     cmd = 'echo '+metadata[first].text+' | wgrib -i -V '+filename
    cmd = 'wgrib -V -p '+StrTrim(metadata[first].pos,2)+' '+filename
    spawn, cmd, result
    w = Where(StrPos(result,'nx') GT 0, cnt)
    IF cnt GT 0 THEN BEGIN
      result = result[w[0]]
      p0 = StrPos(result,'nx')
      p1 = StrPos(result,'ny')
```

```
      nx = Long(StrMid(result,p0+3,p1-p0-3))

      p0 = p1
      p1 = StrPos(result,'GDS')
      ny = Long(StrMid(result,p0+3,p1-p0-3))

      p0 = StrPos(result,'grid')
      p1 = StrPos(result,'num_in')
      gridtype = Long(StrMid(result,p0+5,p1-p0-5))

      wok = Where(metadata.code EQ ucodes[i])
      metadata[wok].nx = nx
      metadata[wok].ny = ny
      metadata[wok].gridtype = gridtype
    ENDIF

  ENDFOR

END


FUNCTION get_wgrib_header, inventoryfile, filename

  ;; Analyse header information extracted from wgrib
  OpenR, lun, inventoryfile, /Get_Lun
  line = ''

  count = 0L
  WHILE NOT eof(lun) DO BEGIN
    ReadF, lun, line
    record = parse_wgrib_line(line)
    IF count EQ 0 THEN BEGIN
      metadata = Replicate(record, 1000)
    ENDIF ELSE BEGIN
      IF count MOD 1000 EQ 0 THEN BEGIN
        metadata = [ metadata, Replicate(record, 1000) ]
      ENDIF ELSE BEGIN
        metadata[count] = record
      ENDELSE
    ENDELSE
    count = count + 1
  ENDWHILE
  Free_Lun, lun
  metadata = metadata[0:count-1]

  Message,'Read header information for '+StrTrim(count,2)+' records.',/Info

  ;; Add timestep property:
```

```
  ;; Assumption is that each time step is stored with a different
  ;; date value
  dates = metadata.date
  udates = dates[Uniq(dates, sort(dates))]

  FOR i=0L, N_Elements(udates)-1 DO BEGIN
    w = Where(dates EQ udates[i])
    metadata[w].timestep = i+1
  ENDFOR

  RETURN, metadata
END



FUNCTION get_wgrib_info, metadata, codes=codes, dates=dates, $
                         timesteps=timesteps, levels=levels

  ;; Return uniq values of properties. Set keyword to return a
  ;; selected property.

  result = -1L

  IF Keyword_Set(codes) THEN BEGIN
    thecodes = metadata.code
    result = thecodes[Uniq(thecodes, sort(thecodes))]
  ENDIF

  IF Keyword_Set(dates) THEN BEGIN
    thedates = metadata.date
    result = thedates[Uniq(thedates, sort(thedates))]
  ENDIF

  IF Keyword_Set(timesteps) THEN BEGIN
    thetimesteps = metadata.timestep
    result = thetimesteps[Uniq(thetimesteps, sort(thetimesteps))]
  ENDIF

  IF Keyword_Set(levels) THEN BEGIN
    thelevels = metadata.level
    result = thelevels[Uniq(thelevels, sort(thelevels))]
  ENDIF

  RETURN, result

END
```

```
FUNCTION get_wgrib_selection, metadata, code=code, date=date, $
                              timestep=timestep, level=level


  ;; Create default result: use all records
  result = Make_Array(N_Elements(metadata),/Long,Value=1L)

  ;; Extract all codes and find out which ones are to be excluded
  IF N_Elements(code) GT 0 THEN BEGIN
    thecodes = metadata.code
    ucodes = thecodes[Uniq(thecodes, sort(thecodes))]
    FOR i=0L, N_Elements(ucodes)-1 DO BEGIN
      wok = Where(code EQ ucodes[i], cnt)
      IF cnt EQ 0 THEN BEGIN
        wexcl = Where(thecodes EQ ucodes[i], cnt)
        IF cnt GT 0 THEN result[wexcl] = 0
      ENDIF
    ENDFOR
  ENDIF

  ;; Extract all dates and find out which ones are to be excluded
  IF N_Elements(date) GT 0 THEN BEGIN
    thedates = metadata.date
    udates = thedates[Uniq(thedates, sort(thedates))]
    FOR i=0L, N_Elements(udates)-1 DO BEGIN
      wok = Where(date EQ udates[i], cnt)
      IF cnt EQ 0 THEN BEGIN
        wexcl = Where(thedates EQ udates[i], cnt)
        IF cnt GT 0 THEN result[wexcl] = 0
      ENDIF
    ENDFOR
  ENDIF

  ;; Extract all timesteps and find out which ones are to be excluded
  IF N_Elements(timestep) GT 0 THEN BEGIN
    thetimesteps = metadata.timestep
    utimesteps = thetimesteps[Uniq(thetimesteps, sort(thetimesteps))]
    FOR i=0L, N_Elements(utimesteps)-1 DO BEGIN
      wok = Where(timestep EQ utimesteps[i], cnt)
      IF cnt EQ 0 THEN BEGIN
        wexcl = Where(thetimesteps EQ utimesteps[i], cnt)
        IF cnt GT 0 THEN result[wexcl] = 0
      ENDIF
    ENDFOR
  ENDIF

  ;; Extract all levels and find out which ones are to be excluded
```

```
  IF N_Elements(level) GT 0 THEN BEGIN
    thelevels = metadata.level
    ulevels = thelevels[Uniq(thelevels, sort(thelevels))]
    FOR i=0L, N_Elements(ulevels)-1 DO BEGIN
      wok = Where(level EQ ulevels[i], cnt)
      IF cnt EQ 0 THEN BEGIN
        wexcl = Where(thelevels EQ ulevels[i], cnt)
        IF cnt GT 0 THEN result[wexcl] = 0
      ENDIF
    ENDFOR
  ENDIF

  ;; Return everything that's left as an index
  RETURN, Where(result GT 0)

END



FUNCTION get_wgrib_datarecords, metadata, code, filename, datasize

  ;; Extract selected data for 1 code as binary data and read it into
  ;; memory
  w = Where(metadata.code EQ code, cnt)
  IF cnt EQ 0 THEN Message, ' Nanu??? '

  ;; Create inventory file for wgrib
  OpenW, lun, 'tmpinventory~~', /Get_Lun
  FOR i=0L, N_Elements(w)-1 DO BEGIN
    PrintF, lun, metadata[w[i]].text
  ENDFOR
  Free_Lun, lun

  ;; Call wgrib to extract data
  cmd = 'wgrib -i -nh '+filename+' < tmpinventory~~'
  spawn,cmd, result
  spawn,'ls -l dump', result
  cmd = 'rm -f tmpinventory~~'
  spawn, cmd, result

  ;; Read binary data
  data = FltArr(datasize)
  OpenR, lun, 'dump', /Get_Lun, /Binary
  ReadU, lun, data
  Free_Lun, lun

  cmd = 'rm -f dump'
  spawn, cmd, result
```

```
  RETURN, data

END



PRO read_wgrib, data, filename=filename, $
            code=code, date=date, timestep=timestep, level=level, $
            metadata=metadata, dims=dims, $
            header_only=header_only


  ;; Default filename
  IF N_Elements(filename) EQ 0 THEN filename = '*'

  Open_File, filename, lun, filename=truename
  IF lun LE 0 THEN RETURN
  Free_Lun, lun

  ;; Get header information
  inventoryfile = 'inventory~~'
  print,'Analysing file '+truename
  cmd = 'rm -f '+inventoryfile
  spawn, cmd, result

  time0 = Systime(1)
  cmd = 'wgrib '+truename+' > '+inventoryfile
  spawn, cmd, result
  time1 = Systime(1)
  print,'Time to scan GRIB header : ',time1-time0,' secs'

  ;; Analyse header information
  metadata = get_wgrib_header(inventoryfile, truename)
  time2 = Systime(1)
  print,'Time to analyse GRIB header : ',time2-time1,' secs'

  ;; Add extra grid information
  get_wgrib_extrainfo, metadata, truename, inventoryfile
  time3 = Systime(1)
  print,'Time to extract extra grid info : ',time3-time2,' secs'
  time2 = time3

  ;; Get record selection
  index = get_wgrib_selection(metadata, code=code, date=date, $
                              timestep=timestep, level=level)

  time3 = Systime(1)
```

```
  print,'Time to make record selection : ',time3-time2,' secs'

  ;; Write selected metadata back into inventory file
;   IF index[0] GE 0 THEN BEGIN
;     OpenW, lun, inventoryfile, /Get_Lun
;     FOR i=0L, N_Elements(index)-1 DO PrintF, lun, metadata[index[i]].text
;     Free_Lun, lun
;   ENDIF

  IF Keyword_Set(header_only) THEN RETURN

  ;; Extract selected records as binary files without header
  ;; Do this code by code and reform result to yield 3D or 4D array

  ;; Get unique codes of selection
  ucodes = get_wgrib_info(metadata[index], /CODES)
  usteps = get_wgrib_info(metadata[index], /TIMESTEPS)

  ;; Get dimensionality for each code (X, Y, Z, T)
  dims = LonArr(4, N_Elements(ucodes))
  datasize = LonArr(N_Elements(ucodes))
  ;; Enter number of selected time steps
  dims[3,*] = N_Elements(usteps)
  FOR i=0L, N_Elements(ucodes)-1 DO BEGIN
    first = ( Where(metadata.code EQ ucodes[i]) )[0]
    IF metadata[first].nx GT 0 THEN dims[0,i] = metadata[first].nx
    IF metadata[first].ny GT 0 THEN dims[1,i] = metadata[first].ny
    ;; Get number of levels
    w = Where(metadata[index].code EQ ucodes[i] AND $
            metadata[index].timestep EQ usteps[0], cnt)
    dims[2,i] = cnt
    datasize[i] = dims[0,i] * dims[1,i] * dims[2,i] * dims[3,i]
;; print,'Dimensions for code ',ucodes[i],' = ',dims[*,i],' Size = ',datasize[i]
  ENDFOR

  time4 = Systime(1)
  print,'Time to prepare for reading of data : ',time4-time3,' secs'

  ;; Create result structure
  data = { date:get_wgrib_info(metadata[index], /DATES) }
  FOR i=0L, N_Elements(ucodes)-1 DO BEGIN
    IF datasize[i] GT 0 THEN BEGIN
      therecord =  get_wgrib_datarecords(metadata[index],ucodes[i],truename,dat asize[i])
      therecord = Reform(Reform(therecord,dims[*,i]))
      data = Create_Struct(data, 'code'+StrTrim(ucodes[i],2), therecord )
      time5 = Systime(1)
      print,'Time to read records for code ',ucodes[i],' : ',time5-time4,' secs'
      time4 = time5
```

```
    ENDIF
  ENDFOR

  print,'Total time spent in read_wgrib : ',time5-time0,' secs'

END
```