References

Alissandrakis, C. E. "On the Computation of Constant α Force-free Magnetic Field," Astronomy and Astrophysics **100** (1981): 197.

Antiochos, S. K. "The Topology of Force-Free Magnetic Fields and Its Implications for Coronal Activity," The Astrophysical Journal **312** (1987): 884-894.

Arnaud, M., and Raymond, J. "Iron Ionization and Recombination Rates and Ionization Equilibrium," The Astrophysical Journal **398** (1992): 394.

Bracewell, R. N. The Fourier Transform and Its Applications. 2nd ed. Singapore: McGraw-Hill, 1986.

Cuperman, S., Ofman, L., and Semel, M. "Determination of constant-α force-free magnetic field above the photosphere using three-component boundary conditions," Astronomy and Astrophysics **216** (1989): 265-277.

Cuperman, S., Ofman, L., and Semel, M. "The absolute value and sign of the function $\alpha(r)$ in the force-free magnetic modelling of photospheric observations," Astronomy and Astrophysics **227** (1990): 227-234.

Emslie, A. G., and Nagai, F. "Gas Dynamics in the Impulsive Phase of Solar Flares II. The Structure of the Transition Region–A Diagnostic of Energy Transport Processes," The Astrophysical Journal **288** (1985): 85.

Foukal, P. Solar Astrophysics. New York: John Wiley & Sons, 1990.

Hanaoka, Y., Kurokawa, H., and Saito, S. "The Post Flare Loops Observed at the Total Eclipse of February 16, 1980," Solar Physics **105** (1986): 133.

Hanaoka, Y., Kurokawa, H., and Saito, S. "Active Region Coronal Loops Observed at the Total Solar Eclipse of February 16, 1980," Publ. Astron. Soc. Japan **40** (1988): 369-382.

Hood, A. W., and Priest, E. R. "The Equilibrium of Solar Coronal Magnetic Loops," Astronomy and Astrophysics **77** (1979): 233-251.

Jackson, J. D. Classical Electrodynamics. 2nd ed. New York: John Wiley & Sons, 1975.

Krall, K. R. "The Effect of Nonlinear Conduction on the Cooling of Flare Loops," Solar Physics **55** (1977): 455-458.

Kurokawa, H., Kitai, R., and Ishiura, K. "The Observation of the Inner Corona at the 1991 Total Solar Eclipse," (1995).

Machado, M. E., and Emslie, A. G. "A Comparison of High-Temperature Flare Models with Observations and Implications for the Low-Temperature Flare," The Astrophysical Journal **232** (1979): 903-914.

Press, W. H., Flannery, B. P., Teukolsky, S. A., and Vetterling, W. T. Numerical Recipes in C. New York: Cambridge University Press, 1988.

Press, W. H., Flannery, B. P., Teukolsky, S. A., and Vetterling, W. T. Numerical Recipes in C. 2nd ed. New York: Cambridge University Press, 1992.

Priest, E. R. "Magnetohydrodynamics," in Plasma Astrophysics, by Kirk, J. G., Melrose, D. B., and Priest, E. R. Heidelberg: Springer-Verlag, 1994, pp. 1-36.

Ruffolo, D. "Observations of the Total Solar Eclipse of 1995 October 24 and a Map of Emission from Fe^{+9} in the Solar Corona"(in Thai), Science (Thailand), **50** (1996): 20-26.

Rust, D. M., Simnett, G. M., and Smith, D. F. "Observational Evidence for Thermal Wave Fronts in Solar Flares," Astrophysical Journal **288** (1985): 401.

Semel, M. "Extrapolation function for constant-α force-free field. Green's method for the oblique boundary value," Astronomy and Astrophysics **198** (1988): 293-299.

Wu, S. T., Sun, M. T., Change, H. M., Hagyard, M. T., and Gary, G. A. "On the Numerical Computation of Nonlinear Force-free Magnetic Fields," The Astrophysical Jornal **362** (1990): 698-708.

Appendix A

Energy Levels of Fe^{+9}

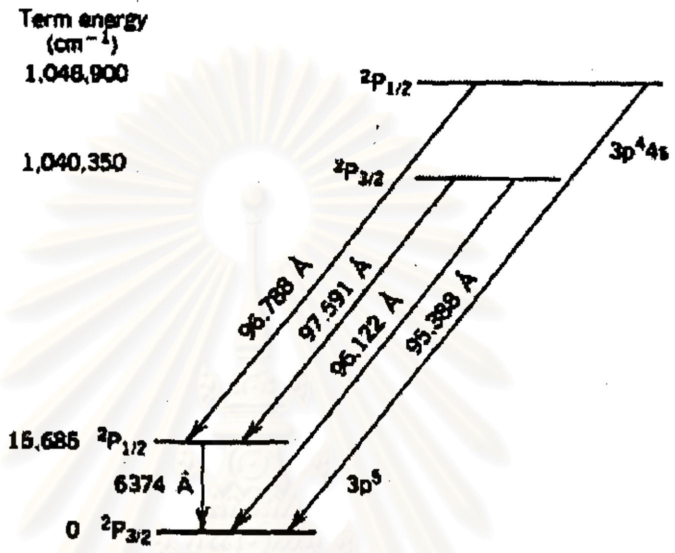

Figure A.1: Term splitting in the ground state of Fe^{+9} giving rise to the λ 6374 forbidden coronal line (Foukal, 1990).

The transitions that do not obey standard selection rules are called forbidden. The vertical line in the energy-level diagram shows a transition between the fine structure levels within the ground terms of highly ionized iron. The red line λ 6374 arises from a transition between the $^2P_{1/2}$ and $^2P_{3/2}$ fine structure levels of the ground state of Fe^{+9} (see Figure A.1). Fe^{+9} can be found at the temperature of about 10^6 K (Arnaud and Raymond, 1992).

Appendix B

Minimization by Powell's Methods[1]

If we start at a point **P** in N-dimensional space, and proceed from there in some vector direction **n**, then any function of N variables $f(\mathbf{P})$ can be minimized along the line **n** by our one-dimensional methods. One can dream up various multidimensional minimization methods that consist of sequences of such line minimizations. Different methods will differ only by how, at each stage, they choose the next direction **n** to try. All such methods presume the existence of a "black-box" sub-algorithm, which we might call linmin, whose definition can be taken for now as

linmin: Given as input the vectors **P** and **n**, and the function f, find the scalar λ that minimizes $f(\mathbf{P}+\lambda\mathrm{N})$. Replace **n** by $\lambda\mathbf{n}$. Done.

All the minimization methods in this section fall under this general schema of successive line minimizations. In this section we consider a class of methods whose choice of successive directions does not involve explicit computation of the function's gradient. You will note that we need not specify whether linmin uses gradient information or not; that choice is up to you, and its optimization depends on your particular function. You would be crazy, however, to use gradients in linmin and *not* use them in the choice of directions, since in this latter role they can drastically reduce the total computational burden.

[1]from Press et al. (1988)

But what if, in your application, calculation of the gradient is out of the question. You might first think of this simple method: Take the unit vectors $\mathbf{e}_1$, $\mathbf{e}_2, \ldots \mathbf{e}_N$ as a *set of directions.* Using linmin, move along the first direction to its minimum; then *from there* along the second direction to *its* minimum, and so on, cycling through the whole set of directions as many times as necessary, until the function stops decreasing.

This simple method, or the related "steepest descent" method, is actually not too bad for many functions. Even more interesting is why it *is* bad, i.e., very inefficient, for some other functions. Consider a function of two dimensions whose contour map (level lines) happens to define a long, narrow valley at some angle to the coordinate basis vectors (see Figure B.1). Then the only way "down the length of the valley" going along the basis vectors at each stage is by a series of many tiny steps. More generally, in N dimensions, if the function's second derivatives are much larger in magnitude in some directions than in others, then many cycles through all N basis vectors will be required in order to get anywhere. This condition is not all that unusual; according to Murphy's Law, you should count on it.

Obviously what we need is a better set of directions than the $\mathbf{e}_i$'s. All *directions* consist of prescriptions for updating the set of directions as the method proceeds, attempting to come up with a set which either (i) includes some very good directions that will take us far along narrow valleys, or else (more subtly) (ii) includes some number of "non-interfering" directions with the special property that minimization along on is not "spoiled" by subsequent minimization along another, so that interminable cycling through the set of directions can be avoided.

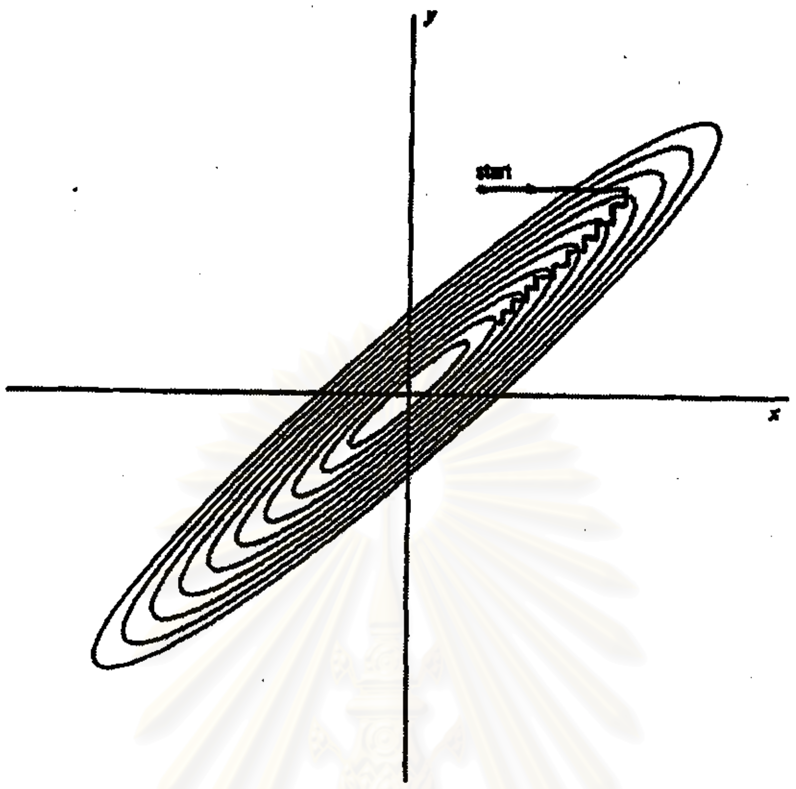

Figure B.1: Successive minimizations along coordinate directions (the "steepest-descent" method) in a long, narrow "valley" (shown as contour lines). Unless the valley is optimally oriented, this method is extremely inefficient, taking many tiny steps to get to the minimum, crossing and re-crossing the principal axis (Press et al., 1988).

powell.c

It contains the method described in previous paragraphs (adapted from Numerical Recipes in C).

nrutil.c

It contains routines to reserve and unreserve the memory for arrays (adapted from Numerical Recipes in C).

The program:

powell.c

```
/*  Copied from "Numerical Recipes in C" */

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

#define   BITMAX 10000
#define   CGOLD 0.3819660
#define   GLIMIT 100.0
#define   GOLD 1.618034
#define   ITMAX 200
#define   MAX(a,b) ((a) > (b) ? (a) : (b))
#define   SIGN(a,b) ((b) > 0.0 ? fabs(a) : -fabs(a))
#define   SHFT(a,b,c,d) (a)=(b); (b)=(c); (c)=(d);
static double squarg;
#define   SQR(a) (squarg =(a), squarg*squarg)
#define   TINY 1.0e-20
#define   TOL   1.0e-6
#define   ZEPS 1.0e-10

int ncom=0;
double *pcom=0, *xicom=0, (*nrfunc)();

void powell(p,xi,n,ftol,iter,fret,func)
double p[], **xi, ftol, *fret, (*func)();
int n, *iter;
{
   int     i, ibig, j;
   double  t, fptt, fp, del;
   double  *pt, *ptt, *xit, *dvector();
   void    linmin(), nrerror(), free_dvector();

      pt = dvector(1,n);
     ptt = dvector(1,n);
     xit = dvector(1,n);
   *fret = (*func)(p);

   for(j=1;j<=n;j++)  pt[j]=p[j];
   for(*iter=1;;(*iter)++) {
      fp=(*fret);
      ibig=0;
      del=0.0;
      for(i=1;i<=n;i++)  {
         for(j=1;j<=n;j++) xit[j]=xi[j][i];
         fptt=(*fret);
         linmin(p,xit,n,fret,func);
         for(j=1;j<=n;j++) printf("\nx[%d] = %17.12lf",j,p[j]);
         printf("\n");
         if(fabs(fptt-(*fret)) > del) {
            del=fabs(fptt-(*fret));
            ibig=i;
         }
      }
      if(2.0*fabs(fp-(*fret)) <= ftol*(fabs(fp)+fabs(*fret))){
         free_dvector(xit,1,n);
         free_dvector(ptt,1,n);
         free_dvector(pt,1,n);
         return;
      }
      if(*iter==ITMAX)
         nrerror("Warning!!! too many iterations in POWELL");
      for(j=1;j<=n;j++)  {
         ptt[j]=2.0*p[j]-pt[j];
         xit[j]=p[j]-pt[j];
         pt[j]=p[j];
      }
      fptt=(*func)(ptt);
      if(fptt < fp) {
         t=2.0*(fp-2.0*(*fret)+fptt)*SQR(fp-(*fret)-del)
           -del*SQR(fp-fptt);
         if(t < 0.0) {
            linmin(p,xit,n,fret,func);
            for(j=1;j<=n;j++) xi[j][ibig]=xit[j];
         }
      }
   }
}

void linmin(p,xi,n,fret,func)
double p[], xi[], *fret, (*func)();
int n;
{
   int j;
   double xx, xmin, fx, fb,fa, bx,ax;
   double brent(), f1dim(), *dvector();
   void mnbrak(), free_dvector();

   ncom=n;
   pcom=dvector(1,n);
   xicom=dvector(1,n);
   nrfunc=func;
   printf("l\t");
   for(j=1;j<=n;j++) {
      pcom[j]=p[j];
      xicom[j]=xi[j];
   }
   ax=0.0;
   xx=1.0;
   bx=2.0;
   mnbrak(&ax,&xx,&bx,&fa,&fx,&fb,f1dim);
   *fret=brent(ax,xx,bx,f1dim,TOL,&xmin);
   for(j=1;j<=n;j++) {
      xi[j]*=xmin;
      p[j]+=xi[j];
   }
   free_dvector(xicom,1,n);
   free_dvector(pcom,1,n);
}

double brent(ax,bx,cx,f,tol,xmin)
double ax,bx,cx,tol,*xmin;
double (*f)();
{
   int iter;
   double a,b,d,etemp,fu,fv,fw,fx,p,q,r,tol1,tol2,u,v,w,x,xm;
   double e=0.0;
   void nrerror();
   printf("b\t");
   a=((ax < cx) ? ax : cx);
   b=((ax > cx) ? ax : cx);
   x=w=v=bx;
   fw=fv=fx=(*f)(x);
   for(iter=1;iter <= BITMAX;iter++) {
      xm=0.5*(a+b);
      tol2=2.0*(tol1=tol*fabs(x)+ZEPS);
      if(fabs(x-xm) <= (tol2-0.5*(b-a)))  {
         *xmin=x;
         return fx;
      }
      if(fabs(e) > tol1) {
         r=(x-w)*(fx-fv);
         q=(x-v)*(fx-fw);
         p=(x-v)*q-(x-w)*r;
         q=2.0*(q-r);
         if(q > 0.0) p =  -p;
         q = fabs(q);
         etemp=e;
         e=d;
         if(fabs(p)>=fabs(0.5*q*etemp)||p<=q*(a-x)||p>=q*(b-x))
            d=CGOLD*(e=(x >= xm ? a-x: b-x));
         else {
            d=p/q;
            u=x+d;
            if (u-a < tol2 || b-u < tol2)
               d=SIGN(tol1,xm-x);
         }
      } else {
         d=CGOLD*(e=(x >= xm ? a-x : b-x));
      }
      u=(fabs(d) >= tol1 ? x+d: x+SIGN(tol1,d));
      fu=(*f)(u);
      if(fu <= fx) {
         if(u >= x) a=x; else b=x;
         SHFT(v,w,x,u)
         SHFT(fv,fw,fx,fu)
      } else {
         if(u < x) a=u; else b=u;
         if(fu <= fw || w == x) {
            v=w;
            w=u;
            fv=fw;
            fw=fu;
         } else if (fu <= fv || v == x || v == w) {
            v=u;
            fv=fu;
         }
      }
   }
   nrerror("Warning!!! too many iterations in BRENT");
   *xmin = x;
   return fx;
}

void mnbrak(ax,bx,cx,fa,fb,fc,func)
double *ax, *bx, *cx, *fa, *fb, *fc;
double (*func)();
/*
   Given a function func, and given distinct initial points ax and bx,
   this routine searches in the downhill direction (defined by the
   function as evaluated at the initial points) and returns new points
   ax, bx, cx which bracket a minimum of the minimum of the function.
   Also returned are the function values at the three points, fa, fb
   and fc.
*/
```

```
{
   double ulim,u,r,q,fu,dum;

   *fa=(*func)(*ax);
   *fb=(*func)(*bx);

   if(*fb > *fa) {
      SHFT(dum,*ax,*bx,dum)
      SHFT(dum,*fb,*fa,dum)
   }
   *cx=(*bx)+GOLD*(*bx - *ax);
   *fc=(*func)(*cx);
   while(*fb > *fc) {
      r=(*bx - *ax) * (*fb - *fc);
      q=(*bx - *cx) * (*fb - *fa);
      u=(*bx)-((*bx - *cx) * q - (*bx - *ax) * r)/
      (2.0*SIGN(MAX(fabs(q-r),TINY),q-r));
      ulim=(*bx)+GLIMIT*(*cx - *bx);

      printf("mnbrak: entering big loop...\n");

      if((*bx - u)*(u - *cx) > 0.0) {
         fu=(*func)(u);
         if(fu < *fc) {
            *ax=(*bx);
            *bx=u;
            *fa=(*fb);
            *fb=fu;
            return;
         } else if (fu > *fb) {
            *cx=u;
            *fc=fu;
            return;
         }
         u=(*cx)+GOLD*(*cx - *bx);
         fu=(*func)(u);
      } else if((*cx-u)*(u-ulim) > 0.0 ) {
         fu=(*func)(u);
         if(fu < *fc) {
            SHFT(*bx,*cx,u, *cx+GOLD*(*cx - *bx))
            SHFT(*fb,*fc,fu,(*func)(u))
         }
      } else if((u - ulim)*(ulim - *cx) >= 0.0) {
         u=ulim;
         fu=(*func)(u);
      } else {
         u=(*cx)+GOLD*(*cx - *bx);
         fu=(*func)(u);
      }
      SHFT(*ax,*bx,*cx,u)
      SHFT(*fa,*fb,*fc,fu)
   }
}

   /* THESE ARE UTILITIES COPIED FROM THE BOOK,     */
   /* NUMERICAL RECIPES IN C                         */
   /* This is the correct version for nrutil.c       */
   /* Last update   01-MAY-2536                      */

#include <stdio.h>

/* #include <malloc.h> -- not understood by VMS at Bartol */
/*      @(#)malloc.h 1.4 88/02/07 SMI; from S5R2 1.2    */

/*
        Constants defining mallopt operations
*/
#define M_MXFAST 1 /* set size of 'small blocks' */
#define M_NLBLKS 2 /* set num of small blocks in holding block */
#define M_GRAIN  3 /* set rounding factor for small blocks */
#define M_KEEP   4 /* (nop) retain contents of freed blocks */

/*
   malloc information structure
*/
struct mallinfo  {
   int arena;    /* total space in arena */
   int ordblks; /* number of ordinary blocks */
   int smblks;  /* number of small blocks */
   int hblks;   /* number of holding blocks */
   int hblkhd;  /* space in holding block headers */
   int usmblks; /* space in small blocks in use */
   int fsmblks; /* space in free small blocks */
   int uordblks; /* space in ordinary blocks in use */
   int fordblks; /* space in free ordinary blocks */
   int keepcost; /* cost of enabling keep option */

   int mxfast; /* max size of small blocks */
   int nlblks; /* number of small blocks in a holding block */
   int grain;  /* small block rounding factor */
   int uordbytes; /* space (including overhead) allocated
                     in ord. blks */
   int allocated; /* number of ordinary blocks allocated */
   int treeoverhead; /* bytes used in maintaining the free tree */
};

extern  char    *malloc();
extern  char    *realloc();
extern  int     mallopt();
extern  struct mallinfo mallinfo();

void nrerror(error_text)
char error_text[];
{
   void exit();

   fprintf(stderr,"Numerical Recipes run-time error...\n");
   fprintf(stderr,"%s\n",error_text);
   fprintf(stderr,"...now exiting to system...\n");
   exit(1);
}

float *vector(nl,nh)
int nl,nh;
{
   float *v;

   v=(float *)malloc((unsigned)(nh-nl+1)*sizeof(float));
   if(!v) nrerror("allocation failure in vector()");
   return v-nl;
}
```

```
int *ivector(nl,nh)
int nl,nh;
{
   int *v;

   v=(int *)malloc((unsigned)(nh-nl+1)*sizeof(int));
   if(!v) nrerror("allocation failure in ivector()");
   return v-nl;
}

double *dvector(nl,nh)
int nl,nh;
{
   double *v;

   v=(double *)malloc((unsigned)(nh-nl+1)*sizeof(double));
   if(!v) nrerror("allocation failure in dvector()");
   return v-nl;
}

float **matrix(nrl,nrh,ncl,nch)
int nrl,nrh,ncl,nch;
{
   int i;
   float **m;

   m=(float **)malloc((unsigned)(nrh-nrl+1)*sizeof(float*));
   if(!m) nrerror("allocation failure 1 in matrix()");
   m -= nrl;

   for(i=nrl;i<=nrh;i++) {
      m[i]=(float *)malloc((unsigned)(nch-ncl+1)*sizeof(float));
      if(!m[i]) nrerror("allocation failure 2 in matrix()");
      m[i] -= ncl;
   }
   return m;
}

double **dmatrix(nrl,nrh,ncl,nch)
int nrl,nrh,ncl,nch;
{
   int i;
   double **m;

   m=(double **)malloc((unsigned)(nrh-nrl+1)*sizeof(double*));
   if(!m) nrerror("allocation failure 1 in dmatrix()");
   m -= nrl;

   for(i=nrl;i<=nrh;i++) {
      m[i]=(double *)malloc((unsigned)(nch-ncl+1)*sizeof(double));
      if(!m[i]) nrerror("allocation failure 2 in dmatrix()");
      m[i] -= ncl;
   }
   return m;
}

int **imatrix(nrl,nrh,ncl,nch)
int nrl,nrh,ncl,nch;
{
   int i;
   int **m;
```

```
    m=(int **)malloc((unsigned)(nrh-nrl+1)*sizeof(int*));
    if(!m) nrerror("allocation failure 1 in imatrix()");
    m -= nrl;

    for(i=nrl;i<=nrh;i++) {
       m[i]=(int *)malloc((unsigned)(nch-ncl+1)*sizeof(int));
       if(!m[i]) nrerror("allocation failure 2 in imatrix()");
       m[i] -= ncl;
    }
    return m;
}

float **submatrix(a,oldrl,oldrh,oldcl,oldch,newrl,newcl)
float **a;
int oldrl,oldrh,oldcl,oldch,newrl,newcl;
{
    int i,j;
    float **m;

    m=(float **)malloc((unsigned)(oldrh-oldrl+1)*sizeof(float*));
    if(!m) nrerror("allocation failure in submatrix()");
    m -= newrl;
    for(i=oldrl,j=newrl;i<=oldrh;i++,j++) m[j]=a[i]+oldcl-newcl;
    return m;
}

void free_vector(v,nl,nh)
float *v;
int nl,nh;
{
    free((char*) (v+nl));
}

void free_ivector(v,nl,nh)
int *v,nl,nh;
{
    free((char*) (v+nl));
}

void free_dvector(v,nl,nh)
double *v;
int nl,nh;
{
    free((char*) (v+nl));
}

void free_matrix(m,nrl,nrh,ncl,nch)
float **m;
int nrl,nrh,ncl,nch;
{
    int i;

    for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
    free((char*) (m+nrl));
}

void free_dmatrix(m,nrl,nrh,ncl,nch)
double **m;
int nrl,nrh,ncl,nch;
{
    int i;
```

```
   for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
   free((char*) (m+nrl));
}

void free_imatrix(m,nrl,nrh,ncl,nch)
int **m;
int nrl,nrh,ncl,nch;
{
   int i;

   for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
   free((char*) (m+nrl));
}

void free_submatrix(b,nrl,nrh,ncl,nch)
float **b;
int nrl,nrh,ncl,nch;
{
   free((char*) (b+nrl));
}

float **convert_matrix(a,nrl,nrh,ncl,nch)
/* for a use &a[0][0] */
float *a;
int nrl,nrh,ncl,nch;
{
   int i,j,nrow,ncol;
   float **m;

   nrow=nrh-nrl+1;
   ncol=nch-ncl+1;
   m=(float **)malloc((unsigned)(nrow)*sizeof(float*));
   if(!m) nrerror("allocation failure in convert_matrix()");
   m -= nrl;
   for(i=0,j=nrl;i<=nrow-1;i++,j++) m[j]=a+ncol*i-ncl;
   return m;
}

void free_convert_matrix(b,nrl,nrh,ncl,nch)
float **b;
int nrl,nrh,ncl,nch;
{
   free((char*) (b+nrl));
}
```

nrutil.c

```
/* nrutil_v.c (t) -- February 8, 1996

      Minor fix to free_dmatrix(), probably does not
      affect performance.

      Added darray2().

   nrutil_v.c (t) -- July 5th, 1995

      Added lvector().

   nrutil_v.c -- March 13th, 1994

      imatrix(), free_imatrix() removed.

   nrutil.v.c -- February 19th, 1994
```

```
      Modified to accomodate long upper limits,
      dmatrix and imatrix added.

   September 7th, 1992 -- nrutil.w.c

      A subroutine for wind, mostly from Numerical Recipes in C.

      Added malloc.h.

   September 5th, 1992 -- nrutil.w.c

      Unused routines removed.  Added the new routines,
      ***darray and free_darray for a three-dimensional array.

   June 15th, 1999 -- nrutil.w.c

      Add the new routine **dmatrix2(), **free_dmatrix2() for a
      two-dimensional array.

      David Ruffolo
      Department of Physics
      Faculty of Science
      Chulalongkorn University
      Bangkok 10330, Thailand
*/

     /* THESE ARE UTILITIES COPIED FROM THE BOOK,    */
     /* NUMERICAL RECIPES IN C                       */

#include <stdio.h>

/* #include <malloc.h> -- not understood by VMS at Bartol */
/* @(#)malloc.h 1.4 88/02/07 SMI; from S5R2 1.2 */

/*
   Constants defining mallopt operations
*/

/* set size of 'small blocks' */
/* set num of small blocks in holding block */
/* set rounding factor for small blocks */
/* (nop) retain contents of freed blocks */

/*
 * comment the following, because they are redefined in malloc.h.

#define M_MXFAST 1
#define M_NLBLKS 2
#define M_GRAIN 3
#define M_KEEP 4
*/

/*
   malloc information structure
*/
   /* total space in arena */
   /* number of ordinary blocks */
   /* number of small blocks */
   /* number of holding blocks */
   /* space in holding block headers */
   /* space in small blocks in use */
   /* space in free small blocks */
   /* space in ordinary blocks in use */
   /* space in free ordinary blocks */
   /* cost of enabling keep option */

   /* max size of small blocks */
   /* number of small blocks in a holding block */
   /* small block rounding factor */
   /* space (including overhead) allocated in ord. blks */
   /* number of ordinary blocks allocated */
   /* bytes used in maintaining the free tree */

/*
struct mallinfo  {
   int arena;
   int ordblks;
   int smblks;
   int hblks;
   int hblkh;
   int usmblks;
   int fsmblks;
   int uordblks;
   int fordblks;
   int keepcost;

   int mxfast;
   int nlblks;
   int grain;
   int uordby;
   int allocated;
   int treeoverhead;
};
*/

extern char *malloc();
extern char *realloc();
extern int mallopt();
extern struct mallinfo mallinfo();

void nrerror(error_text)
char error_text[];
{
   void exit();

   fprintf(stderr,"Numerical Recipes run-time error...\n");
   fprintf(stderr,"%s\n",error_text);
   fprintf(stderr,"...now exiting to system...\n");
   exit(1);
}

double *dvector(nl,nh)
int nl,nh;
{
   double *v;

   v=(double *)malloc((unsigned)(nh-nl+1)*sizeof(double));
   if(!v) nrerror("allocation failure in dvector()");
   return v-nl;
}

long *lvector(nl,nh)
```

```
int nl,nh;
{
   long *v;

   v=(long *)malloc((unsigned)(nh-nl+1)*sizeof(long));
   if (!v) nrerror("allocation failure in lvector()");
   return v-nl;
}

double **dmatrix(nrl,nrh,ncl,nch)
int nrl,ncl,nch;
long nrh;
{
   long i;
   double **m;

   m=(double **)malloc((unsigned)(nrh-nrl+1)*sizeof(double*));
   if(!m) nrerror("allocation failure 1 in dmatrix()");
   m -= nrl;

   for(i=nrl;i<=nrh;i++) {
      m[i]=(double *)malloc((unsigned)(nch-ncl+1)*sizeof(double));
      if(!m[i]) nrerror("allocation failure 2 in dmatrix()");
      m[i] -= ncl;
   }
   return m;
}

double **dmatrix2(nrl,nrh,ncl,nch)
int nrl,ncl,nch,nrh;
{
   int i;
   double **m;

   m=(double **)malloc((unsigned)(nrh-nrl+1)*sizeof(double*));
   if(!m) nrerror("allocation failure 1 in dmatrix()");
   m -= nrl;

   for(i=nrl;i<=nrh;i++) {
      m[i]=(double *)malloc((unsigned)(nch-ncl+1)*sizeof(double));
      if(!m[i]) nrerror("allocation failure 2 in dmatrix()");
      m[i] -= ncl;
   }
   return m;
}

double ***darray(n1l,n1h,n2l,n2h,n3l,n3h)
int n1l,n1h,n2l,n3l,n3h;
long *n2h;
{
   int i;
   long j;
   double ***a;

   a=(double ***)malloc((unsigned)(n1h-n1l+1)*sizeof(double**));
   if(!a) nrerror("allocation failure 1 in darray()");
   a -= n1l;

   for(i=n1l;i<=n1h;i++) {
      a[i]=(double **)malloc((unsigned)(n2h[i]-n2l+1)*sizeof(double*));
      if(!a[i]) nrerror("allocation failure 2 in darray()");
      a[i] -= n2l;
   }

   for(i=n1l;i<=n1h;i++) {
      for(j=n2l;j<=n2h[i];j++) {
         a[i][j]=(double *)malloc((unsigned)(n3h-n3l+1)*sizeof(double));
         if(!a[i][j]) nrerror("allocation failure 3 in darray()");
         a[i][j] -= n3l;
      }
   }
   return a;
}

double ***darray2(n1l,n1h,n2l,n2h,n3l,n3h)
int n1l,n1h,n2l,n2h,n3l,n3h;
{
   int      i, j;
   double   ***a;

   a=(double ***)malloc((unsigned)(n1h-n1l+1)*sizeof(double**));
   /*if(!a) nrerror("allocation failure 1 in darray()");*/
   a -= n1l;

   for(i=n1l;i<=n1h;i++) {
      a[i]=(double **)malloc((unsigned)(n2h-n2l+1)*sizeof(double*));
      /* if(!a[i]) nrerror("allocation failure 2 in darray()"); */
      a[i] -= n2l;
   }

   for(i=n1l;i<=n1h;i++) {
      for(j=n2l;j<=n2h;j++) {
         a[i][j]=(double *)malloc((unsigned)(n3h-n3l+1)*sizeof(double));
         /*if(!a[i][j]) nrerror("allocation failure 3 in darray()");*/
         a[i][j] -= n3l;
      }
   }
   return a;
}

void free_dvector(v,nl)
double *v;
int nl;
{
   free((char*) (v+nl));
}

void free_lvector(v,nl)
int  nl;
long *v;
{
   free((char*) (v+nl));
}

void free_dmatrix(m,nrl,nrh,ncl)
double **m;
int nrl,ncl;
long nrh;
{
   long i;

   for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
   free((char*) (m+nrl));
}

void free_dmatrix2(m,nrl,nrh,ncl)
double **m;
int nrl,ncl,nrh;
{
   int i;

   for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
   free((char*) (m+nrl));
}

void free_darray(a,n1l,n1h,n2l,n2h,n3l)
double ***a;
int n1l,n1h,n2l,n3l;
long *n2h;
{
   int   i;
   long  j;

   for(i=n1h;i>=n1l;i--)
      for(j=n2h[i];j>=n2l;j--)
         free((char*) (a[i][j]+n3l));
   for(i=n1h;i>=n1l;i--) free((char*) (a[i]+n2l));
   free((char*) (a+n1l));
}

void free_darray2(a,n1l,n1h,n2l,n2h,n3l)
double ***a;
int n1l,n1h,n2l,n2h,n3l;
{
   int i, j;

   for(i=n1h;i>=n1l;i--)
      for(j=n2h;j>=n2l;j--)
         free((char*) (a[i][j]+n3l));
   for(i=n1h;i>=n1l;i--) free((char*) (a[i]+n2l));
   free((char*) (a+n1l));
}
```

Appendix C

Linear Interpolation Program[1]

polin.c

This includes subroutines for linearly interpolating values in 1, 2, or 3 dimensions (see sections 4.5.1 - 4.5.3). These are in functions: Interpolin2d(), Interpolin3d(), polint1(), polint2(), and polint3().

The program:

```
/*
    27th January, 1999 --> polin.c
*/

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

double Interpolin2d(ndim,nn,xin,yin,data)
int    nn[], ndim;
double xin, yin, data[];
{
   int      i, j, k, x, y, z, no;
   void     polint2();
   double   x0tmp, y0tmp, deltas;
   double   xa[3], ya[3], D_pt;
   double   Dtmp[5], D[nn[1]+1][nn[2]+1];

/* Save data in parameter 2 dimentions. */
   for(j=1;j<=nn[2];j++) {
      for(i=1;i<=nn[1];i++) {
         D[i][j] = data[nn[1]*(j-1)+i];
      }
   }

   x0tmp = xin; /* Keep data */
   y0tmp = yin;
   i = x0tmp;
   j = y0tmp;

/* Check the boundary in 2 directions in each case in if command.
   If it's not in this program will out of this program.  */

   if(i<0 || i>(nn[1]-1) || j<0 || j>(nn[2]-1)) {
   /* Check data will is in the boundaries. */
```

[1]written by the author

```
      printf("\nOut of 2 dimensions boundarys.");
      exit(0);
   } else {
      if((i==nn[1]-1) && (j==nn[2]-1)) {
         /* At the two edges of boundaries. */
         for(y=1;y<=2;y++) {
            xa[y] = (i-1)+y-1.0;
            ya[y] = (j-1)+y-1.0;
         }
         for(y=1;y<=2;y++) {
            for(x=1;x<=2;x++) Dtmp[2*(y-1)+x] = D[(i-1)+x][(j-1)+y];
         }
      } else if(i==nn[1]-1 && j<nn[2]-1) {
        /* At the edge of x-boundary. */
         for(y=1;y<=2;y++) {
            xa[y] = (i-1)+y-1.0;
            ya[y] = j+y-1.0;
         }
         for(y=1;y<=2;y++) {
            for(x=1;x<=2;x++) Dtmp[2*(y-1)+x] = D[(i-1)+x][j+y];
         }
      } else if(i<nn[1]-1 && j==nn[2]-1) {
         for(y=1;y<=2;y++) {
            xa[y] = i+y-1.0;
            ya[y] = (j-1)+y-1.0;
         }
         for(y=1;y<=2;y++) {
            for(x=1;x<=2;x++) Dtmp[2*(y-1)+x] = D[i+x][(j-1)+y];
         }
      } else {
         for(y=1;y<=2;y++) {
            xa[y] = i+y-1.0;
            ya[y] = j+y-1.0;
         }
         for(y=1;y<=2;y++) {
            for(x=1;x<=2;x++) Dtmp[2*(y-1)+x] = D[i+x][j+y];
         }
      }
      polint2(xin,yin,xa,ya,Dtmp,&D_pt);
   }
   return D_pt;
}

double Interpolin3d(ndim,nn,xin,yin,zin,step,data)
int    ndim, nn[];
double xin, yin, zin, step[], data[];
/*
   xin, yin, zin = real number of positions.
   ndim = number of dimension = 3.
   nn[] = number of data in each directions.
   step[] = step size in each dimension.
   data = data values in each coordinate.
*/
{
   int     i, j, k, x, y, z;
   int     itmp, jtmp, ktmp;
   void    polint3();
   double  ido, jdo, kdo, Lx, Ly, Lz;
   double  *xa, *ya, *za, *Dtmp;
double  D_PT, ***D;
double  *dvector(), ***darray2();

/* Begin program. */
D = darray2(1,nn[1],1,nn[2],1,nn[3]);
Dtmp = dvector(1,8);
xa = dvector(1,2);
ya = dvector(1,2);
za = dvector(1,2);

for(k=1;k<=nn[3];k++) {
   for(j=1;j<=nn[2];j++) {
      for(i=1;i<=nn[1];i++) {
         D[i][j][k] = data[(k-1)*nn[1]*nn[2]+(j-1)*nn[1]+i];
      }
   }
}

Lx=(nn[1]-1)*step[1];
Ly=(nn[2]-1)*step[2];
Lz=(nn[3]-1)*step[3];
i=xin;
j=yin;
k=zin;
ido=(xin-i)/step[1];
jdo=(yin-j)/step[2];
kdo=(zin-k)/step[3];
itmp=ido;
jtmp=jdo;
ktmp=kdo;
ido=itmp*step[1];
jdo=jtmp*step[2];
kdo=ktmp*step[3];

if((i<0||i>(int) Lx)||(j<0||j>(int) Ly)||(k<0||k>(int) Lz)) {
   printf("\nOut of ranges in interpolate 3D Program.");
   printf("\ni = %3d, j = %3d, k = %3d",i,j,k);
   exit(1);
} else {
   if((i==(int) Lx)&&(j==(int) Ly)&&(k==(int) Lz)) {
      for(y=1;y<=2;y++) {
         xa[y] = i+ido+(y-2)*step[1];
         ya[y] = j+jdo+(y-2)*step[2];
         za[y] = k+kdo+(y-2)*step[3];
      }
   } else if((i==(int) Lx) && (j==(int) Ly) && (k<(int) Lz)) {
      for(y=1;y<=2;y++) {
         xa[y] = i+ido+(y-2)*step[1];
         ya[y] = j+jdo+(y-2)*step[2];
         za[y] = k+kdo+(y-1)*step[3];
      }
   } else if(i==(int) Lx && j<(int) Ly && k==(int) Lz) {
      for(y=1;y<=2;y++) {
         xa[y] = i+ido+(y-2)*step[1];
         ya[y] = j+jdo+(y-1)*step[2];
         za[y] = k+kdo+(y-2)*step[3];
      }
   } else if(i<(int) Lx && j==(int) Ly && k==(int) Lz) {
      for(y=1;y<=2;y++) {
            xa[y] = i+ido+(y-1)*step[1];
            ya[y] = j+jdo+(y-2)*step[2];
            za[y] = k+kdo+(y-2)*step[3];
         }
      } else if(i<(int) Lx && j<(int) Ly && k==(int) Lz) {
         for(y=1;y<=2;y++) {
            xa[y] = i+ido+(y-1)*step[1];
            ya[y] = j+jdo+(y-1)*step[2];
            za[y] = k+kdo+(y-2)*step[3];
      }
      } else if(i==(int) Lx && j<(int) Ly && k<(int) Lz) {
         for(y=1;y<=2;y++) {
            xa[y] = i+ido+(y-2)*step[1];
            ya[y] = j+jdo+(y-1)*step[2];
            za[y] = k+kdo+(y-1)*step[3];
         }
      } else if(i<(int) Lx && j==(int) Ly && k<(int) Lz) {
         for(y=1;y<=2;y++) {
            xa[y] = i+ido+(y-1)*step[1];
            ya[y] = j+jdo+(y-2)*step[2];
            za[y] = k+kdo+(y-1)*step[3];
         }
      } else {
         for(y=1;y<=2;y++) {
            xa[y]=i+ido+(y-1)*step[1];
            ya[y]=j+jdo+(y-2)*step[2];
            za[y]=k+kdo+(y-1)*step[3];
         }
      }
      for(y=1;y<=2;y++) {
         for(z=1;z<=2;z++) {
            for(x=1;x<=2;x++) {
               ido=xa[x]/step[1]+0.5;
               jdo=ya[y]/step[2]+0.5;
               kdo=za[z]/step[3]+0.5;
               i=ido;
               j=jdo;
               k=kdo;
               Dtmp[4*(y-1)+2*(z-1)+x] = D[i+1][j+1][k+1];
            }
         }
      }
      polint3(xin,yin,zin,xa,ya,za,Dtmp,&D_PT);
   }
   free_dvector(za,1);
   free_dvector(ya,1);
   free_dvector(xa,1);
   free_dvector(Dtmp,1);
   free_darray2(D,1,nn[1],1,nn[2],1);

   return D_PT;
}

void polint1(x,xa,fpt,f)
double x,xa[],fpt[],*f;
/*
   x   = real x.
   f   = pointer of data output.
   xa  = array of 2 points between x in x-direction.
```

```
     fpt = input data of 2 positions around interested
           coordinates(x,y).
  */
  {
     double frac;

     frac=(x-xa[1])/(xa[2]-xa[1]);
     *f=(1-frac)*fpt[1]+frac*fpt[2];
  }

  void polint2(x,y,xa,ya,fpt,f)
  double x,y,xa[],ya[],fpt[],*f;
  /*
     x   = real x.
     y   = real y.
     f   = pointer of data output.
     xa  = array of 2 points between x in x-direction.
     ya  = array of 2 points between y in y-direction.
     fpt = input data of 4 positions around interested
           coordinate (x,y).
       */
  {
     int i,j;
     void polint1();
     double *fmtmp,*ftmp,*dvector();

     fmtmp=dvector(1,2);
     ftmp=dvector(1,2);
     /* Find fa and fb. */
     for(j=1;j<=2;j++) {
        for(i=1;i<=2;i++) ftmp[i]=fpt[i+(j-1)*2];
        polint1(x,xa,ftmp,&fmtmp[j]);
     }
     /* Find fs */
     polint1(y,ya,fmtmp,f);
     free_dvector(ftmp,1);
     free_dvector(fmtmp,1);
  }

  void polint3(x,y,z,xa,ya,za,fpt,f)
  double x,y,z,xa[],ya[],za[],fpt[],*f;
  /*
     x   = real x.
     y   = real y.
     z   = real z.
     f   = pointer of data output.
     xa  = array of 2 points between x in x-direction.
     ya  = array of 2 points between y in y-direction.
     za  = array of 2 points between z in z-direction.
     fpt = input data of 8 positions around interested
           coordinate (x,y,z).
  */
  {
     int i,j,k,no;
     void polint1();
     double *fmtmp,*frtmp,*ftmp,*dvector();

     fmtmp=dvector(1,2);
     frtmp=dvector(1,2);
     ftmp=dvector(1,2);
```

```
   /* Find (fa, fb and fp) and (fc, fd and fq). */
   for(k=1;k<=2;k++) {
      for(j=1;j<=2;j++) {
         for(i=1;i<=2;i++) ftmp[i]=fpt[i+(j-1)*2+(k-1)*4];
         polint1(x,xa,ftmp,&fmtmp[j]);
      }
      polint1(z,za,fmtmp,&frtmp[k]);
   }
   /* Find fs */
   polint1(y,ya,frtmp,f);
   free_dvector(ftmp,1);
   free_dvector(frtmp,1);
   free_dvector(fmtmp,1);
}
```

Appendix D

Program for Simulating Magnetic Fields in Three Dimensions[1]

maimmag.c

The main program for simulating the magnetic field in 3D. It gets values from the user and executes other files. The file mainmag.c contains functions:

main() gets the input value, selects the magnetogram data and the plane we want to plot from the user.

subplane() sets the calculation for the magnetic field in the selected plane.

openfile() reads input data from a data file in FITS format.

sortdata() provides the output data in a form to run in the next process.

fourier2.c

It includes the forward and inverse Fourier transforms in two dimensions by using the fast Fourier transform (FFT) (Press et al., 1988). The subroutines in this program are

Forward_FFT() and **Inverse_FFT()** provide the data for a forward or inverse Fourier transform.

four2() is the FFT routine adapted from Numerical Recipes in C (Press et al., 1988).

[1]written by the author, except where noted

truedata() is the routine to convert the data from **four2()** to real values, e.g., multiplying $1/(N_1 N_2)$ for an inverse FFT.

magnetic.c

It is used for calculating the magnetic field in three components. Included subroutines are:

calculatebx() calculates the magnetic field in the x-direction.

calculateby() calculates the magnetic field in the y-direction.

calculatebz() calculates the magnetic field in the z-direction.

rotate.c

It is a program use to rotate from system 1 to system 2. Included subroutines are:

rotate_interpolate() calculates the rotation and interpolation.

polin.c

See Appendix C.

printout.c

Prints out the magnetic field data in 3-components on the selected plane.

nrutil.c

It contains routines to reserve and unreserve the memory for arrays (see Appendix B).

The program:
mainmag.c

```
/*
   30th September 1998 -- mainmag.c
          First written.

   15th June 1999 -- mainmag.c
          Add the rotate function before FFT.

   16th June 1999 -- mainmag.c
          Change the range of region which I interesting; Xmin,
          Xmax, Ymin and Ymax of each days after rotate.
   17th June 1999 -- ofile.c
           Change the line of sight angle by using sin(phi) = x/R
           of each days.

   15th July 1999 -- ofile.c
           Add print Sun in openfile().
*/

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

#define Pi  4.0*atan(1.0)

int     ndim = 2;
int     nn[3];
double  data[399619];
double  delta[3];
double  TINY = 0.000001;

void main(void)
{
   int     i, x, y, ntot;
   int     openfile(), choice, plane;
   double  z, alpha, phi, rotatetheta;

   void    subplane();

   printf("\n\tMagnetogram Data AR7912 from Kitt Peak's files.");
   printf("\n\t\t1. ==> 951015m.fit");
   printf("\n\t\t2. ==> 951018m.fit");
   printf("\n\t\t3. ==> 951020m.fit");
   printf("\n\t\t4. ==> 951021m.fit");
   printf("\n\t\t5. ==> Input by yourselve");
   printf("\n\t\tSelect magnetic file data: ");
   scanf("%d",&choice);
   switch(choice) {
      case 1: rotatetheta = -63.0; /*16.34853538*/
              phi = 16.82644886;
              break;
      case 2: rotatetheta = 26.0; /*-38.90698985*/
              phi = -37.24778086;
              break;
      case 3: rotatetheta = 13.0; /*-55.49245489*/
              phi = -63.47464799;
              break;
      case 4: rotatetheta = 9.0;
              phi = -71.07535559;
              break;
      case 5: printf("\n\t\tRotate theta angle (degree) = ");
              scanf("%lf",&rotatetheta);
              printf("\n\t\tEnter phi angle (degree) = ");
              scanf("%lf",&phi);
              break;
   }
   printf("\n\n\t\tDo you want printout: ");
   printf("\n\t\t1. x-y plane.");
   printf("\n\t\t2. x-z plane.");
   printf("\n\t\t3. y-z plane.");
   printf("\n\t\t4. x-y-z dimensions.");
   printf("\n\t\tEnter select plane ==> ");
   scanf("%d",&plane);
   printf("\n\t\tEnter alpha constant (arc sec.^-1) ==> ");
   scanf("%lf",&alpha);
   phi*=Pi/180.0;
   rotatetheta*=Pi/180.0;
   switch(plane) {
      case 1: /* x-y plane will be fit z-axis. */
              printf("\n\t\tPrint out x-y plane will select
                     constant z.");
              printf("\n\t\tEnter fit constant hight z = ");
              scanf("%lf",&z);
              x=y=1;
              subplane(choice,alpha,phi,rotatetheta,plane,x,y,z);
              break;
      case 2: /* x-z plane will be fit y-axis. */
              printf("\n\t\tPrint out x-z plane will select
                     constant y.");
              printf("\n\t\tEnter fit constant y = ");
              scanf("%d",&y);
              z=0.0;
              x=1;
              subplane(choice,alpha,phi,rotatetheta,plane,x,y,z);
              break;
      case 3: /* y-z plane will be fit x-axis. */
              printf("\n\t\tPrint out y-z plane will select
                     constant x.");
              printf("\n\t\tEnter fit constant x ==> ");
              scanf("%d",&x);
              z=0.0;
              y=1;
              subplane(choice,alpha,phi,rotatetheta,plane,x,y,z);
              break;
      case 4: /* x-y-z dimensions. */
              printf("\n\t\tPrint out 3D.");
              x=1;
              y=1;
              z=0.0;
              subplane(choice,alpha,phi,rotatetheta,plane,x,y,z);
              break;
   }
   printf ("\n\t  !!!!!Thank you every much for your visit!!!!!\n\n");
}

void subplane(choice,alpha,phi,rotatetheta,plane,x,y,z)
int    x, y, plane, choice;
double alpha, phi, z, rotatetheta;
{
   int    i, j, nz, ntot, k, openfile();
   char   savebx[25], saveby[25], savebz[25];
   double step;

   void   Forward_FFT(), Inverse_FFT(), sortdata();
   void   printoutxy(), printoutxz(), printoutyz();
   void   printoutnn(), printoutxyz();
   void   calculatebx(), calculateby(), calculatebz();

   printf("\n\t\tSave file name for Bx: ");
   scanf("%s",savebx);
   printf("\n\t\tSave file name for By: ");
   scanf("%s",saveby);
   printf("\n\t\tSave file name for Bz: ");
```

```
scanf("%s",savebz);

switch(plane) {
   case 1: /* x-y plane */
           /* Calculate Bx(x,y,z) */
           ntot=1;
           i=0;
           ntot=openfile(ntot,choice,phi,rotatetheta);
           Forward_FFT(ndim,nn,delta,data);
           calculatebx(i,phi,alpha,z,data,delta,nn,ndim);
           Inverse_FFT(ndim,nn,delta,data);
           sortdata(ntot);
           printoutxy(nn,data,savebx);

           /* Calculate By(x,y,z) */
           ntot=1;
           i=0;
           ntot=openfile(ntot,choice,phi,rotatetheta);
           Forward_FFT(ndim,nn,delta,data);
           calculateby(i,phi,alpha,z,data,delta,nn,ndim);
           Inverse_FFT(ndim,nn,delta,data);
           sortdata(ntot);
           printoutxy(nn,data,saveby);

           /* Calculate Bz(x,y,z) */
           ntot=1;
           i=0;
           ntot=openfile(ntot,choice,phi,rotatetheta);
           Forward_FFT(ndim,nn,delta,data);
           calculatebz(i,phi,alpha,z,data,delta,nn,ndim);
           Inverse_FFT(ndim,nn,delta,data);
           sortdata(ntot);
           printoutxy(nn,data,savebz);
           break;

   case 2: /* x-z plane */
           printf("\n\t\tEnter the step hight z (arc sec.) ==> ");
           scanf("%lf",&step);
           printf("\n\t\tEnter number of step z (times) ==> ");
           scanf("%d",&nz);

           for(i=1;i<=nz;i++) {
              /* Calculate Bx(x,y,z) */
              ntot=1;
              ntot=openfile(ntot,choice,phi,rotatetheta);
              Forward_FFT(ndim,nn,delta,data);
              calculatebx(i,phi,alpha,z,data,delta,nn,ndim);
              Inverse_FFT(ndim,nn,delta,data);
              sortdata(ntot);
              printoutxz(nn,y,data,savebx);

              /* Calculate By(x,y,z) */
              ntot=1;
              ntot=openfile(ntot,choice,phi,rotatetheta);
              Forward_FFT(ndim,nn,delta,data);
              calculateby(i,phi,alpha,z,data,delta,nn,ndim);
              Inverse_FFT(ndim,nn,delta,data);
              sortdata(ntot);
              printoutxz(nn,y,data,saveby);

              /* Calculate Bz(x,y,z) */
```

```
                ntot=1;
                ntot=openfile(ntot,choice,phi,rotatetheta);
                Forward_FFT(ndim,nn,delta,data);
                calculatebz(i,phi,alpha,z,data,delta,nn,ndim);
                Inverse_FFT(ndim,nn,delta,data);
                sortdata(ntot);
                printoutxz(nn,y,data,savebz);
                /*printf("\n");*/

                z+=step;
             }
             break;

    case 3:  /* y-z plane */
             printf("\n\t\tEnter the step hight z (arc sec.) ==> ");
             scanf("%lf",&step);
             printf("\n\t\tEnter number of step z (times) ==> ");
             scanf("%d",&nz);

             for(i=1;i<=nz;i++) {
                /* Calculate Bx(x,y,z) */
                ntot=1;
                ntot=openfile(ntot,choice,phi,rotatetheta);
                Forward_FFT(ndim,nn,delta,data);
                calculatebx(i,phi,alpha,z,data,delta,nn,ndim);
                Inverse_FFT(ndim,nn,delta,data);
                sortdata(ntot);
                printoutyz(nn,x,data,savebx);

                /* Calculate By(x,y,z) */
                ntot=1;
                ntot=openfile(ntot,choice,phi,rotatetheta);
                Forward_FFT(ndim,nn,delta,data);
                calculateby(i,phi,alpha,z,data,delta,nn,ndim);
                Inverse_FFT(ndim,nn,delta,data);
                sortdata(ntot);
                printoutyz(nn,x,data,saveby);

                /* Calculate Bz(x,y,z) */
                ntot=1;
                ntot=openfile(ntot,choice,phi,rotatetheta);
                Forward_FFT(ndim,nn,delta,data);
                calculatebz(i,phi,alpha,z,data,delta,nn,ndim);
                Inverse_FFT(ndim,nn,delta,data);
                sortdata(ntot);
                printoutyz(nn,x,data,savebz);
                /*printf("\n");*/

                z+=step;
             }
             break;

    case 4:  /* Print out 3D data. */
             printf("\n\t\tEnter the step hight z (arc sec.) ==> ");
             scanf("%lf",&step);
             printf("\n\t\tEnter number of step z (times) ==> ");
             scanf("%d",&nz);

             for(i=1;i<=nz;i++) {
                /* Calculate Bx(x,y,z) */
```

```
                    ntot=1;
                    ntot=openfile(ntot,choice,phi,rotatetheta);
                    Forward_FFT(ndim,nn,delta,data);
                    calculatebx(i,phi,alpha,z,data,delta,nn,ndim);
                    Inverse_FFT(ndim,nn,delta,data);
                    sortdata(ntot);
                    printoutxyz(nn,data,savebx);

                    /* Calculate By(x,y,z) */
                    ntot=1;
                    ntot=openfile(ntot,choice,phi,rotatetheta);
                    Forward_FFT(ndim,nn,delta,data);
                    calculateby(i,phi,alpha,z,data,delta,nn,ndim);
                    Inverse_FFT(ndim,nn,delta,data);
                    sortdata(ntot);
                    printoutxyz(nn,data,saveby);

                    /* Calculate Bz(x,y,z) */
                    ntot=1;
                    ntot=openfile(ntot,choice,phi,rotatetheta);
                    Forward_FFT(ndim,nn,delta,data);
                    calculatebz(i,phi,alpha,z,data,delta,nn,ndim);
                    Inverse_FFT(ndim,nn,delta,data);
                    sortdata(ntot);
                    printoutxyz(nn,data,savebz);
                    /*printf("\n");*/

                    z+=step;
                 }
                 break;
   }
   printf("\n\n\t\tSave file '%s' already!!\n",savebx);
   printf("\t\tSave file '%s' already!!\n",saveby);
   printf("\t\tSave file '%s' already!!\n\n",savebz);
}

int openfile(ntot,choice,phi,rotatetheta)
int    ntot, choice;
double phi, rotatetheta;
{
   int    i, j, x, y, idim;
   static int check=1;
   int    Xmin, Xmax, Ymin, Ymax;
   int    ORIGIN, TAPE[448][448];
   char   input_name[15], rotate_name[15];
   FILE   *fp, *fs, *fopen();
   double step[3], datatmp[448][448];
   double BZERO, BSCALE, BACKGROUND;

   void   rotate_interpolate();

   switch(choice) {
      case 1: /* Magnetograph data on 15-10-1995 */
              if((fp=fopen("951015m.fit","r"))==NULL) {
                 printf("\n\t\tError in open file '951015m.fit' for
                         read\n");
                 exit(1);
              }
              if((fs=fopen("951015r.dat","w"))==NULL) {
                 printf("\n\t\tError in open file '951015r.dat' for
                         read\n");
```

```
                   exit(1);
                }
                BSCALE      = 9.7992226909e0;
                BZERO       = -1.4537992126e3;
                BACKGROUND = 148;
                Xmin = 131;
                Xmax = 194;
                Ymin = 193;
                Ymax = 256;
                break;
        case 2: /* Magnetograph data on 18-10-1995 */
                if((fp=fopen("951018m.fit","r"))==NULL) {
                   printf("\n\t\tError in open file '951018m.fit' for
                           read\n");
                   exit(1);
                }
                if((fs=fopen("951018r.dat","w"))==NULL) {
                   printf("\n\t\tError in open file '951018r.dat' for
                           read\n");
                   exit(1);
                }
                BSCALE      = 7.2362275612;
                BZERO       = -1.3742362205e3;
                BACKGROUND =  189;
                Xmin = 317;
                Xmax = 380;
                Ymin = 191;
                Ymax = 254;
                break;
        case 3: /* Magnetograph data on 20-10-1995 */
                if((fp=fopen("951020m.fit","r"))==NULL) {
                   printf("\n\t\tError in open file '951020m.fit' for
                           read\n");
                   exit(1);
                }
                if((fs=fopen("951012r.dat","w"))==NULL) {
                   printf("\n\t\tError in open file '951020r.dat' for
                            read\n");
                   exit(1);
                }
                BSCALE      = 6.9212665708;
                BZERO       = -1.1339212598e3;
                BACKGROUND =  163;
                Xmin = 394;
                Xmax = 425;
                Ymin = 192;
                Ymax = 223;
                break;
        case 4: /* Magnetograph data on 21-10-1995 */
                if((fp=fopen("951021m.fit","r"))==NULL) {
                   printf("\nError in open file '951021m.fit' for
                           read\n");
                   exit(1);
                }
                if((fs=fopen("951021r.dat","w"))==NULL) {
                   printf("\n\t\tError in open file '951021r.dat' for
                           save\n");
                   exit(1);
                }
                BSCALE      = 5.1338633139;
                BZERO       = -7.5513385827e2;
```

```
              BACKGROUND = 147;
              Xmin = 401;
              Xmax = 432;
              Ymin = 196;
              Ymax = 227;
              break;
      case 5: /* Magnetograph data input by yourself.
                 Open <filename>.fit */
              printf("\n\t\tOpen input data <.fit> : ");
              scanf("%s",input_name);
              fp=fopen(input_name,"r");
              if(fp==NULL) {
                 printf("\n\t\tError in open file '%s' for read\n",
                        input_name);
                 exit(1);
              }
              printf("\n\t\tOpen save rotated data : ");
              scanf("%s",rotate_name);
              fs=fopen(rotate_name,"w");
              if(fs==NULL) {
                 printf("\n\t\tError in open file '%s' for read\n",
                       rotate_name);
                 exit(1);
              }
              printf("\t\tBSCALE     = "); scanf("%lf",&BSCALE);
              printf("\t\tBZERO      = "); scanf("%lf",&BZERO);
              printf("\t\tBACKGROUND = "); scanf("%lf",&BACKGROUND);
              printf("\t\tXmin = "); scanf("%d",&Xmin);
              printf("\t\tXmax = "); scanf("%d",&Xmax);
              printf("\t\tYmin = "); scanf("%d",&Ymin);
              printf("\t\tYmax = "); scanf("%d",&Ymax);
              break;
   }
   /* First character position in <######m.fit> program. */
   nn[1]=447;
   nn[2]=447;
   step[1]=step[2]=1.0;
   ORIGIN=8640;
   for(i=0;i<ORIGIN;i++) getc(fp);
   for(y=1;y<=nn[2];y++) {
      for(x=1;x<=nn[1];x++)  TAPE[x][y]=getc(fp);
   }
   ntot=1;
   for(idim=1;idim<=ndim;idim++) ntot*=nn[idim];
   for(j=1;j<=nn[2];j++) {
      for(i=1;i<=nn[1];i++) {
         if(TAPE[i][j]==TAPE[1][1]) data[(j-1)*nn[1]+i]=0.0;
         else if((double)TAPE[i][j]==BACKGROUND)
                data[(j-1)*nn[1]+i]=0.0;
         else data[(j-1)*nn[1]+i]=TAPE[i][j]*BSCALE+BZERO
                                  -(BACKGROUND*BSCALE+BZERO);
      }
   }
   for(j=1;j<=nn[2];j++) {
      for(i=1;i<=nn[1];i++) {
         datatmp[i][j]=data[(j-1)*nn[1]+i];
      }
   }
   rotate_interpolate(rotatetheta,nn,step,data);
   for(j=1;j<=nn[2];j++) {
      for(i=1;i<=nn[1];i++) {
         datatmp[i][j]=data[(j-1)*nn[1]+i];
      }
   }
/*
   if(check==1) {
      for(j=1;j<=447;j+=2) {
         for(i=1;i<=447;i+=2) {
            printf("%13.6lf",data[(j-1)*447+i];
         }
         printf{"\n");
      }
      check++;
   }
*/
   for(j=Ymin;j<=Ymax;j++) {
      for(i=Xmin;i<=Xmax;i++) {
         fprintf(fs,"%13.6lf ",datatmp[i][j]);
      }
      fprintf(fs,"\n");
   }
   delta[1]=1.0/cos(phi);
   delta[2]=1.0;
   nn[1]=2*(Xmax-Xmin+1); /* numbers of x-dimensions. */
   nn[2]=2*(Ymax-Ymin+1); /* numbers of y-dimensions. */
   printf("\nNo. of x = %d",nn[1]);
   printf("\nNo. of y = %d",nn[2]);
   for(idim=1;idim<=ndim;idim++) ntot*=nn[idim];
   i=1;
   for(y=1;y<=(nn[2]/4);y++) {
      for(x=1;x<=nn[1]/4;x++,i+=2) {
         data[i]=datatmp[Xmax-(nn[1]/4)+x][Ymax-(nn[2]/4)+y];
         data[i+1]=0.0;
      }
      for(x=1;x<=nn[1]/4;x++,i+=2) {
         data[i]=0.0;
         data[i+1]=0.0;
      }
      for(x=1;x<=nn[1]/4;x++,i+=2) {
         data[i]=0.0;
         data[i+1]=0.0;
      }
      for(x=-(nn[1]/4)+1;x<=0;x++,i+=2) {
         data[i]=datatmp[Xmin-1+(nn[1]/4)+x][Ymax-(nn[2]/4)+y];
         data[i+1]=0.0;
      }
   }
   for(y=1;y<=nn[2]/4;y++) {
      for(x=1;x<=nn[1];x++,i+=2) {
         data[i]=0.0;
         data[i+1]=0.0;
      }
   }
   for(y=1;y<=nn[2]/4;y++) {
      for(x=1;x<=nn[1];x++,i+=2) {
         data[i]=0.0;
         data[i+1]=0.0;
      }
   }
   for(y=-(nn[2]/4)+1;y<=0;y++) {
      for(x=1;x<=nn[1]/4;x++,i+=2) {
         data[i]=datatmp[Xmax-(nn[1]/4)+x][Ymin-1+(nn[2]/4)+y];
         data[i+1]=0.0;
      }
      for(x=1;x<=nn[1]/4;x++,i+=2) {
         data[i]=0.0;
         data[i+1]=0.0;
      }
      for(x=1;x<=nn[1]/4;x++,i+=2) {
         data[i]=0.0;
         data[i+1]=0.0;
      }
      for(x=-(nn[1]/4)+1;x<=0;x++,i+=2) {
         data[i]=datatmp[Xmin-1+(nn[1]/4)+x][Ymin-1+(nn[2]/4)+y];
         data[i+1]=0.0;
      }
   }
   fclose(fp);
   fclose(fs);
   fclose(fs1);
   return ((i-1)/2);
}

void sortdata(ntot)
int ntot;
{
   int i,x,y;
   double *Bl;
   double *dvector();

   Bl=dvector(1,2*ntot);
   for(i=1;i<=2*ntot;i+=2) {
      Bl[i]=data[i];
      Bl[i+1]=data[i+1];
   }

   i=1;
   for(y=nn[2]/2;y<=nn[2]-1;y++) {
      for(x=nn[1]+1;x<=2*nn[1];x+=2,i+=2) {
         data[i]=Bl[x+2*y*nn[1]];
         data[i+1]=Bl[x+2*y*nn[1]+1];
      }
      for(x=1;x<=nn[1];x+=2,i+=2) {
         data[i]=Bl[x+2*y*nn[1]];
         data[i+1]=Bl[x+2*y*nn[1]+1];
      }
   }
   for(y=0;y<=nn[2]/2-1;y++) {
      for(x=nn[1]+1;x<=2*nn[1];x+=2,i+=2) {
         data[i]=Bl[x+2*y*nn[1]];
         data[i+1]=Bl[x+2*y*nn[1]+1];
      }
      for(x=1;x<=nn[1];x+=2,i+=2) {
         data[i]=Bl[x+2*y*nn[1]];
         data[i+1]=Bl[x+2*y*nn[1]+1];
      }
   }
   free_dvector(Bl,1);
}
```

fourier2.c

```
/*
   fourier2.c  ---> Tuesday 27th August 1998.
   This program is used for Forward Fourier Transform and
Inverse Fourier Transform in 2-dimensions.
*/

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

#define SWAP(a,b) tempr=(a);(a)=(b);(b)=tempr
#define Pi 3.141592653589323

void Forward_FFT(ndim,nn,delta,data)
int ndim,nn[];
double delta[],data[];
{
   int i,isign,ntot;
   double DELTA;
   void four2(),truedata();

   isign=-1;
   ntot=1;
   DELTA=1.0;
   for(i=1;i<=ndim;i++) {
      ntot=ntot*nn[1];
      DELTA=DELTA*delta[i];
   }
   for(i=1;i<=2*ntot;i+=2) {
      data[i]=data[i]/DELTA;
      data[i+1]=data[i+1]/DELTA;
   }
   four2(isign,ndim,nn,data);
   truedata(isign,nn,ndim,delta,data);
}

void Inverse_FFT(ndim,nn,delta,data)
int nn[],ndim;
double delta[],data[];
{
   int i,isign;
   void four2(),truedata();

   isign=1;
   four2(isign,ndim,nn,data);
   truedata(isign,nn,ndim,delta,data);
}

/* This program come from Numerical Recipes in C,
Chapter 12, p469-p470. */
void four2(isign,ndim,nn,data)
int isign,ndim,nn[];
double data[];
{
   int i1,i2,i3,i2rev,i3rev,ip1,ip2,ip3,ifp1,ifp2;
   int ibit,idim,k1,k2,n,nprev,nrem,i,ntot;
   double tempi,tempr;
   double theta,wi,wpi,wpr,wr,wtemp;

   ntot=1;
   for(idim=1;idim<=ndim;idim++)
      ntot*=nn[idim];
nprev=1;
/* Main loop over the dimensions. */
for(idim=ndim;idim>=1;idim--) {
   n=nn[idim];    /* n is Nn(loop1)...N1(loopn) */
   nrem=ntot/(n*nprev);
   ip1=nprev<<1;   /* ip1=2*nprev */
   ip2=ip1*n;
   ip3=ip2*nrem;
   i2rev=1;
   for(i2=1;i2<=ip2;i2+=ip1) {
      if(i2<i2rev) {
         for(i1=i2;i1<=i2+ip1-2;i1+=2) {
            for(i3=i1;i3<=ip3;i3+=ip2) {
               i3rev=i2rev+i3-i2;
               SWAP(data[i3],data[i3rev]);
               SWAP(data[i3+1],data[i3rev+1]);
            }
         }
      }
      ibit=ip2>>1;
      while(ibit>=ip1 && i2rev>ibit) {
         i2rev-=ibit;
         ibit>>=1;
      }
      i2rev+=ibit;
   }
   ifp1=ip1;
   while(ifp1<ip2) {
      ifp2=ifp1<<1;
      theta=isign*2*Pi/(ifp2/ip1);
      /* Initialize for the trig. recurrence. */
      wtemp=sin(0.5*theta);
      wpr=-2.0*wtemp*wtemp;
      wpi=sin(theta);
      wr=1.0;
      wi=0.0;
      for(i3=1;i3<=ifp1;i3+=ip1) {
         for(i1=i3;i1<=i3+ip1-2;i1+=2) {
            for(i2=i1;i2<=ip3;i2+=ifp2) {
               /* Danielson-Lanczos formula: */
               k1=i2;
               k2=k1+ifp1;
               tempr=wr*data[k2]-wi*data[k2+1];
               tempi=wr*data[k2+1]+wi*data[k2];
               data[k2]=data[k1]-tempr;
               data[k2+1]=data[k1+1]-tempi;
               data[k1]+=tempr;
               data[k1+1]+=tempi;
            }
         }
         /* Trigonometric recurrence */
         wr=(wtemp=wr)*wpr-wi*wpi+wr;
         wi=wi*wpr+wtemp*wpi+wi;
      }
      ifp1=ifp2;
   }
   nprev*=n;
 }
}
```

```
void truedata(isign,nn,ndim,delta,data)
int isign,nn[],ndim;
double delta[],data[];
{
   int i,ntot;
   double DELTA;

   ntot=1;
   DELTA=1.0;
   for(i=1;i<=ndim;i++) {
      ntot=ntot*nn[i];
      DELTA=DELTA*delta[1];
   }
   switch(isign) {
      /* Forward Fourier Transform. */
      case -1: for(i=1;i<=2*ntot;i+=2) {
                  data[i]=data[i]*DELTA;
                  data[i+1]=data[i+1]*DELTA;
               }
               break;
      /* Inverse Fourier Transform. */
      case  1: for(i=1;i<=2*ntot;i+=2) {
                  data[i]=data[i]/ntot;
                  data[i+1]=data[i+1]/ntot;
               }
               break;
   }
}
```

magnetic.c

```
/*
   magnetic.c --> Sunday 13th September 1998
*/

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

#define Pi 3.141592653589323
#define SMALL 1e-10

extern int ndim;

void calculatebx(i,phi,alpha,z,data,delta,nn)
int i,nn[];
double z,phi,alpha;
double data[],delta[];
{
   int n,idim,ntot;
   double u,v,k,q;
   double b1,b2,r1,r2;
   double *B1,*dvector();
   void free_dvector();

   ntot=1;
   for(idim=1;idim<=ndim;idim++) ntot=ntot*nn[idim];
   B1=dvector(1,2*ntot);
   for(n=1;n<=2*ntot;n+=2) {
      B1[n]=data[n];
      B1[n+1]=data[n+1];
   }
   n=1;
```

```
for(v=0.0;v<1.0/(2.0*delta[2])&&fabs(v-1./(2.*delta[2]))>SMALL;
    v+=1.0/(delta[2]*nn[2])) {
   for(u=0.0;u<1.0/(2.0*delta[1])&&fabs(u-1./(2.*delta[1]))>SMALL;
       u+=1.0/(delta[1]*nn[1])) {
      q=sqrt(u*u+v*v);
      if(q>=fabs(alpha/(2.0*Pi))) {
         k=sqrt((4.0*Pi*Pi*q*q)-(alpha*alpha));
         r1=sin(phi)*sin(phi)*(u*k-v*alpha)*(u*k-v*alpha);
         r2=4.0*Pi*Pi*q*q*q*q*cos(phi)*cos(phi);
         b1=sin(phi)*(u*k-v*alpha)*(u*k-v*alpha)*exp(-k*z);
         b2=2.0*Pi*q*q*cos(phi)*(u*k-v*alpha)*exp(-k*z);
         data[n]=(b1*Bl[n]+b2*Bl[n+1])/(r1+r2);
         data[n+1]=(b1*Bl[n+1]-b2*Bl[n])/(r1+r2);
      } else {
         data[n]=0.0;
         data[n+1]=0.0;
      }
      n+=2;
   }
   data[n]=0.0;
   data[n+1]=0.0;
   n+=2;
   for(u=-(nn[1]/2.0-1.0)/(nn[1]*delta[1]);u<0.0&&fabs(u)>SMALL;
       u+=1.0/(delta[1]*nn[1])) {
      q=sqrt(u*u+v*v);
      if(q>=fabs(alpha/(2.0*Pi))) {
         k=sqrt((4.0*Pi*Pi*q*q)-(alpha*alpha));
         r1=sin(phi)*sin(phi)*(u*k-v*alpha)*(u*k-v*alpha);
         r2=4.0*Pi*Pi*q*q*q*q*cos(phi)*cos(phi);
         b1=sin(phi)*(u*k-v*alpha)*(u*k-v*alpha)*exp(-k*z);
         b2=2.0*Pi*q*q*cos(phi)*(u*k-v*alpha)*exp(-k*z);
         data[n]=(b1*Bl[n]+b2*Bl[n+1])/(r1+r2);
         data[n+1]=(b1*Bl[n+1]-b2*Bl[n])/(r1+r2);
      } else {
         data[n]=0.0;
         data[n+1]=0.0;
      }
      n+=2;
   }
}
for(;fabs(v-1.0/(2.0*delta[2]))<SMALL;v+=1.0/(delta[2]*nn[2])) {
   for(u=0.0;u<=1.0/(2.0*delta[1]);u+=1.0/(delta[1]*nn[1])) {
      data[n]=0.0;
      data[n+1]=0.0;
      n+=2;
   }
   for(u=-(nn[1]/2.0-1.0)/(nn[1]*delta[1]);u<=-1.0/(nn[1]*delta[1]);
       u+=1.0/(delta[1]*nn[1])) {
      data[n]=0.0;
      data[n+1]=0.0;
      n+=2;
   }
}
for(v=-(nn[2]/2.0-1.0)/(nn[2]*delta[2]);v<0.0&&fabs(v)>SMALL;
     v+=1.0/(delta[2]*nn[2])) {
   for(u=0.0;u<1.0/(2.0*delta[1])&&fabs(u-1./(2.*delta[1]))>SMALL;
       u+=1.0/(delta[1]*nn[1])) {
      q=sqrt(u*u+v*v);
      if(q>=fabs(alpha/(2.0*Pi))) {
         k=sqrt((4.0*Pi*Pi*q*q)-(alpha*alpha));
         r1=sin(phi)*sin(phi)*(u*k-v*alpha)*(u*k-v*alpha);
```

```
            r2=4.0*Pi*Pi*q*q*q*q*cos(phi)*cos(phi);
            b1=sin(phi)*(u*k-v*alpha)*(u*k-v*alpha)*exp(-k*z);
            b2=2.0*Pi*q*q*cos(phi)*(u*k-v*alpha)*exp(-k*z);
            data[n]=(b1*Bl[n]+b2*Bl[n+1])/(r1+r2);
            data[n+1]=(b1*Bl[n+1]-b2*Bl[n])/(r1+r2);
         } else {
            data[n]=0.0;
            data[n+1]=0.0;
         }
         n+=2;
      }
      data[n]=0.0;
      data[n+1]=0.0;
      n+=2;
      for(u=-(nn[1]/2.0-1.0)/(nn[1]*delta[1]);u<0.0&&fabs(u)>SMALL;
          u+=1.0/(delta[1]*nn[1])) {
         q=sqrt(u*u+v*v);
         if(q>=fabs(alpha/(2.0*Pi))) {
            k=sqrt((4.0*Pi*Pi*q*q)-(alpha*alpha));
            r1=sin(phi)*sin(phi)*(u*k-v*alpha)*(u*k-v*alpha);
            r2=4.0*Pi*Pi*q*q*q*q*cos(phi)*cos(phi);
            b1=sin(phi)*(u*k-v*alpha)*(u*k-v*alpha)*exp(-k*z);
            b2=2.0*Pi*q*q*cos(phi)*(u*k-v*alpha)*exp(-k*z);
            data[n]=(b1*Bl[n]+b2*Bl[n+1])/(r1+r2);
            data[n+1]=(b1*Bl[n+1]-b2*Bl[n])/(r1+r2);
         } else {
            data[n]=0.0;
            data[n+1]=0.0;
         }
         n+=2;
      }
   }
   free_dvector(Bl,1);
   printf("\n%d. Calculated Bx(x,y,z=%lf) already!!!",i,z);
}

void calculateby(i,phi,alpha,z,data,delta,nn)
int i,nn[];
double z,phi,alpha;
double data[],delta[];
{
   int n,idim,ntot;
   double u,v,k,q;
   double b1,b2,r1,r2;
   double *Bl,*dvector();
   void free_dvector();

   ntot=1;
   for(idim=1;idim<=ndim;idim++) ntot=ntot*nn[idim];
   Bl=dvector(1,2*ntot);
   for(n=1;n<=2*ntot;n+=2) {
      Bl[n]=data[n];
      Bl[n+1]=data[n+1];
   }
   n=1;
   for(v=0.0;v<1.0/(2.0*delta[2])&&fabs(v-1./(2.*delta[2]))>SMALL;
       v+=1.0/(delta[2]*nn[2])) {
      for(u=0.0;u<1.0/(2.0*delta[1])&&fabs(u-1./(2.*delta[1]))>SMALL;
          u+=1.0/(delta[1]*nn[1])) {
         q=sqrt(u*u+v*v);
         if(q>=fabs(alpha/(2.0*Pi))) {
```

```
         k=sqrt((4.0*Pi*Pi*q*q)-(alpha*alpha));
         r1=sin(phi)*sin(phi)*(u*k-v*alpha)*(u*k-v*alpha);
         r2=4.0*Pi*Pi*q*q*q*q*cos(phi)*cos(phi);
         b1=sin(phi)*(u*k-v*alpha)*(v*k+u*alpha)*exp(-k*z);
         b2=2.0*Pi*q*q*cos(phi)*(v*k+u*alpha)*exp(-k*z);
         data[n]=(b1*Bl[n]+b2*Bl[n+1])/(r1+r2);
         data[n+1]=(b1*Bl[n+1]-b2*Bl[n])/(r1+r2);
      } else {
         data[n]=0.0;
         data[n+1]=0.0;
      }
      n+=2;
   }
   data[n]=0.0;
   data[n+1]=0.0;
   n+=2;
   for(u=-(nn[1]/2.0-1.0)/(nn[1]*delta[1]);u<0.0&&fabs(u)>SMALL;
       u+=1.0/(delta[1]*nn[1])) {
      q=sqrt(u*u+v*v);
      if(q>=fabs(alpha/(2.0*Pi))) {
         k=sqrt((4.0*Pi*Pi*q*q)-(alpha*alpha));
         r1=sin(phi)*sin(phi)*(u*k-v*alpha)*(u*k-v*alpha);
         r2=4.0*Pi*Pi*q*q*q*q*cos(phi)*cos(phi);
         b1=sin(phi)*(u*k-v*alpha)*(v*k+u*alpha)*exp(-k*z);
         b2=2.0*Pi*q*q*cos(phi)*(v*k+u*alpha)*exp(-k*z);
         data[n]=(b1*Bl[n]+b2*Bl[n+1])/(r1+r2);
         data[n+1]=(b1*Bl[n+1]-b2*Bl[n])/(r1+r2);
      } else {
         data[n]=0.0;
         data[n+1]=0.0;
      }
      n+=2;
   }
}
for(;fabs(v-1.0/(2.0*delta[2]))<SMALL;v+=1.0/(delta[2]*nn[2])) {
   for(u=0.0;u<=1.0/(2.0*delta[1]);u+=1.0/(delta[1]*nn[1])) {
      data[n]=0.0;
      data[n+1]=0.0;
      n+=2;
   }
   for(u=-(nn[1]/2.0-1.0)/(nn[1]*delta[1]);u<0.0&&fabs(u)>SMALL;
       u+=1.0/(delta[1]*nn[1])) {
      data[n]=0.0;
      data[n+1]=0.0;
      n+=2;
   }
}
for(v=-(nn[2]/2.0-1.0)/(nn[2]*delta[2]);v<0.0&&fabs(v)>SMALL;
    v+=1.0/(delta[2]*nn[2])) {
   for(u=0.0;u<1.0/(2.0*delta[1])&&fabs(u-1./(2.*delta[1]));
       u+=1.0/(delta[1]*nn[1])) {
      q=sqrt(u*u+v*v);
      if(q>=fabs(alpha/(2.0*Pi))) {
         k=sqrt((4.0*Pi*Pi*q*q)-(alpha*alpha));
         r1=sin(phi)*sin(phi)*(u*k-v*alpha)*(u*k-v*alpha);
         r2=4.0*Pi*Pi*q*q*q*q*cos(phi)*cos(phi);
         b1=sin(phi)*(u*k-v*alpha)*(v*k+u*alpha)*exp(-k*z);
         b2=2.0*Pi*q*q*cos(phi)*(v*k+u*alpha)*exp(-k*z);
         data[n]=(b1*Bl[n]+b2*Bl[n+1])/(r1+r2);
         data[n+1]=(b1*Bl[n+1]-b2*Bl[n])/(r1+r2);
         } else {
            data[n]=0.0;
            data[n+1]=0.0;
         }
         n+=2;
      }
      data[n]=0.0;
      data[n+1]=0.0;
      n+=2;
      for(u=-(nn[1]/2.0-1.0)/(nn[1]*delta[1]);u<0.0&&fabs(u)>SMALL;
          u+=1.0/(delta[1]*nn[1])) {
         q=sqrt(u*u+v*v);
         if(q>=fabs(alpha/(2.0*Pi))) {
            k=sqrt((4.0*Pi*Pi*q*q)-(alpha*alpha));
            r1=sin(phi)*sin(phi)*(u*k-v*alpha)*(u*k-v*alpha);
            r2=4.0*Pi*Pi*q*q*q*q*cos(phi)*cos(phi);
            b1=sin(phi)*(u*k-v*alpha)*(v*k+u*alpha)*exp(-k*z);
            b2=2.0*Pi*q*q*cos(phi)*(v*k+u*alpha)*exp(-k*z);
            data[n]=(b1*Bl[n]+b2*Bl[n+1])/(r1+r2);
            data[n+1]=(b1*Bl[n+1]-b2*Bl[n])/(r1+r2);
         } else {
            data[n]=0.0;
            data[n+1]=0.0;
         }
         n+=2;
      }
   }
   free_dvector(Bl,1);
   printf("\n%d. Calculated By(x,y,z=%lf) already!!!",i,z);
}

void calculatebz(i,phi,alpha,z,data,delta,nn)
int i,nn[];
double z,phi,alpha;
double data[],delta[];
{
   int n,idim,ntot;
   double u,v,k,q;
   double b1,b2,r1,r2;
   double *Bl,*dvector();
   void free_dvector();

   ntot=1;
   for(idim=1;idim<=ndim;idim++) ntot=ntot*nn[idim];
   Bl=dvector(1,2*ntot);
   for(n=1;n<=2*ntot;n+=2) {
      Bl[n]=data[n];
      Bl[n+1]=data[n+1];
   }
   n=1;
   for(v=0.0;v<1.0/(2.0*delta[2])&&fabs(v-1./(2.*delta[2]))>SMALL;
       v+=1.0/(delta[2]*nn[2])) {
      for(u=0.0;u<1.0/(2.0*delta[1])&&fabs(u-1./(2.*delta[1]))>SMALL;
          u+=1.0/(delta[1]*nn[1])) {
         q=sqrt(u*u+v*v);
         if(q>=fabs(alpha/(2.0*Pi))) {
            k=sqrt((4.0*Pi*Pi*q*q)-(alpha*alpha));
            r1=sin(phi)*sin(phi)*(u*k-v*alpha)*(u*k-v*alpha);
            r2=4.0*Pi*Pi*q*q*q*q*cos(phi)*cos(phi);
            b1=4.0*Pi*Pi*q*q*q*q*cos(phi)*exp(-k*z);
            b2=2.0*Pi*q*q*sin(phi)*(u*k-v*alpha)*exp(-k*z);
          data[n]=(b1*Bl[n]-b2*Bl[n+1])/(r1+r2);
          data[n+1]=(b1*Bl[n+1]+b2*Bl[n])/(r1+r2);
       } else {
          data[n]=0.0;
          data[n+1]=0.0;
       }
       n+=2;
    }
    data[n]=0.0;
    data[n+1]=0.0;
    n+=2;
    for(u=-(nn[1]/2.0-1.0)/(nn[1]*delta[1]);u<0.0&&fabs(u)>SMALL;
        u+=1.0/(delta[1]*nn[1])) {
       q=sqrt(u*u+v*v);
       if(q>=fabs(alpha/(2.0*Pi))) {
          k=sqrt((4.0*Pi*Pi*q*q)-(alpha*alpha));
          r1=sin(phi)*sin(phi)*(u*k-v*alpha)*(u*k-v*alpha);
          r2=4.0*Pi*Pi*q*q*q*q*cos(phi)*cos(phi);
          b1=4.0*Pi*Pi*q*q*q*q*cos(phi)*exp(-k*z);
          b2=2.0*Pi*q*q*sin(phi)*(u*k-v*alpha)*exp(-k*z);
          data[n]=(b1*Bl[n]-b2*Bl[n+1])/(r1+r2);
          data[n+1]=(b1*Bl[n+1]+b2*Bl[n])/(r1+r2);
       } else {
          data[n]=0.0;
          data[n+1]=0.0;
       }
       n+=2;
    }
 }
 for(;fabs(v-1.0/(2.0*delta[2]))<SMALL;v+=1.0/(delta[2]*nn[2])) {
    for(u=0.0;u<=1.0/(2.0*delta[1]);u+=1.0/(delta[1]*nn[1])) {
       data[n]=0.0;
       data[n+1]=0.0;
       n+=2;
    }
    for(u=-(nn[1]/2.0-1.0)/(nn[1]*delta[1]);u<0.0&&fabs(u)>SMALL;
        u+=1.0/(delta[1]*nn[1])) {
       data[n]=0.0;
       data[n+1]=0.0;
       n+=2;
    }
 }
 for(v=-(nn[2]/2.0-1.0)/(nn[2]*delta[2]);v<0.0&&fabs(v)>SMALL;
     v+=1.0/(delta[2]*nn[2])) {
    for(u=0.0;u<1.0/(2.0*delta[1])&&fabs(u-1./(2.*delta[1]))>SMALL;
        u+=1.0/(delta[1]*nn[1])) {
       q=sqrt(u*u+v*v);
       if(q>=fabs(alpha/(2.0*Pi))) {
          k=sqrt((4.0*Pi*Pi*q*q)-(alpha*alpha));
          r1=sin(phi)*sin(phi)*(u*k-v*alpha)*(u*k-v*alpha);
          r2=4.0*Pi*Pi*q*q*q*q*cos(phi)*cos(phi);
          b1=4.0*Pi*Pi*q*q*q*q*cos(phi)*exp(-k*z);
          b2=2.0*Pi*q*q*sin(phi)*(u*k-v*alpha)*exp(-k*z);
          data[n]=(b1*Bl[n]-b2*Bl[n+1])/(r1+r2);
          data[n+1]=(b1*Bl[n+1]+b2*Bl[n])/(r1+r2);
       } else {
          data[n]=0.0;
          data[n+1]=0.0;
       }
       n+=2;
    }
```

```
      data[n]=0.0;
      data[n+1]=0.0;
      n+=2;
      for(u=-(nn[1]/2.0-1.0)/(nn[1]*delta[1]);u<0.0&&fabs(u)>SMALL;
          u+=1.0/(delta[1]*nn[1])) {
         q=sqrt(u*u+v*v);
         if(q>=fabs(alpha/(2.0*Pi))) {
            k=sqrt((4.0*Pi*Pi*q*q)-(alpha*alpha));
            r1=sin(phi)*sin(phi)*(u*k-v*alpha)*(u*k-v*alpha);
            r2=4.0*Pi*Pi*q*q*q*q*cos(phi)*cos(phi);
            b1=4.0*Pi*Pi*q*q*q*q*cos(phi)*exp(-k*z);
            b2=2.0*Pi*q*q*sin(phi)*(u*k-v*alpha)*exp(-k*z);
            data[n]=(b1*Bl[n]-b2*Bl[n+1])/(r1+r2);
            data[n+1]=(b1*Bl[n+1]+b2*Bl[n])/(r1+r2);
         } else {
            data[n]=0.0;
            data[n+1]=0.0;
         }
         n+=2;
      }
   }
   free_dvector(Bl,1);
   printf("\n%d. Calculated Bz(x,y,z=%lf) already!!!",i,z);
}

\noindent {\bf rotate.c}
{\small
\baselineskip=10pt plus 0.5pt minus 0.5pt
\setlength{\parsep}{25pt}
\begin{verbatim}
/*
   rotate.c ---> Monday 2nd Novenber 1998.
   Modified from rotate1d.c and interpol.c.
*/

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

#define Pi 3.141592653589323
#define small 1e-6

void rotate_interpolate(theta,nn,step,data)
int nn[];
double theta,*data,step[];
{
   int x,y,i,k,inew,jnew,trans;
   int upper1,lower1,upper2,lower2;
   void free_dmatrix(),free_dvector(),polint2();
   double oldx,oldy,fracx,fracy,*xa,*ya,*d;
   double **f,**dmatrix(),*dvector();

   f=dmatrix(1,nn[1],1,nn[2]);
   d=dvector(1,4);
   xa=dvector(1,2);
   ya=dvector(1,2);
   trans=((nn[2]/2)*nn[1])+(nn[1]/2)+1;
   if((nn[1]%2)!=0) {
      upper1=nn[1]/2;
      lower1=-nn[1]/2;
   } else {
      upper1=nn[1]/2-1;
```

```
    lower1=-nn[1]/2;
 }
 if((nn[2]%2)!=0) {
    upper2=nn[2]/2;
    lower2=-nn[2]/2;
 } else {
    upper2=nn[2]/2-1;
    lower2=-nn[2]/2;
 }
 k=1;
 for(jnew=0;jnew<=upper2;jnew++) {
    for(inew=0;inew<=upper1;inew++,k++) {
       oldx=inew*cos(theta)+jnew*sin(theta);
       if(fabs(oldx)<=small) oldx=0.0;
       x=oldx;
       if((oldx-x)<0.0 && fabs(oldx-x)<small) oldx=x;
       if((oldx-x)<0.0 && fabs(oldx-x)>small) x=x-1;
       oldy=jnew*cos(theta)-inew*sin(theta);
       if(fabs(oldy)<=small) oldy=0.0;
       y=oldy;
       if(((oldy-y)<0.0)&&(fabs(oldy-y)<small)) oldy=y;
       if(((oldy-y)<0.0)&&(fabs(oldy-y)>small)) y=y-1;
       if((x<upper1)&&(x>=lower1)&&(y<upper2)&&(y>=lower2)) {
          for(i=1;i<=2;i++) {
             xa[i]=(x+i-1)*step[1];
             ya[i]=(y+i-1)*step[2];
     d[i]=data[trans+y*nn[1]+x+i-1];
     d[i+2]=data[trans+(y+1)*nn[1]+x+i-1];
          }
          polint2(oldx,oldy,xa,ya,d,&f[nn[1]/2+1+inew]
                                      [nn[2]/2+1+jnew]);
       } else f[nn[1]/2+1+inew][nn[2]/2+1+jnew]=0.0;
    }
    for(inew=-1;inew>=lower1;inew--,k++) {
       oldx=inew*cos(theta)+jnew*sin(theta);
       if(fabs(oldx)<=small) oldx=0.0;
       x=oldx;
       if((oldx-x)<0.0 && fabs(oldx-x)<small) oldx=x;
       if((oldx-x)<0.0 && fabs(oldx-x)>small) x--;
       oldy=jnew*cos(theta)-inew*sin(theta);
       if(fabs(oldy)<=small) oldy=0.0;
       y=oldy;
       if((oldy-y)<0.0 && fabs(oldy-y)<small) oldy=y;
       if((oldy-y)<0.0 && fabs(oldy-y)>small) y--;
       if((x<upper1)&&(x>=lower1)&&(y<upper2)&&(y>=lower2)) {
          for(i=1;i<=2;i++) {
             xa[i]=(x+i-1)*step[1];
             ya[i]=(y+i-1)*step[2];
     d[i]=data[trans+y*nn[1]+x+i-1];
     d[i+2]=data[trans+(y+1)*nn[1]+x+i-1];
          }
          polint2(oldx,oldy,xa,ya,d,&f[nn[1]/2+1+inew]
                                      [nn[2]/2+1+jnew]);
       } else f[nn[1]/2+1+inew][nn[2]/2+1+jnew]=0.0;
    }
 }
 for(jnew=-1;jnew>=lower2;jnew--) {
    for(inew=0;inew<=upper1;inew++,k++) {
       oldx=inew*cos(theta)+jnew*sin(theta);
```

```
         if(fabs(oldx)<=small) oldx=0.0;
         x=oldx;
         if((oldx-x)<0.0 && fabs(oldx-x)<small) oldx=x;
         if((oldx-x)<0.0 && fabs(oldx-x)>small) x--;
         oldy=jnew*cos(theta)-inew*sin(theta);
         if(fabs(oldy)<=small) oldy=0.0;
         y=oldy;
         if((oldy-y)<0.0 && fabs(oldy-y)<small) oldy=y;
         if((oldy-y)<0.0 && fabs(oldy-y)>small) y--;
         if((x<upper1)&&(x>=lower1)&&(y<upper2)&&(y>=lower2)) {
            for(i=1;i<=2;i++) {
               xa[i]=(x+i-1)*step[1];
               ya[i]=(y+i-1)*step[2];
       d[i]=data[trans+y*nn[1]+x+i-1];
       d[i+2]=data[trans+(y+1)*nn[1]+x+i-1];
            }
            polint2(oldx,oldy,xa,ya,d,&f[nn[1]/2+1+inew]
                                         [nn[2]/2+1+jnew]);
         } else f[nn[1]/2+1+inew][nn[2]/2+1+jnew]=0.0;
         oldx=inew*cos(theta)+jnew*sin(theta);
      }
      for(inew=-1;inew>=lower1;inew--) {
         oldx=inew*cos(theta)+jnew*sin(theta);
         if(fabs(oldx)<=small) oldx=0.0;
         x=oldx;
         if((oldx-x)<0.0 && fabs(oldx-x)<small) oldx=x;
         if((oldx-x)<0.0 && fabs(oldx-x)>small) x--;
         oldy=jnew*cos(theta)-inew*sin(theta);
         if(fabs(oldy)<=small) oldy=0.0;
         y=oldy;
         if((oldy-y)<0.0 && fabs(oldy-y)<small) oldy=y;
         if((oldy-y)<0.0 && fabs(oldy-y)>small) y--;
         if((x<upper1)&&(x>=lower1)&&(y<upper2)&&(y>=lower2)) {
            for(i=1;i<=2;i++) {
               xa[i]=(x+i-1)*step[1];
               ya[i]=(y+i-1)*step[2];
       d[i]=data[trans+y*nn[1]+x+i-1];
       d[i+2]=data[trans+(y+1)*nn[1]+x+i-1];
            }
            polint2(oldx,oldy,xa,ya,d,&f[nn[1]/2+1+inew]
                                         [nn[2]/2+1+jnew]);
         } else f[nn[1]/2+1+inew][nn[2]/2+1+jnew]=0.0;
      }
   }
   for(jnew=1;jnew<=nn[2];jnew++) {
      for(inew=1;inew<=nn[1];inew++) {
         data[(jnew-1)*nn[1]+inew]=f[inew][jnew];
      }
   }
   free_dvector(ya,1);
   free_dvector(xa,1);
   free_dvector(d,1);
   free_dmatrix(f,1,nn[1],1);
}

void rotate_limit(theta,Xmin,Xmax,Ymin,Ymax)
int *Xmin,*Xmax,*Ymin,*Ymax;
double theta;
{
   int i,j,Xmintmp,Xmaxtmp,Ymintmp,Ymaxtmp;

   Xmaxtmp=*Xmax*cos(theta)-*Ymax*sin(theta);
   Ymaxtmp=*Xmax*sin(theta)+*Ymax*cos(theta);
   Xmintmp=*Xmin*cos(theta)-*Ymin*sin(theta);
   Ymintmp=*Xmin*cos(theta)+*Ymin*sin(theta);
   *Xmax=Xmaxtmp;
   *Ymax=Ymaxtmp;
   *Xmin=Xmintmp;
   *Ymin=Ymintmp;
}
```

printout.c

```
/*
   printout.c --> Sunday 13th September 1998.
   Purpose:  This program use save files for using with idl
             program and vel and velovect plot in x-z, x-y,
             y-z and x-y-z plane.
*/

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

void printoutxz(nn,y,data,save_name)
int nn[],y;
char save_name[];
double data[];
{
   int x;
   FILE *save_file,*fopen();

   save_file=fopen(save_name,"a");
   if(save_file==NULL) {
      printf("\nError in open file %s for save.\n",save_name);
      exit(1);
   }
   for(x=nn[1]/2+1;x<=2*nn[1]*3/4;x+=2) {
      fprintf(save_file,"%12.6lf\n",data[2*(y-1)*nn[1]+x]);
   }
   fclose(save_file);
}

void printoutxy(nn,data,save_name)
int nn[];
char save_name[];
double data[];
{
   int   x,y;
   FILE  *save_file,*fopen();

   save_file=fopen(save_name,"w");
   if(save_file==NULL) {
      printf("\nError in open file %s for save.\n",save_name);
      exit(1);
   }
   for(y=nn[2]/4+1;y<=nn[2]*3/4;y++) {
      for(x=2*nn[2]/4+1;x<=2*nn[1]*3/4;x+=2) {
         fprintf(save_file,"%12.6lf\n",data[2*(y-1)*nn[1]+x]);
      }
   }
   fclose(save_file);
}

void printoutyz(nn,x,data,save_name)
int nn[],x;
char save_name[];
double data[];
{
   int y;
   FILE *save_file,*fopen();

   save_file=fopen(save_name,"a");
   if(save_file==NULL) {
      printf("\nError in open file %s for save.\n",save_name);
      exit(1);
   }
   for(y=1;y<=nn[2];y++)
      fprintf(save_file,"%12.6lf\n",data[2*(y-1)*nn[1]+(2*(x-1)+1)]);
   fclose(save_file);
}

void printoutnn(nn,nz,ndim,save_name)
int nz,ndim,nn[];
char save_name[];
{
   int idim;
   FILE *save_file,*fopen();

   save_file=fopen(save_name,"w");
   if(save_file==NULL) {
      printf("\nError in open file %s for save.\n",save_name);
      exit(1);
   }
   for(idim=1;idim<=ndim;idim++)
      fprintf(save_file,"%d\n",nn[idim]/2);
   fprintf(save_file,"%d\n",nz);
   fclose(save_file);
}

void printoutxyz(nn,data,save_name)
int nn[];
char save_name[];
double data[];
{
   int idim,x,y,z;
   FILE *save_file,*fopen();

   save_file=fopen(save_name,"a");
   if(save_file==NULL) {
      printf("\nError in open file %s for save.\n",save_name);
      exit(1);
   }
   for(y=1;y<=nn[2];y++) {
      for(x=1;x<=2*nn[1];x+=2) {
         fprintf(save_file,"%12.6lf\n",data[2*(y-1)*nn[1]+x]);
      }
   }
   fclose(save_file);
}
```

Appendix E

Minimization Program[1]

Minimization program from the function in section 4.4, using Powell's Method (Appendix B).

mainmin.c

The main program to read input data and call other subroutines for minimization, including printing out data.

function.c

It contains the minimization functional for the minimization program.

powell.c

It contains the minimization method to minimize the functional (Appendix B).

polin.c

We used the Interpoline2d() routine for finding the unknown value from known values (Appendix C).

nrutil.c

It contains routines to reserve and unreserve the memory for arrays (Appendix B).

[1]written by the author

The program:
mainmin.c

```
/*
     Sunday 23th July 1999 --> mainmin.c
         1.  Change Bz to Bx.
         2.  nn[1] is y-direction
             nn[2] is z-direction.

     Tuesday 27th July 1999 --> mainmin.c

     Friday 30th July 1999 --> mainmin.c
          1. Copy mainmin.c -> mainmin.yz
          1. Change Bx to By.
          3. nn[1] is x-direction.
             nn[2] is z-direction.
*/

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

#define   FTOL   1.0e-4

int ndim = 2;
int ntot;
int N;
int nn[3];
double step[3];
double *By;

void main(void)
{
   int       i,idim,iter,j,n;
   char      name_open[25],name_save[25];
   double    ftol,fret,*p,**xi,Lx,Lz,phi,pty,stepy;
   double    f2(),**dmatrix(),*dvector();
   void      powell(),free_dmatrix(),free_dvector();

   FILE      *fp, *fp1,*fopen();

   printf("\nOpen data files : ");
   scanf("%s",name_open);
   fp=fopen(name_open,"r");
   if(fp == NULL) {
      printf("\nError to open file.\n");
      exit(1);
   }
   for(idim=1;idim<=ndim;idim++) {
      printf("nn[%d] = ",idim);
      scanf("%d",&nn[idim]);
   }
   printf("line of sight : ");
   scanf("%lf",&phi);
   stepy = 1.0;
   step[1] = 1.0/cos(phi);
   step[2] = 1.0;

   ntot = 1;
   for(idim=1;idim<=ndim;idim++) ntot*=nn[idim];
   Lx = (nn[1]-1)*step[1];
   Lz = (nn[2]-1)*step[2];
```

```
By = dvector(1,ntot);
for(j=1;j<=nn[2];j++) {
   for(i=1;i<=nn[1];i++) {
      fscanf(fp,"%lf",&By[i+nn[1]*(j-1)]);
      By[i+nn[1]*(j-1)] = fabs(By[i+nn[1]*(j-1)]);
   }
}
fclose(fp);
printf("\nOpen points data: ");
scanf("%s",name_open);
fp = fopen(name_open,"r");
if(fp == NULL) {
   printf("\nError to open file.\n");
   exit(1);
}
printf("\nEnter no. of points = ");
scanf("%d",&n);
printf("\nEnter y constant = ");
scanf("%lf",&pty);
pty *= stepy;
printf("\nSave file name : ");
scanf("%s",name_save);
fp1 = fopen(name_save,"a");
if(fp1==NULL) {
   printf("\nError open to save file.\n");
   exit(1);
}
N=n;
p=dvector(1,n*ndim);
xi=dmatrix(1,n*ndim,1,n*ndim);
for(i=1;i<=ndim*n;i++) {
   fscanf(fp,"%lf",&p[i]);
}
for(i=1;i<=ndim*n;i+=2) {
   p[i]*=step[1];
   p[i+1]*=step[2];
}
fclose(fp);
for(i=1;i<=n*ndim;i++) {
   for(j=1;j<=n*ndim;j++) {
      if(i == j) xi[i][j]=1.0;
      else xi[i][j]=0.0;
   }
}

ftol=FTOL;
printf("\nEntering powell...\n");
powell(p,xi,n*ndim,ftol,&iter,&fret,f2);
printf("\nExiting powell...\n");
for(i=1;i<=n*ndim;i+=2) {
   if(p[i]<0.0) {
      do {
         p[i] += Lx;
      } while(p[i] < 0.0);
   } else if(p[i] > Lx) {
      do {
         p[i] -= Lx;
      } while(p[i] > Lx);
   }
```

```
      if(p[i+1] < 20.0*step[2]) {
         do {
            p[i+1] += 1.0;
         } while(p[i+1] < 20.0*step[2]);
      } else if(p[i+1] > Lz) {
         do {
            p[i+1] -= 1.0;
         } while(p[i+1] > Lz);
      }
   }
   for(i=1;i<=n*ndim;i+=2) {
      fprintf(fp1,"%17.12lf ",p[i]);
      fprintf(fp1,"%17.12lf ",pty);
      fprintf(fp1,"%17.12lf\n",p[i+1]);
   }
   fprintf(fp1,"\nf = %17.12lf\n",fret);

   fclose(fp1);
   free_dmatrix(xi,1,n*ndim,1,n*ndim);
   free_dvector(p,1,n*ndim);
   free_dvector(By,1,ntot);
}
```

function.c

```
/*
     Sunday 23th July 1999 --> function.c
           Change boundary and variable in subroutine f2() from
     x-y plane to  y-z plane. And change boundary of z axis.
           Change Bz to Bx in line 17 and function f2() line 224 too.

     Friday 30th July 1999 --> function.c
           Change Bx to By and Ly to Lx in f2().
*/

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

extern int ncom;  /* from pp.317  */
extern int ndim;
extern int N;
extern int ntot;
extern int nn[3];
extern double step[3];
extern double *Bx;
extern double *Bz;
extern double *By;
extern double *pcom, *xicom, (*nrfunc)();

/* extern void (*nrdfun)();  */

static int check = 0;

double f1dim(x)
double x;
{
   int j;
   double f, *xt, *dvector();
   void free_dvector();

   xt=dvector(1,ncom);
   for(j=1;j<=ncom;j++) xt[j]=pcom[j] + x*xicom[j];
   f=(*nrfunc)(xt);
   free_dvector(xt,1,ncom);
   return f;
}

double f2(xin)
double *xin;
{
   int     i, j, j1, j2, idim;
   void    nrerror(), free_dvector();
   double  Interpolin2d();
   double  x[N+1], z[N+1], V[N+1];
   double  r, rmin1, rmin2, Lx, Lz;
   double  xc, zc, xmin1, zmin1, xmin2, zmin2, xtemp, ztemp;
   double  Bi, Bmin1, Bmin2, Bc, Btmp_avg, B_avg;
   double  S, V_avg, area;
   double  *dvector();

   Lx = (nn[1]-1)*step[1];
   Lz = (nn[2]-1)*step[2];

   for(i=1;i<=N;i++) {
      x[i] = xin[2*i-1];
      if(x[i] < 0.0) {
         do {
           x[i] += Lx;
         } while(x[i] < 0.0);
      } else if(x[i] > Lx) {
         do {
            x[i] -= Lx;
         } while(x[i] > Lx);
      }
      z[i] = xin[2*i];
      if(z[i] < 20.0*step[2]){
         do {
            z[i] += 1.0;
         } while(z[i] < 20.0*step[2]);
      } else if(z[i] > Lz) {
         do {
            z[i] -= 1.0;
         } while(z[i] > Lz);
      }
   }
   B_avg = 0.0;
   for(i=1;i<=N;i++) {
      Btmp_avg = Interpolin2d(nn,x[i],z[i],By,step);
      B_avg += Btmp_avg;
   }
   V_avg = B_avg/N;
   for(i=1;i<=N;i++) {
      j1 = -1;
      j2 = -2;
      rmin1 = 999999;
      rmin2 = 9999999;
      for(j=1;j<=N;j++) {
         if(i != j) {
            if(x[j] > (x[i] + Lx/2.0)) {
               xtemp = x[j] - Lx;
            } else if(x[j] < (x[i] - Lx/2.0)) {
               xtemp = x[j] + Lx;
      } else {
         xtemp = x[j];
      }
      ztemp = z[j];

      r = sqrt((xtemp-x[i])*(xtemp-x[i])
          +(ztemp-z[i])*(ztemp-z[i]));
      if(r < rmin1) {
         j2 = j1;
         rmin2 = rmin1;
         j1 = j;
         rmin1 = r;
      } else if(r < rmin2) {
         j2 = j;
         rmin2 = r;
      }
   }
}
if(j1 < 0 || j2 < 0) nrerror("f2: bad j1 or j2");
xmin1 = x[j1];
zmin1 = z[j1];
xmin2 = x[j2];
zmin2 = z[j2];
   Bi = Interpolin2d(nn,x[i],z[i],By,step);
Bmin1 = Interpolin2d(nn,xmin1,zmin1,By,step);
Bmin2 = Interpolin2d(nn,xmin2,zmin2,By,step);
if(xmin1 > x[i] + Lx/2.0) {
   xmin1 = xmin1 - Lx;
} else if(xmin1 < x[i] - Lx/2.0) {
   xmin1 = xmin1 + Lx;
} else {
   xmin1 = xmin1;
}
if(xmin2 > x[i] + Lx/2.0) {
   xmin2 = xmin2 - Lx;
} else if(xmin2 < x[i] - Lx/2.0) {
   xmin2 = xmin2 + Lx;
} else {
   xmin2 = xmin2;
}
if(zmin1 > z[i]) {
   zmin1 = z[i];
} else if(zmin1 < z[i]) {
   zmin1 = zmin1;
} else {
   zmin1 = zmin1;
}
if(zmin2 > z[i]) {
   zmin2 = z[i];
} else if(zmin2 < z[i]) {
   zmin2 = zmin2;
} else {
   zmin2 = zmin2;
}
area = fabs((xmin1-x[i])*(zmin2-z[i])
       -(xmin2-x[i])*(zmin1-z[i]))/2.0;
xc = (x[i] + xmin1 + xmin2)/3.0;
zc = (z[i] + zmin1 + zmin2)/3.0;
if(xc < 0.0) {
   do {
      xc += Lx;
      } while(xc < 0.0);
   } else if(xc > Lx) {
      do {
         xc -= Lx;
      } while(xc > Lx);
   }
   if(zc < 20.0*step[2]) {
      do {
         zc += 1.0;
      } while(zc < 20.0*step[2]);
   } else if(zc > Lz) {
      do {
         zc -= 1.0;
      } while(zc > Lz);
   }

   Bc = Interpolin2d(nn,xc,zc,By,step);
   V[i] = ((Bi+Bmin1+Bmin2)/6.0 + Bc/2.0)*area;
  }
  S = 0.0;
  for(i=1;i<=N;i++) S+=(V[i]-V_avg)*(V[i]-V_avg);
  return S;
}
```

Appendix F

Program to Trace Field Lines[1]

The program to trace the magnetic fields in 3D comprises:

FstepL3d.c

The program traces the magnetic field calculated from eq. (4.10) in section 4.3. The program gets the 3D values of the magnetic field from Appendix C and minimized initial points from Appendix D to trace a field line.

polin.c

See Appendix C.

outpoint.c

Prints out the (x, y, z) coordinates of each line.

nrutil.c

It contains routines to reserve and unreserve the memory for arrays (Appendix B).

The program:
mainmin.c

```
/*
   Tuesday 22nd December 1998 --> FstepL3d.c
          First write this program.

   19th April 1999 --> FstepL3d.c
          I add input phi angle from data's day I use.  And Change the
   step of y by
                        step[2] = step[1]/cos(phi).

   Sunday 18th July 1999 --> FstepL3d.c
          Add xi1 and yi2 to the real position before adding
   zero around the real data.
```

[1] written by the author

```
   Monday 19th July 1999 --> FstepL3d.c
          Change xi1 and yi2 minus with nn/2.

   Thursday 22th July 1999 --> FstepL3d.c
          Change input coordinate in 3 values.
*/

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

#define  Pi 4.0*atan(1.0)
int ndim=3;

void main(void)
{
   int      i, j, k, x, y, z, ntot;
   int      nn[ndim+1], Numpt, n;
   char     n_f1[20], n_f2[20], n_f3[20];
   char     n_pt[20], n_s[20];
   FILE     *fp1, *fp2, *fp3, *fs,*fopen();
   void     checkfile(),outpoint3d();
   double   xi,yi,zi,xi1,yi1,L[ndim+1],phi;
   double   FIELD,deltas,deltastmp,step[ndim+1];
   double   *xpt,*ypt,*zpt,Ftemp;
   double   FD_PT[ndim+1],*Fx,*Fy,*Fz,*dvector();
   double   Interpolin2d(),Interpolin3d();

   /* Read Magnetic field data files calculated in 3D
      by FFT method. */
   printf("\nOpen first file  : ");
   scanf("%s",n_f1);
   if((fp1=fopen(n_f1,"r"))==NULL) {
      printf("\nError open file '%s' for read.\n",n_f1);
      exit(1);
   }
   printf("\nOpen second file : ");
   scanf("%s",n_f2);
   if((fp2=fopen(n_f2,"r"))==NULL) {
      printf("\nError open file '%s' for read.\n",n_f2);
      exit(1);
   }
   printf("\nOpen third file  : ");
   scanf("%s",n_f3);
   if((fp3=fopen(n_f3,"r"))==NULL) {
      printf("\nError open file '%s' for read.\n",n_f3);
      exit(1);
   }
   /* Save coordinates of a magnetic field line. */
   printf("\nSave file : ");
   scanf("%s",n_s);

   /* Read number of data in each dimension. */
   for(i=1;i<=ndim;i++) {
      printf("\nnn[%d] = ",i);
      scanf("%d",&nn[i]);
   }
   /* Read phi angle of initial day we use. */
   printf("\nphi angle (degree) = ");
   scanf("%lf",&phi);
```

```
phi *= Pi/180.0;
/* Total number of magnetic data. */
ntot = 1;
for(i=1;i<=ndim;i++) ntot*=nn[i];
/* Step size of each dimensions should be like delta[] in
   FFT program. */
step[1]=1.0/cos(phi);
step[2]=1.0;
step[3]=1.0;
/* Length of each dimensions. */
for(i=1;i<=ndim;i++) {
   L[i] = (nn[i]-1)*step[i];
}
/* Read data file */
Fx = dvector(1,ntot);
Fy = dvector(1,ntot);
Fz = dvector(1,ntot);
for(k=1;k<=nn[3];k++) {
   for(j=1;j<=nn[2];j++) {
      for(i=1;i<=nn[1];i++) {
         fscanf(fp1,"%lf",&Fx[i+(j-1)*nn[1]+(k-1)*nn[1]*nn[2]]);
         fscanf(fp2,"%lf",&Fy[i+(j-1)*nn[1]+(k-1)*nn[1]*nn[2]]);
         fscanf(fp3,"%lf",&Fz[i+(j-1)*nn[1]+(k-1)*nn[1]*nn[2]]);
      }
   }
}
fclose(fp3);
fclose(fp2);
fclose(fp1);
/* Read origin points to find the field lines from file. */
printf("\nOpen initial point data file : ");
scanf("%s",n_pt);
if((fp1=fopen(n_pt,"r"))==NULL) {
   printf("\nError open in file '%s'.\n",n_pt);
   exit(1);
}
printf("\nNumber of initial points = ");
scanf("%d",&Numpt);
printf("\n deltas = ");
scanf("%lf",&deltas);
deltastmp = deltas;
xpt = dvector(1,Numpt);
ypt = dvector(1,Numpt);
zpt = dvector(1,Numpt);
/* Initial points in 3d.*/
for(i=1;i<=Numpt;i++) {
   fscanf(fp1,"%lf",&xpt[i]);
   fscanf(fp1,"%lf",&ypt[i]);
   fscanf(fp1,"%lf",&zpt[i]);
}
fclose(fp1);
/* Begin draw line in each initial which magnetic field < 0.0 */
for(i=1;i<=Numpt;i++) {
   /* Save file which will obtain the coordinate. */
   fs = fopen(n_s,"w");
   if(fs==NULL) {
      printf("\nError to open file %s ",n_s);
      exit(0);
```

```
}
/* Begin find magnetic field line. */
deltas=-fabs(deltastmp);
n = 1;
xi = xpt[i];
yi = ypt[i];
zi = zpt[i];
for(;;) {
   if((xi<0.0||xi>L[1])||(yi<0.0||yi>L[2])||(zi<0.0||zi>L[3])) {
      printf("\nOut of ranges in the first loop at ");
      if(xi<0.0||xi>L[1]) printf("x = %13.6lf",xi);
      else if(yi<0.0||yi>L[2]) printf("y = %13.6lf",yi);
      else printf("z = %13.6lf",zi);
      break;
   }
   xi1=xi-L[1]/2.0;
   yi1=yi-L[2]/2.0;
   fprintf(fs,"%13.6lf %13.6lf %13.6lf\n",xi1,yi1,zi);
   FD_PT[1] = Interpolin3d(ndim,nn,xi,yi,zi,step,Fx);
   FD_PT[2] = Interpolin3d(ndim,nn,xi,yi,zi,step,Fy);
   FD_PT[3] = Interpolin3d(ndim,nn,xi,yi,zi,step,Fz);
   FIELD = sqrt(FD_PT[1]*FD_PT[1]+FD_PT[2]*FD_PT[2]
           +FD_PT[3]*FD_PT[3]);
   xi = ((FD_PT[1]*deltas)/FIELD)+xi;
   yi = ((FD_PT[2]*deltas)/FIELD)+yi;
   zi = ((FD_PT[3]*deltas)/FIELD)+zi;
   if((zi>2.5) && (n==1)) {
      deltas = deltas*10.0;
      n++;
   } else if((zi<=2.5) && (n==2)) {
      deltas = deltas/10.0;
      n--;
   }
}
deltas=fabs(deltastmp);
n = 1;
xi = xpt[i];
yi = ypt[i];
zi = zpt[i];
for(;;) {
   if((xi<0.0||xi>L[1])||(yi<0.0||yi>L[2])||(zi<0.0||zi>L[3])) {
      printf("\nOut of ranges in the second loop at ");
      if(xi<0.0||xi>L[1]) printf("x = %lf",xi);
      else if(yi<0.0||yi>L[2]) printf("y = %lf",yi);
      else printf("z = %lf",zi);
      break;
   }
   xi1=xi-L[1]/2.0;
   yi1=yi-L[2]/2.0;
   fprintf(fs,"%13.6lf %13.6lf %13.6lf\n",xi1,yi1,zi);
   FD_PT[1] = Interpolin3d(ndim,nn,xi,yi,zi,step,Fx);
   FD_PT[2] = Interpolin3d(ndim,nn,xi,yi,zi,step,Fy);
   FD_PT[3] = Interpolin3d(ndim,nn,xi,yi,zi,step,Fz);
   FIELD = sqrt(FD_PT[1]*FD_PT[1]+FD_PT[2]*FD_PT[2]
           +FD_PT[3]*FD_PT[3]);
   xi = ((FD_PT[1]*deltas)/FIELD)+xi;
   yi = ((FD_PT[2]*deltas)/FIELD)+yi;
   zi = ((FD_PT[3]*deltas)/FIELD)+zi;
   if((zi>3.0) && (n==1)) {
```

```
            deltas*=10.0;
            n++;
         } else if((zi<=3.0) && (n==2)) {
            deltas/=10.0;
            n--;
         }
      }
      fclose(fs);
      if(n_s[5]=='9') {
         n_s[4]++;
         n_s[5]='0';
      } else {
         n_s[5]++;
      }
   }
   free_dvector(zpt,1);
   free_dvector(ypt,1);
   free_dvector(xpt,1);
   free_dvector(Fz,1);
   free_dvector(Fy,1);
   free_dvector(Fx,1);
}
```

outpoint.c

```
/*
   outpoint.c --> Monday 21th December 1998
*/

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

void outpoint2d(x,y,save_name)
char save_name[];
double x,y;
{
   FILE *fp,*fopen();

   if((fp=fopen(save_name,"a"))==NULL) {
      printf("\nError written in this file.");
      exit(1);
   }
   fprintf(fp,"%10.6lf %10.6lf\n",x,y);
   fclose(fp);
}

void outpoint3d(x,y,z,save_name)
char save_name[];
double x,y,z;
{
   FILE *fp,*fopen();

   if((fp=fopen(save_name,"a"))==NULL) {
      printf("\nError wriiten in this file.");
      exit(1);
   }
   fprintf(fp,"%10.6lf %10.6lf %10.6lf\n",x,y,z);
   fclose(fp);
}
```

Appendix G

Image Drawing Program[1]

The image drawing method in section 4.5.4 can be writted in the programs:

rmain.c

This program is used for rotating from system 3 to system 8 and inputting initial and final dates and times.

nrutil.c

It contains routines to reserve and unreserve the memory for arrays (Appendix B).

The program:
rmain.c

```
/*
   Monday 1st February 1999 --> rmain.c
   Tuesday 29th June 1999 --> rmain.c
       Remove the transformation in spherical coordinate.  And add the
       Transformation in step:
       1. Place the loop coordinate on the sphere.
       2. Rotate the magnetic to the real position. (Change coordinate
          from the center.)
       3. Turn up the sun's globe.
       4. Rotate the sun's globe to the day we want.
       5. Turn down the sun's globe.
       ==> rc10
   Thursday 15th July 1999 --> rmain.c
       Comment initial theta.
   Monday 19th July 1999 --> rmain.c
       Change print out to file by add z>=0.

*/

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

[1]written by the author

```
#define R       697000.00  /*242.3962875 arcsec*/
                           /*697000 unit km*/
#define omega   2.443460953e-6 /*unit radius/second*/
#define arc     3287.735849 /*unit km*/
                            /*use for change unit arcsecond -> km*/
#define Pi      4.0*atan(1.0)

int ndim = 3;

void main(void)
{
   int i, j, k, i1, j1, k1, no, no_file, no_sec();
   int i_yr, i_month, i_day, i_hr, i_min, i_sec;
   int e_yr, e_month, e_day, e_hr, e_min, e_sec;
   char n_fp1[25], n_fs1[25];
   FILE *fp1, *fs1, *fopen();
   double phi0, r, theta, phi, iangle, rotheta, phi_ro;
   double x, y, z, x0, y0, z0, x1, y1, z1, x2, y2, z2;
   double x4, y4, z4, x5, y5, z5, x3, y3, z3, deltat;

   printf("\nOpen file : ");
   scanf("%s",n_fp1);
   printf("\nNumber of files = ");
   scanf("%d",&no_file);
   printf("Ratate theta (degree) = ");
   scanf("%lf",&rotheta);
   rotheta *= Pi/180.0;
/*
   printf("\nInitial theta (degree) = ");
   scanf("%lf",&theta0);
   theta0 *= Pi/180.0;
*/
   printf("Initial phi (degree) = ");
   scanf("%lf",&phi0);
   phi0 *= Pi/180.0;
   printf("\nInclination (degree) = "); /* >= 0 if inclinate face.
                                            < 0 if rotate back. */
   scanf("%lf",&iangle);
   iangle *= Pi/180.0;
   /*# Time : */
   printf("\nInitial   day = "); scanf("%d",&i_day);
   printf("        Month = ");   scanf("%d",&i_month);
   printf("         Year = ");   scanf("%d",&i_yr);
   printf("Time    hours = ");   scanf("%d",&i_hr);
   printf("       minute = ");   scanf("%d",&i_min);
   printf("       second = ");   scanf("%d",&i_sec);
   printf("\nEnd       day = "); scanf("%d",&e_day);
   printf("        Month = ");   scanf("%d",&e_month);
   printf("         Year = ");   scanf("%d",&e_yr);
   printf("  Time  hours = ");   scanf("%d",&e_hr);
   printf("       minute = ");   scanf("%d",&e_min);
   printf("       second = ");   scanf("%d",&e_sec);
   deltat = (double) no_sec(i_yr,i_month,i_day,i_hr,i_min,i_sec,
                            e_yr,e_month,e_day,e_hr,e_min,e_sec);
   printf("\n>> Seconds = %13.6lf",deltat);
   printf("\nEnter save file name : ");
   scanf("%s",n_fs1);
   for(i=1;i<=no_file;i++) {
```

```
if((fp1=fopen(n_fp1,"r")) == NULL) {
   printf("\nError to open file.\n");
   exit(1);
}
if((fs1=fopen(n_fs1,"w")) == NULL) {
   printf("\nError to save file.\n");
   exit(1);
}
for(;;) {
   fscanf(fp1,"%lf",&x);
   fscanf(fp1,"%lf",&y);
   fscanf(fp1,"%lf",&z);
   x*=arc;
   y*=arc;
   z*=arc;
   if(feof(fp1)==0) { /*# Check end of file. */

 /*#1 Put the magnetic field on the direct position in
      cartesian coordinate. */
      x1 = R*sin(phi0) + x*cos(phi0) + z*sin(phi0);
      y1 = y;
      z1 = R*cos(phi0) - x*sin(phi0) + z*cos(phi0);

 /*#2 Rotate to the real position. --> z = constant*/
      x2 = x1*cos(rotheta) + y1*sin(rotheta);
      y2 = y1*cos(rotheta) - x1*sin(rotheta);
      z2 = z1;

 /*#3 Transform the first up inclination. */
      x3 = x2;
      y3 = y2*cos(iangle) + z2*sin(iangle);
      z3 = -y2*sin(iangle) + z2*cos(iangle);

 /*#4 Rotation to date we interested. */
      phi_ro = omega*deltat;
      x4 = x3*cos(phi_ro) + z3*sin(phi_ro);
      y4 = y3;
      z4 = -x3*sin(phi_ro) + z3*cos(phi_ro);

 /*#5 Transformation inclination to old angle. */
      x5 = x4;
      y5 = y4*cos(iangle) - z4*sin(iangle);
      z5 = y4*sin(iangle) + z4*cos(iangle);
      if((x5*x5+y5*y5)>=(R*R) || z5>=0)
         fprintf(fs1,"%13.6lf %13.6lf %13.6lf\n",x5,y5,z5);
   } else {
      printf("\nAlready to rotate file");
      break;
   }
}
fclose(fp1);
printf("\nClose file %s for read already.",n_fp1);
fclose(fs1);
printf("\nSave file %s already.",n_fs1);

if(n_fp1[5]=='9') {
   if(n_fp1[4]=='9') {
      n_fp1[3]++;
      n_fp1[4]='0';
      n_fp1[5]='0';
```

```
          } else {
             n_fp1[4]++;
             n_fp1[5]='0';
          }
       } else {
          n_fp1[5]++;
       }
       if(n_fs1[5]=='9') {
          if(n_fs1[4]=='9') {
             n_fs1[3]++;
             n_fs1[4]='0';
             n_fs1[5]='0';
          } else {
             n_fs1[4]++;
             n_fs1[5]='0';
          }
       } else {
          n_fs1[5]++;
       }
    }
    if(no_file>=1) {
       fs1 = fopen(n_fs1,"w");
       if(fs1 == NULL) {
          printf("\nError to save file.\n");
          exit(1);
       }
       for(j=0;j<=360;j++) {
          x = R*cos((double) j*Pi/180.0);
          y = R*sin((double) j*Pi/180.0);
          z = 0.0;
          fprintf(fs1,"%13.6lf %13.6lf %13.6lf\n",x,y,z);
       }
       printf("\nClose file %s for read already.",n_fp1);
    }
    printf("\n");
 }

int no_sec(i_yr,i_month,i_day,i_hr,i_min,i_sec,
           e_yr,e_month,e_day,e_hr,e_min,e_sec)
int i_yr,i_month,i_day,i_hr,i_min,i_sec;
int e_yr,e_month,e_day,e_hr,e_min,e_sec;
{
   int sum_day,sum_hr,sum_min,sum_sec;
   int i,ndpm(),ndpy();

   if(abs(e_yr-i_yr)!=0) {
      sum_day = ndpm(i_month,i_yr)-i_day;
      for(i=i_month;i<=12;i++) sum_day += ndpm(i,i_yr);
      for(i=i_yr+1;i<=e_yr-1;i++) sum_day += ndpy(i);
      for(i=1;i<=e_month-1;i++) sum_day += ndpm(i,e_yr);
      sum_day += (e_day-1);
      sum_hr = (sum_day*24) + (24-i_hr-1) + e_hr;
      sum_min = (sum_hr*60) + (60-i_min-1) + e_min;
      sum_sec = (sum_min*60) + (60-i_sec) + e_sec;
   } else {
      if(abs(e_month-i_month)!=0) {
         sum_day = 0;
         for(i=i_month+1;i<=e_month-1;i++) sum_day += ndpm(i,i_yr);
         sum_day = (ndpm(i_month,i_yr)-i_day-1)+e_day;
         sum_hr = (sum_day*24) + (24-i_hr-1) + e_hr;
```

```
            sum_min = (sum_hr*60) + (60-i_min-1) + e_min;
            sum_sec = (sum_min*60) + (60-i_sec) + e_sec;
         } else {
            if(abs(e_day-i_day)!=0) {
               sum_day = 0;
               sum_day = e_day-i_day-1;
               sum_hr = (sum_day*24) + (24-i_hr-1) + e_hr;
               sum_min = (sum_hr*60) + (60-i_min-1) + e_min;
               sum_sec = (sum_min*60) + (60-i_sec) + e_sec;
            } else {
               if(abs(e_hr-i_hr)!=0) {
                  sum_hr = e_hr-i_hr-1;
                  sum_min = (sum_hr*60) + (60-i_min-1) + e_min;
                  sum_sec = (sum_min*60) + (60-i_sec) + e_sec;
               } else {
                  if(abs(e_min-i_min)!=0) {
                     sum_min = e_min-i_min-1;
                     sum_sec = (sum_min*60) + (60-i_sec) + e_sec;
                  } else {
                     sum_sec = e_sec-i_sec;
                  }
               }
            }
         }
   }
   return sum_sec;
}

int ndpy(yr)
int yr;
{
   int no;

   if(yr%4==0) no = 366;
   else no = 365;
   return no;
}

int ndpm(m,yr)
int m,yr;
{
   int no;

   if(m==1) no = 31;
   else if(m==2) {
      if(yr%4==0) no = 29;
      else no = 28;
   } else if(m==3) no = 31;
   else if(m==4) no = 30;
   else if(m==5) no = 31;
   else if(m==6) no = 30;
   else if(m==7) no = 31;
   else if(m==8) no = 31;
   else if(m==9) no = 30;
   else if(m==10) no = 31;
   else if(m==11) no = 30;
   else if(m==12) no = 31;
   return no;
}
```

Curriculum Vitae

Paisan Tooprakai,

1974	Born : Apr, 2nd 1974 in Lampang, THAILAND.
	Father : Nirat Tooprakai.
	Mother : Nawarat Tooprakai.
1992-1996	Bachelor of Science (Physics),
	Chiangmai University, Chiangmai, THAILAND.