Subject: Re: warping continents to magnetic coordinates
Posted by Brian Jackel on Wed, 23 May 2001 17:09:14 GMT
View Forum Message <> Reply to Message

Hi Ken

I've got some code that you may find useful. It
works in "geomagnetic dipole" coordinates as defined
in Russell or Hapgood. Fairly similar to CGM in
most places.

The basic idea is to pull data directly from the IDL
map files, then apply a coordinate transformation. The
rotation matrix is obtained from some *VERY* rough code
of mine called "geospace_convert". It seems to work
well for this purpose, but please use with caution.

Let me know if you have any trouble with this. It
should just compile and run. Again, this is not
high quality code, but hopefully you'll find it useful.

Brian

```
;1999/08/03      Brian Jackel
;
; This IDL script draws a continental map in dipole geomagnetic
coordinates.
; It takes a few key lines from "map_continents" and does coordinate
transformation
; using "geospace_convert".
;

  name= FILEPATH('clow.ndx',SUBDIR=['resource','maps','low'])
  OPENR,lun,fname,/XDR,/GET_LUN

  nsegments=0L
  READU,lun,nsegments

  result= REPLICATE({fptr:0L, npts:0L, latmax:0.0, latmin:0.0,
lonmax:0.0, lonmin:0.0},nsegments)
  READU,lun,result
  FREE_LUN,lun

map_set,90,-50,79,/GNOMIC,LIMIT=[55,-120,85,190]
re= 6371.2

;
```

```
;

;rotation_matrix= FLTARR(3,3)
;rotation_matrix[0,0]= 1.0
;rotation_matrix[1,1]= 1.0
;rotation_matrix[2,2]= 1.0

  CDF_EPOCH,Epoch,1990,1,1,23,0,0,/COMPUTE_EPOCH
  dummy=
 GEOSPACE_CONVERT([1,1,1],'geo','mag',ROTATION_MATRIX=rotatio n_matrix,
CDF_EPOCH_DATE=epoch)


  fname= FILEPATH('clow.dat',SUBDIR=['resource','maps','low'])
  OPENR,lun,fname,/XDR,/GET_LUN

  FOR indx=0,nsegments-1 DO BEGIN
   IF (result[indx].npts GE 2) THEN BEGIN
    POINT_LUN,lun,result[indx].fptr
    xy= FLTARR(2,result[indx].npts,/NOZERO)
    READU,lun,xy

     R= 1
     lat_rad= REFORM(xy[0,*]) *!dtor
     lon_rad= REFORM(xy[1,*]) *!dtor
     temp= FLTARR(result[indx].npts,3)
     temp(0,0)= R * cos(lat_rad) * cos(lon_rad)
     temp(0,1)= R * cos(lat_rad) * sin(lon_rad)
     temp(0,2)= R * sin(lat_rad)
     temp= rotation_matrix##temp

     l= SQRT( temp[*,0]^2 + temp[*,1]^2 )

     xy[0,*]= ATAN(temp[*,2],l) * !radeg
     xy[1,*]= ATAN(temp[*,1],temp[*,0]) * !radeg

    PLOTS,xy[1,*],xy[0,*],NOCLIP=0,/DATA;,COLOR=128
   ENDIF
  ENDFOR

  FREE_LUN,lun

;!p.font=1
;DEVICE, SET_FONT='Helvetica Bold Italic', /TT_FONT

 MAP_GRID,lats=[50,55,60,65,70,75,80,85],lons=[-35]

;Contwoyto Lake  65.75 248.75  72.64 298.10
```

```
;Fort Simpson  61.76 238.77  67.27 290.66

;Hopen Island  76.51 25.01  71.92 131.66
;Soroya   70.54 22.22  67.30 120.50


;USERSYMBOL,'star',/FILL,SIZE=2
 PLOTS,298.10,72.64,PSYM=5  ;cont
 XYOUTS,298.10,72.64,'Contwoto Lake',CHARSIZE=2
 PLOTS,131.66,71.92,PSYM=5  ;hop
 XYOUTS,131.66,71.92,'Hopen Island',CHARSIZE=2

USERSYMBOL,'triangle',/FILL,SIZE=2
 PLOTS,290.66,67.27,PSYM=5  ;sim
 XYOUTS,290.66,67.27,'Fort Simpson',CHARSIZE=2
 PLOTS,120.50,67.30,PSYM=5  ;sor
 XYOUTS,120.50,67.30,'Soroya',CHARSIZE=2


  ;MAP_GRID,lats=[0,90],lons=[298.1,290.66,131.66,120.5],GLINE STYLE=0

;!p.font=0

END



;+
; NAME:        GEOSPACE_CONVERT
;
; PURPOSE:      Conversion between various geocentric Cartesian
;               space physics coordinate systems.
;
; CATEGORY:     Coordinate systems
;
; CALLING SEQUENCE:     a= GEOSPACE_CONVERT( Position, From, Into )
;
; INPUTS:
;       Position        a 3 element vector or 3xN element array
;                       containing the position(s) to be converted.
;                       Usually in Cartesian coordinates, use
;                       /FROM_SPHERICAL if in spherical coordinates.
;
;       From    the names of the initial and final coordinate
;       Into    systems.  Must be a 2-3 letter string from the set
;                       of 6 allowable coordinate systems:
;
;               GEI     Geocentric equatorial inertial
```

```
;             GEO    Geographic (rotating earth)
;             GSE    Geocentric solar ecliptic
;             GSM    Geocentric solar magnetospheric
;             SM     Solar magnetic
;             MAG    Geomagnetic
;
; KEYWORD PARAMETERS:  Many of the coordinate systems require time
;                 information, which can be provided via the
;                 following keywords.  Note that JD starts at
;                 noon, while UT starts at midnight.  Not my idea.
;
;      JULIAN_DAY     Result = JULDAY(Month, Day, Year)
;      UNIVERSAL_TIME
;
;      CDF_EPOCH
;
; OUTPUTS:          a 3 element vector or 3 x n element array the
;                same size as input Position.  Usually Cartesian,
;                unless the /INTO_SPHERICAL keyword is set, in
;                which case it will be (Radius,Latitude,Longitude)
;
; RESTRICTIONS:
;
; PROCEDURE:    Construct the appropriate rotation matrices, combine
them
;             and apply to the initial coordinates.  For a detailed
;             description, see:
;
;            Space Physics Coordinate Transformations: A User Guide
;            M. A. Hapgood, Planetary and Space Science,
;            Volume 40, Number 5, pages 711-717, 1992
;
;
; MODIFICATION HISTORY:
; September 8 1993      Written by Brian Jackel, University of Western
Ontario
;
;-
 ;------------------------------------------------------------ ------------------

function GEOSPACE_ROTATION,angle,axis
;
;This is a short utility routine for creating rotation matrices.
;
;zeta is rotation angle in degrees
;axis is rotation axis, a string of 'X','Y' or 'Z'
;
;The basic idea is to
```

```
;  start with an empty 3x3 matrix
;  put a 1 in the diagonal element corresponding to the rotation axis
;  put cos(zeta) in the other diagonal elements
;  determine the two off-diagonal terms in the same columns and rows as
the cos(zeta) values
;   put -sin(zeta) in the term above the diagonal
;   put sin(zeta) in the term below
;
;
;element #'s       example for a 'Z' rotation
; 0  1  2         cos(zeta)   -sin(zeta)      0
; 3  4  5         sin(zeta)    cos(zeta)      0
; 6  7  8          0             0          1
;

  sin_zeta= sin(angle*!dtor)
  cos_zeta= cos(angle*!dtor)

  fill= [1.0d0,cos_zeta,cos_zeta,-sin_zeta,sin_zeta]

  rmatrix= fltarr(3,3)
  CASE STRUPCASE(axis) OF
  'X': rmatrix([0,4,8,7,5])= fill
  'Y': rmatrix([4,0,8,6,2])= fill
  'Z': rmatrix([8,0,4,3,1])= fill
  ELSE: MESSAGE,'AXIS parameter must be string of X, Y, or Z'
  ENDCASE

  RETURN,rmatrix
END

 ;------------------------------------------------------------ --------
function GEOSPACE_CONVERT,Position,From,Into,HELP=help, $
                      ROTATION_MATRIX=rotation_matrix, $
                      CDF_EPOCH_DATE=cdf_epoch_date, $
                      YMDHMS_DATE=ymdhms_date, $
;                      JULIAN_DATE=julian_date,
UNIVERSAL_TIME=universal_time, $
                      FROM_SPHERICAL=from_spherical,
INTO_SPHERICAL=into_spherical

 ;ON_ERROR,2

 IF KEYWORD_SET(HELP) THEN BEGIN
 DOC_LIBRARY,'Geospace_Convert'
 return, -1.0
 END
```

```
  IF (N_PARAMS() EQ 0) THEN MESSAGE,"Error- input value Position not
defined"

  siz= SIZE(position)
 ; IF (siz(0) EQ 0) THEN MESSAGE,"Error- input value Position not
defined"
 ; IF (siz(0) GT 2) THEN MESSAGE,"Error- Position must be a 3 element
vector, or a 3xn element array"
 ; IF (siz(1) NE 3) THEN MESSAGE,"Error- Position must be a 3 element
vector, or a 3xn element array"


  _from= STRMID( STRCOMPRESS(STRUPCASE(from),/REMOVE_ALL) ,0,3 )
  _into= STRMID( STRCOMPRESS(STRUPCASE(into),/REMOVE_ALL) ,0,3 )
  IF (_from EQ _into) THEN RETURN,position   ;indentity transformation

;
;
;
  seconds_in_day= 24.0d0 * 60.0d0 * 60.0d0
  seconds_in_century= 36525.0d0 * seconds_in_day


  CDF_EPOCH,epoch2000,2000,01,01,12,/COMPUTE
  CDF_EPOCH,epochunix,1970,01,01,/COMPUTE
  CDF_EPOCH,epochmjd,1858,11,17,/COMPUTE

  IF KEYWORD_SET(CDF_EPOCH_DATE) THEN BEGIN

 CDF_EPOCH,cdf_epoch_date,year,month,day,hour,minute,second,/ BREAKDOWN
    IF (year LT 1970) OR (year GT 2100) THEN MESSAGE,'Note- year
'+STRING(year)+' is outside expected range',/INFORMATIONAL
    epoch= cdf_epoch_date
  ENDIF ELSE IF KEYWORD_SET(YMDHMS_DATE) THEN BEGIN
    ymdhms= ymdhms_date
    CASE N_ELEMENTS(ymdhms) OF
    1:CDF_EPOCH,epoch,ymdhms[0],/COMPUTE
    2:CDF_EPOCH,epoch,ymdhms[0],ymdhms[1],/COMPUTE
    3:CDF_EPOCH,epoch,ymdhms[0],ymdhms[1],ymdhms[2],/COMPUTE
     4:CDF_EPOCH,epoch,ymdhms[0],ymdhms[1],ymdhms[2],ymdhms[3],/C OMPUTE

 5:CDF_EPOCH,epoch,ymdhms[0],ymdhms[1],ymdhms[2],ymdhms[3],ym dhms[4],/COMPUTE

 6:CDF_EPOCH,epoch,ymdhms[0],ymdhms[1],ymdhms[2],ymdhms[3],ym
dhms[4],ymdhms[5],/COMPUTE
    ELSE:MESSAGE,'Error- keyword parameter YMDHMS must have between 1
(just year) and 6 (year, month, day, hour, minute, second) elements'
    ENDCASE
```

```
  ENDIF ELSE BEGIN
    MESSAGE,'Warning- no time provided, using current system
time',/INFORMATIONAL
    st= SYSTIME(1)*1.0d3      ;convert from second to milliseconds
    epoch= epochunix + st
  ENDELSE



; Determine the modified julian date (since 00:00 UT 17 November 1858)
; and time of day in hours.  Then calculate "T_0", the time in julian
; centuries from 12:00 UT on 1 January 2000 (epoch 2000) to the previous
; midnight.
;
  mjdate= (epoch - epochmjd)/1d3                ;modified julian date
in seconds
  utime= (mjdate MOD seconds_in_day) / 3600.0d0    ;time of day in hours
  mjdate= LONG(mjdate / seconds_in_day)           ;modified julian date
in days, truncated to integer
  T_0= (mjdate - 51544.5d0) / 36525.0d0



; If the keyword is set, convert from spherical to
; Cartesian coordinates.  Otherwise just make a
; working copy of the position vector.
;
  new_position= position
  IF KEYWORD_SET(FROM_SPHERICAL) THEN BEGIN
     R= position(0,*)
     lat_rad= position(1,*) *!dtor
     lon_rad= position(2,*) *!dtor
     new_position(0,0)= R * cos(lat_rad) * cos(lon_rad)
     new_position(1,0)= R * cos(lat_rad) * sin(lon_rad)
     new_position(2,0)= R * sin(lat_rad)
  ENDIF


; namelist= ['SM','GEO','GEI','GSE','GSM','SM']
; tlist= [-5,-1,2,3,4]
; w= WHERE


 ;=============================================================
====================================
;Determine what transformation ("T") matrices are required.  This is
just a 6x6
;lookup table, from which we get a sequence of transformations.  The
```

```
number indicates
;the transformation matrix (using Hapgood's convention), with 0 just a
place keeper.
;
 CASE _from OF
  'GEI': matrices= [ [0,0,0],        [1,0,0],        [2,0,0],
[3,2,0],       [4,3,2],        [5,1,0] ]
  'GEO': matrices= [ [-1,0,0,0],     [0,0,0,0],      [2,-1,0,0],
[3,2,-1,0],    [4,3,2,-1],     [5,0,0,0] ]
  'GSE': matrices= [ [-2,0,0],       [1,-2,0],       [0,0,0],
[3,0,0],       [4,3,0],        [5,1,-2] ]
  'GSM': matrices= [ [-2,-3,0,0],    [1,-2,-3,0],    [-3,0,0,0],
[0,0,0,0],     [4,0,0,0],      [5,1,-2,-3] ]
  'SM':  matrices= [ [-2,-3,-4,0,0], [1,-2,-3,-4,0], [-3,-4,0,0,0],
[-4,0,0,0,0], [0,0,0,0,0],    [5,1,-2,-3,-4] ]
  'MAG': matrices= [ [-1,-5,0,0,0],  [-5,0,0,0,0],   [2,-1,-5,0,0],
[3,2,-1,-5,0],  [4,3,2,-1,-5], [0,0,0,0,0] ]
 ELSE:MESSAGE,'Parameter From must be one of GEI, GEO, GSE, GSM, SM,
MAG'
 ENDCASE

 CASE _into OF
  'GEI': matrices= REFORM( matrices(*,0) )
  'GEO': matrices= REFORM( matrices(*,1) )
  'GSE': matrices= REFORM( matrices(*,2) )
  'GSM': matrices= REFORM( matrices(*,3) )
  'SM' : matrices= REFORM( matrices(*,4) )
  'MAG': matrices= REFORM( matrices(*,5) )
 ELSE:MESSAGE,'Parameter Into must be one of GEI, GEO, GSE, GSM, SM,
MAG'
 ENDCASE

 matrices= matrices( WHERE(matrices NE 0) )         ;throw away any
zeros
 ;---------------------------------------------------------- ------------------------------

 ;============================================================
==============================
;This is where any required rotation matrices are actually constructed.
;Make a stack of 6 3x3 matrices, but only calculate the necessary ones.
;Note that some matrices may require partial or full evaluation of
;other matrices, so group some calculations together depending on what
;is required.  The result is a little cryptic, but reasonably efficient.
;


  ;Make a stack for 6 3x3 rotation matrices, only the top five of which
will be used.
```

```
  ;Leave the 0th one empty to keep consistent with notation in Hapgood
  tstack= FLTARR(3,3,6)

  w1= TOTAL( ABS(matrices) EQ 1 )
  w2= TOTAL( ABS(matrices) EQ 2 )
  w3= TOTAL( ABS(matrices) EQ 3 )
  w4= TOTAL( ABS(matrices) EQ 4 )
  w5= TOTAL( ABS(matrices) EQ 5 )

  IF (w1 OR w3 or w4) THEN BEGIN
     theta= 100.461d0 + 36000.770d0*T_0 + 15.04107d0*utime
;Greenwich mean sidereal time
     tstack(0,0,1) = GEOSPACE_ROTATION(theta,'Z')      ;"T1" in Hapgood,
equation 2
  ENDIF

  IF (w2 OR w3 OR w4) THEN BEGIN
     ecliptic_obliquity= 23.439d0 - 0.013d0*T_0
     mean_anomaly= 357.528d0 + 35999.050d0*T_0 + 0.04107d0*utime
     mean_longitude= 280.460d0 + 36000.772d0*T_0 + 0.04107d0*utime
     ecliptic_longitude= mean_longitude + (1.915d0 -
0.0048d0*T_0)*sin(mean_anomaly*!dtor) + 0.020*sin(2*mean_anomaly*!dtor)
     Ez= GEOSPACE_ROTATION(ecliptic_longitude,'Z')
     Ex= GEOSPACE_ROTATION(ecliptic_obliquity,'X')
     tstack(0,0,2) = Ez##Ex
  ENDIF

  IF (w3 or w4 or w5) THEN BEGIN

    dyear= (mjdate-46066.0d0)/365.25    ;should be centered on 1985
    phi=     78.7646 + 0.0351936*dyear -0.000128034*dyear^2
    lambda=  289.097 - 0.0363523*dyear -0.00142802*dyear^2

    IF (w5) THEN BEGIN
       Ez= GEOSPACE_ROTATION(lambda,'Z')
       Ey= GEOSPACE_ROTATION(phi-90.0,'Y')
       tstack(0,0,5) = Ey##Ez
    ENDIF

  ENDIF

  IF (w3 or w4) THEN BEGIN
     phi_rad= phi*!dtor
     cos_phi_rad= cos(phi_rad)
     lambda_rad= lambda*!dtor
     Qg= [ cos_phi_rad*cos(lambda_rad), cos_phi_rad*sin(lambda_rad),
sin(phi_rad) ]
     Qe= REFORM(tstack[*,*,2]) ## ( TRANSPOSE(REFORM(tstack[*,*,1]))##Qg
```

```
)   ;equation (7)

     IF (w3) THEN BEGIN
        psi= ATAN( Qe(1),Qe(2) )*!radeg
        tstack(0,0,3) = GEOSPACE_ROTATION(-psi,'X')
     ENDIF

     IF (w4) THEN BEGIN
        mu= ATAN( Qe(0), SQRT( Qe(1)^2+Qe(2)^2 ) )*!radeg
        tstack(0,0,4)= GEOSPACE_ROTATION(-mu,'Y')
     ENDIF

  ENDIF

 ;
 ;Combine the "T" matrices, do inversions (transpositions) on
 ;those matrices that need it
 ;
  nstack= N_ELEMENTS(matrices)
  T= fltarr(3,3)
  T([0,4,8])= [1.0,1.0,1.0]         ;set diagonal elements equal to 1
(identity matrix)
  FOR i=nstack-1,0,-1 DO BEGIN
     temp= REFORM( tstack(*,*,ABS(matrices(i))) )
     IF ( matrices(i) LT 0 ) THEN temp= TRANSPOSE(temp)
     T= temp##T
  ENDFOR

 ;
 ;Apply rotation
 ;
  new_position= T ## new_position
  rotation_matrix= T
 ;
 ;Maybe convert back from Cartesian into spherical
 ;
  IF KEYWORD_SET(INTO_SPHERICAL) THEN BEGIN
     X= new_position(*,0)
     Y= new_position(*,1)
     Z= new_position(*,2)
     l= SQRT( X^2 + Y^2 )
     new_position(0,0)= SQRT( l^2 + Z^2 )
     new_position(0,1)= ATAN(z,l) * !radeg
     new_position(0,2)= ATAN(y,x) * !radeg  ;Hapgood uses an ArcCos, but
this should work
  ENDIF

 RETURN,new_position
```

```
END

PRO GEOSPACE_CONVERT_TEST

;Formats and units:
;    Day/Time format: YY/MM/DD HH.HHHHH
;    Degrees/Hemisphere format: Decimal degrees with 2 place(s).
;        Longitude 0 to 360, latitude -90 to 90.
;    Distance format: Earth radii with 5 place(s).
;
;polar
;       Time                 GEI (RE)                    GEO
(RE)                      GM (RE)                    GSE
(RE)                     GSM (RE)                     SM (RE)
;yy/mm/dd hh.hhhhh      X         Y         Z         X
Y         Z         X         Y         Z         X
Y         Z         X         Y         Z         X
Y         Z
;
;00/01/01  0.00000    1.89412    1.02594    6.89192    0.68262
-2.04311    6.89192    0.85108    0.00458    7.17039   -3.30372
2.49712    5.91512   -3.30372    0.82850    6.36693   -0.19481
0.82850    7.17039
;00/01/01  4.95000    1.85155    0.17499   -2.04050   -1.82577
-0.35419   -2.04050    0.14029   -1.84442   -2.04962    0.96501
1.70908   -1.94173    0.96501    1.83839   -1.81978   -0.20466
1.83839   -2.04962
;00/01/01  9.90000    7.25976    1.80121    3.82529   -4.29675
6.12262    3.82529   -7.74501   -2.15036    2.44379   -1.82681
7.70989    2.79316   -1.82681    7.99081    1.84148   -0.86959
7.99081    2.44379
;00/01/01 14.85000    5.14900    1.70083    7.19941    3.11398
4.43939    7.19941   -4.50089    4.35341    6.48277   -3.41217
5.86883    5.92878   -3.41217    5.95795    5.83921   -1.92691
5.95795    6.48277
;00/01/01 19.80000   -0.41009    0.40680    5.17012   -0.07490
0.57276    5.17012   -1.50754    0.10925    4.97787   -2.46367
0.04846    4.58167   -2.46367   -0.92680    4.48722   -1.19400
-0.92680    4.97787


;00/01/01  0.00000    1.89412    1.02594    6.89192    0.68262
-2.04311    6.89192    0.85108    0.00458    7.17039   -3.30372
2.49712    5.91512   -3.30372    0.82850    6.36693   -0.19481
0.82850    7.17039
  CDF_EPOCH,epoch,2000,01,01,00,00,00,/COMPUTE
  xgei= [1.89412 ,   1.02594  ,  6.89192 ]
  xgeo= [0.68262 ,  -2.04311  ,  6.89192 ]
```

```
  xmag= [0.85108 ,   0.00458  ,  7.17039  ]
  xgse= [-3.30372 ,   2.49712  ,  5.91512 ]
  xgsm= [-3.30372 ,   0.82850  ,  6.36693 ]
  xsm= [-0.19481 ,   0.82850  ,  7.17039 ]

  PRINT,'MAG to GEO
 ',SQRT(TOTAL((xgeo-GEOSPACE_CONVERT(xmag,'mag','geo',CDF_EPO CH=epoch))^2))
  PRINT,'GEO to GEI
 ',SQRT(TOTAL((xgei-GEOSPACE_CONVERT(xgeo,'geo','gei',CDF_EPO CH=epoch))^2))
  PRINT,'GEI to GSE
 ',SQRT(TOTAL((xgse-GEOSPACE_CONVERT(xgei,'gei','gse',CDF_EPO CH=epoch))^2))
  PRINT,'GSE to GSM
 ',SQRT(TOTAL((xgsm-GEOSPACE_CONVERT(xgse,'gse','gsm',CDF_EPO CH=epoch))^2))
  PRINT,'GSM to SM
 ',SQRT(TOTAL((xsm-GEOSPACE_CONVERT(xgsm,'gsm','sm',CDF_EPOCH =epoch))^2))


  PRINT,'SM to MAG
 ',SQRT(TOTAL((xmag-GEOSPACE_CONVERT(xsm,'sm','mag',CDF_EPOCH =epoch))^2))
  PRINT,'GSM to GEO
 ',SQRT(TOTAL((xgeo-GEOSPACE_CONVERT(xgsm,'gsm','geo',CDF_EPO CH=epoch))^2))

PRINT,'   '

;00/01/01  4.95000    1.85155    0.17499   -2.04050   -1.82577
-0.35419   -2.04050    0.14029   -1.84442   -2.04962    0.96501
1.70908   -1.94173    0.96501    1.83839   -1.81978   -0.20466
1.83839   -2.04962
  CDF_EPOCH,epoch,2000,01,01,04,57,00,/COMPUTE
  xgei= [ 1.85155 ,   0.17499 ,  -2.04050]
  xgeo= [-1.82577 ,  -0.35419 ,  -2.04050]
  xmag= [0.14029 ,  -1.84442 ,  -2.04962 ]
  xgse= [ 0.96501 ,   1.70908 ,  -1.94173]
  xgsm= [ 0.96501 ,   1.83839 ,  -1.81978]
  xsm= [-0.20466  ,  1.83839  , -2.04962 ]

  PRINT,'MAG to GEO
 ',SQRT(TOTAL((xgeo-GEOSPACE_CONVERT(xmag,'mag','geo',CDF_EPO CH=epoch))^2))
  PRINT,'GEO to GEI
 ',SQRT(TOTAL((xgei-GEOSPACE_CONVERT(xgeo,'geo','gei',CDF_EPO CH=epoch))^2))
  PRINT,'GEI to GSE
 ',SQRT(TOTAL((xgse-GEOSPACE_CONVERT(xgei,'gei','gse',CDF_EPO CH=epoch))^2))
  PRINT,'GSE to GSM
 ',SQRT(TOTAL((xgsm-GEOSPACE_CONVERT(xgse,'gse','gsm',CDF_EPO CH=epoch))^2))
  PRINT,'GSM to SM
 ',SQRT(TOTAL((xsm-GEOSPACE_CONVERT(xgsm,'gsm','sm',CDF_EPOCH =epoch))^2))
```

```
  PRINT,'SM to MAG
',SQRT(TOTAL((xmag-GEOSPACE_CONVERT(xsm,'sm','mag',CDF_EPOCH =epoch))^2))
  PRINT,'GSM to GEO
',SQRT(TOTAL((xgeo-GEOSPACE_CONVERT(xgsm,'gsm','geo',CDF_EPO CH=epoch))^2))


RETURN
END
```
