```
float rerr=RERR;          /*relative error tolerance*/
float T = 0;              /*slow time*/
float Tout = 0;           /*time end point for numerical
                            integration*/
float A[4];               /*amplitudes -- B[0] is not used*/
float AA[3][1000];        /*array of amplitudes to be plotted*/
float TT[1000];           /*array of times to be plotted*/
float wk[184];            /*float work array used by ODE_*/
const int n = 4;          /*dimension of A array*/
int ind = 1;              /*index used by ODE_*/
int iwk[9];               /*int work array used by ODE_*/
FILE *out;                /*output file */

out = fopen("friction.dat","w");/*open the data output file*/

k[1] = BASE_WAVE_NUMBER;
i = triad(k,omega,z,b);  /*find the resonant triad*/
i = Nj(N,b,H2,z);     /*compute the normalization constants*/
i = cj(c,k,omega);        /*compute the phase velocities*/


r[1] = -R_/c[1]; /*compute the first friction coefficient*/
r[2] = -R_/c[2]; /*compute the second friction coefficient*/
r[3] = -R_/c[3]; /*compute the third friction coefficient*/

i = Kjlm(K,b,H2,N,k,z);/*compute the interaction coefficients*/
KK[1] = c[1]*(K[1][2][3] + K[1][3][2]);
KK[2] = c[2]*(K[2][3][1] + K[2][1][3]);
KK[3] = c[3]*(K[3][1][2] + K[3][2][1]);

for (i =0; i <1000; i++) /*start numerical integration of ode's*/
{
   A[1] = AMP10;          /*initialize the amplitudes*/
   A[2] = AMP20;          /*initialize the amplitudes*/
   A[3] = AMP30;          /*initialize the amplitudes*/

   Tout = Tout + STEP_SIZE;/*set the right end point of
                             integration*/

   /*ODE is a FORTRAN differential equation solver */
   /*see "NSWC LIBRARY OF MATHEMATICS SUBROUTINES (1980)"
     pp. 461-463*/
   ode_(derivative,&n,A,&T,&Tout,&rerr,&aerr,&ind,wk,iwk);

   switch (ind) {
   case 1: break;
   case 2: break;
```

```
case 3:
   printf("WARNING: error tolerance too small. T = %g\n",T);
   Tout = T;
   break;
case 4:
   printf("WARNING: MAXNUM steps were performed. T = %g\n",T);
   Tout = T;
   break;
case 5:
   printf("WARNING: Equation appeas to be stiff. T = %g\n",T);
   break;
case 6:
   printf("WARNING: AERR is set to zero. T= %g\n",T);
   Tout = T;
   break;
case 7:
   printf("ERROR: no computations were performed.T = %g\n",T);
   break;
case -1: break;
case -2: break;
case -3:
   printf("WARNING: error tolerance too small. T = %g\n",T);
   Tout = T;
   break;
case -4:
   printf("WARNING: MAXNUM steps were performed. T = %g\n",T);
   Tout = T;
   break;
case -5:
   printf("WARNING: Equation appeas to be stiff. T = %g\n",T);
   Tout = T;
   break;
case -6:
   printf("WARNING: AERR is set to zero. T = %g\n",T);
   Tout = T;
   break;
case -7:
   printf("ERROR: no computations were performed. T = %g\n",T);
   break;
}
AA[0][i] = A[1];
AA[1][i] = A[2];
AA[2][i] = A[3];
TT[i] = Tout/(EPSILON*CORIOLIS_PARAMETER*60*60*24);

fprintf(out,"%.14e %.14e %.14e %.14e \n",TT[i],A[1],
        A[2],A[3]);
```

```
}
   /*make the graph of the dispersion relation*/
   {
     static char xnote[]="time (days)$";
     static char ynote[]="amplitude (nondimensionalized)$";
     static char title[40];
     int ier;
     const int n = 3;
     const int size = 1000;
     const int _0 = 0;
     const int _1 = 1;
     const int _2 = 2;
     const int _6 = 6;
     const double R = R_;
     sprintf(title,"Amplitude -vs- Time: r = %g $",R);

     gopks_(&_6,&ier);
     gopwk_(&_1,&_2,&_1);
     gacwk_(&_1);
     anotat_(xnote,ynote,&_1,&_1,&_0,&_0);
     ezmxy_(TT,AA,&size,&n,&size,title);
     gdawk_(&_1);
     gclwk_(&_1);
     gclks_();
   }
   fclose(out);
}

int derivative(float time,  /*time*/
               float A[4],  /*amplitude*/
               float dA[4]) /*derivative of amplitude
                              w.r.t. time*/
{
dA[0] = 0;
dA[1] = (float) -KK[1]*A[2]*A[3]-r[1]*A[1];
dA[2] = (float) -KK[2]*A[3]*A[1]-r[2]*A[2];
dA[3] = (float) -KK[3]*A[1]*A[2]-r[3]*A[3];
}
```

Appendix 5

```
/*************************************************************/
/* Name: topo                                                */
/*                                                           */
/* Purpose: This program plots the solution for a resonant   */
/*          triad of waves  over a piece wise constant       */
/*          topographic configuration.                       */
/*                                                           */
/* Written: Feb 13,1992                                      */
/* By: Francois Primeau                                      */
/*                                                           */
/*************************************************************/
#include <stdio.h>
#include <math.h>

FILE *out;

main()
{
double h;          /* jump parameter */
double H;          /* alongshore topographic paramter */
double mu0;        /* alongshore topographic coefficients */
double mu[4];      /* alongshore topographic coefficients */
double cg[4];      /* group velocities */
double bb,H1,H2;   /* cross-shelf profile parameters */
double k[4];       /* triad of wave numbers */
double omega[4];   /* triad of frequencies */
double z[4];      /*triad of roots to transcendental equation*/
double b0[4];      /*initial amplitude envelopes */
double b0hat[4];   /*scaled initial amplitude envelopes */
double phi0[4];    /*initial phases*/
double K0[4];      /*interaction coefficients*/
double PHI;        /* phase envelope variable */
double PHI0;       /* initial phae envelope */
double Gamma;      /*constant of the motion*/
double yhat[4];    /*roots of the cubic equation*/
double bhat[4];    /* scaled amplitude envelopes */
double b[4];       /* amplitude envelopes */
double phi[4];     /* phase envelopes */
double w[4];       /* weight of Pi/2 attributed to each inital
                      phase */
double zeta;       /* zeta = Y - Ys */
double zetanot;    /* yhat(zetanot) = 0 */
double Y;          /* alongshore slow variable */
double Yp;         /* right endpoint of current alongshore
                      topographic segment */
double Yn;         /* left endpoint of current alongshore
                      topographic segment */
```

```
double Ys;        /* shift parameter */
double khat;      /* modulus */
double thetahat; /* sn phase shift */
double step;      /* number of Y interval steps at which to
                     calculate the solution */
double step_size;/* size of Y interval steps */
int i,j,n,ierr;
double TEST;
double SN,CN,DN;
double s[4];
double S;
FILE *in;         /* file from which to readin the parameters
                     and coefficients */

   in = fopen("topo.dat","r"); /* open the input file */
   out = fopen("out1.dat","w");/* open the output data file*/

   Yp = 0;  /* start simulation at Y = 0.0 */
   i = init(in,b0,w,K0,&mu0,mu,cg,&bb,&H1,&H2,k,omega,z);
   i = scale(b0hat,b0,K0);
   PHI0 = -1*M_PI/2;
   TEST = -1*M_PI/2;
   phi0[1] = w[1]*PHI0;
   phi0[2] = w[2]*PHI0;
   phi0[3] = w[3]*PHI0;
   s[1] = 1.0;
   s[2] = 1.0;
   s[3] = 1.0;
   S = 1.0;
   while (astopo(in,&Yn,&Ys,&zetanot,&H,&step) == 0 )
   {
     h = H;
     Y = Yp;
     zeta = Yp - Ys;
     printf("H: %.14e Yn: %.14e  Yp: %.14e \n", h,Yn,Yp);
     printf(
       "b0hat[1]: %.14e  b0hat[2]: %.14e  b0hat[3] %.14e \n",
        b0hat[1],b0hat[2],b0hat[3]);
     printf("PHI0: %.14e   cos(PHI0): %.14e \n",PHI0,cos(PHI0));
     Gamma = b0hat[1]*b0hat[2]*b0hat[3]*cos(PHI0)+
             mu0*h*b0hat[1]*b0hat[1]/2;
     printf("Gamma: %.14e \n",Gamma);
     i = cubic(yhat,b0hat[1],b0hat[2],b0hat[3],Gamma,mu0,h);
     printf("y1: %.14e  y2: %.14e  y3: %.14e \n",yhat[1],
             yhat[2],yhat[3]);
     khat = sqrt((yhat[1]-yhat[2])/(yhat[1]-yhat[3]));
     if ( invsn(&thetahat,
```

```
                sqrt(yhat[1]/(yhat[1]-yhat[2])),khat
                )!=0) error(1);
        step_size = (Yn - Yp)/step;
        i = solve(bhat,phi,s,S,&PHI,&TEST,yhat,b0hat,phi0,PHI0,
                  Gamma,mu0,h,khat,thetahat,zeta+ 10*step_size,
                  zetanot,&SN,&CN,&DN);
        if ( ABS(cos(PHI) - cos(TEST)) > 1e-2  ) S = -1*S;
        printf("S: %.14e \n",S);
        while( Yn <= Y)
        {
           i = solve(bhat,phi,s,S,&PHI,&TEST,yhat,b0hat,phi0,
                     PHI0,Gamma,mu0,h,khat,thetahat,zeta,zetanot,
                     &SN,&CN,&DN);
           phi[1] = phi[1] - h*mu[1]*zeta/cg[1];
           phi[2] = phi[2] - h*mu[2]*zeta/cg[2];
           phi[3] = phi[3] - h*mu[3]*zeta/cg[3];
           i = unscale(b,bhat,K0);
           i = output(Y,h,b,phi,PHI,TEST,SN,CN,DN);
           Y = Y + step_size;
           zeta  = Y - Ys;
        }
        Yp = Yn;
        b0hat[1] = bhat[1];
        b0hat[2] = bhat[2];
        b0hat[3] = bhat[3];
        phi0[1] = phi[1];
        phi0[2] = phi[2];
        phi0[3] = phi[3];
        PHI0 = PHI;
    }
fclose(in);
fclose(out);
}

error(int i)
{
printf("-ERROR- %3d \n",i);
exit(1);
}
```

```
/*************************************************************/
/* Name: init                                                */
/*                                                           */
/* Purpose: This function reads the input file and           */
/*          initializes the parameters and initial           */
/*          conditions.                                      */
/*                                                           */
/* Written: Feb 9, 1992                                      */
/* By: Francois Primeau                                      */
/*                                                           */
/*************************************************************/
#include <stdio.h>
#include <math.h>

init(FILE *in,
     double b0hat[4],
     double w[4],
     double K0[4],
     double *mu0,
     double mu[4],
     double cg[4],
     double *b,
     double *H1,
     double *H2,
     double k[4],
     double omega[4],
     double z[4]
     )
{
   int i;
   double temp;
   double N[4],c[4];

   /*read in the first wave number*/
   fscanf(in,"%le ",&k[1]);

   /*read in the parameters for the exponential
     cross shelf profile*/
   fscanf(in,"%le %le ",b,H2);
   *H1 = *H2/exp(2*(*b));

   /*compute the base triad*/
   i = triad(k,omega,z,*b);

   swap(&k[2],&k[3]);
   swap(&k[1],&k[2]);
   swap(&omega[2],&omega[3]);
```

```
    swap(&omega[1],&omega[2]);
    swap(&z[2],&z[3]);
    swap(&z[1],&z[2]);


    /*read in the initial wave amplitude magnitudes*/
    fscanf(in,"%le %le %le",&b0hat[1],&b0hat[2],&b0hat[3]);

    /*read in the initial wave amplitude phases*/
    fscanf(in,"%le %le %le",&w[1],&w[2],&w[3]);

    /*calculate the interaction coefficients*/
    i = K0j(K0,*b,*H2,k,omega,z);
    \
    /*calculate alongshore topography coefficient*/

    i = Nj(N,*b,*H2,z);
    i = cj(c,k,omega);
    i = cgj(cg,c,*H2,N,k,z);
    i = mu0j(mu,*b,*H2,N,k,omega,z);

    *mu0 = mu[1]/cg[1] + mu[2]/cg[2] + mu[3]/cg[3];


    return(0);
}
```

```
/*********************************************************/
/* Name: scale                                           */
/*                                                       */
/* Purpose: This function scales the wave amplitudes.    */
/*                                                       */
/* Written: Feb 19, 1992                                 */
/* By: Francois Primeau                                  */
/*                                                       */
/*********************************************************/
#include <stdio.h>
#include <math.h>

int scale(double bhat[4],
          const double b[4],
          const double K0[4]
         )
{
   bhat[1] = b[1]*sqrt(_ABS(K0[2]*K0[3]));
   bhat[2] = b[2]*sqrt(_ABS(K0[3]*K0[1]));
   bhat[3] = b[3]*sqrt(_ABS(K0[1]*K0[2]));

   return(0);
}
```

```
/*****************************************************************/
/* Name: astopo                                                  */
/*                                                               */
/* Purpose: This function reads in the along shore topography*/
/*          data.                                                */
/*                                                               */
/* Written: Feb 19, 1992                                         */
/* By: François Primeau                                          */
/*                                                               */
/*****************************************************************/
#include <stdio.h>
#include <math.h>
int astopo(FILE *in,
           double *Yn,
           double *Ys,
           double *zetamot,
           double *H,
           double *step
          )
{
  if (fscanf(in,"%le %le %le %le %le ",Yn,Ys,zetamot,H,step)
      != 5)
      return(1);
  return(0);
}
```

```
/*****************************************************************/
/* Name: cj                                                      */
/*                                                               */
/* Purpose: This function calculates the phase speeds.           */
/*          The function returns the value 0 in all cases.*/
/*                                                               */
/* Written: Feb 9, 1992                                          */
/* By: Francois Primeau                                          */
/*                                                               */
/*****************************************************************/
#include <stdio.h>
#include <math.h>
int cj( double c[4],            /*phase speeds */
        const double k[4],      /*wave numbers*/
        const double omega[4] /*frequencies*/
      )
{
   c[1] = omega[1]/k[1];
   c[2] = omega[2]/k[2];
   c[3] = omega[3]/k[3];

   printf("c1 %e \n",c[1]);
   printf("c2 %e \n",c[2]);
   printf("c3 %e \n",c[3]);

   return(0);
}
```

```
/*****************************************************************/
/* Name: zj                                                      */
/*                                                               */
/* Purpose: This function calculates the roots of the            */
/*          transcendental equation                              */
/*                        tan(zj) = - zj/(b + |kj|).             */
/*          zj>0, given the dispersion relation.                 */
/*          The function returns the value 0 in all cases.      */
/*                                                               */
/* Written: Feb 9, 1992                                          */
/* By: Francois Primeau                                          */
/*                                                               */
/*****************************************************************/
#include <stdio.h>
#include <math.h>
int zj( double z[4],          /*roots*/
        const double b,    /*cross shelf profile parameter*/
        const double k[4],   /*wave numbers */
        const double omega[4]/*frequency*/
      )
{
   z[1] = sqrt(-2*b*k[1]/omega[1]-k[1]*k[1]-b*b);/*disp. rel.
                                                   for wave 1*/
   z[2] = sqrt(-2*b*k[2]/omega[2]-k[2]*k[2]-b*b);/*disp. rel.
                                                   for wave 2*/
   z[3] = sqrt(-2*b*k[3]/omega[3]-k[3]*k[3]-b*b);/*disp. rel.
                                                   for wave 3*/

   printf("z1 = %e \n",z[1]);
   printf("z2 = %e \n",z[2]);
   printf("z3 = %e \n",z[3]);

   return(0);
}
```

```
/*****************************************************************/
/* Name: Nj                                                      */
/*                                                               */
/* Purpose: This function calculates the normalization           */
/*          constants as given by Hsieh and Mysak (JPO 1980).    */
/*          The function returns the value 1 in all cases.       */
/*                                                               */
/* Written: Feb 9, 1992                                          */
/* By: Francois Primeau                                          */
/*                                                               */
/*****************************************************************/
#include <stdio.h>
#include <math.h>
int Nj( double N[4],          /*Normalization constants*/
        const double b,       /*cross shelf profile parameter*/
        const double H2,      /*cross shelf profile parameter*/
        const double z[4]     /*roots*/
      )
{

   N[1] = sqrt(2*z[1]*H2/(b*(2*z[1]-sin(2*z[1]))));
   N[2] = sqrt(2*z[2]*H2/(b*(2*z[2]-sin(2*z[2]))));
   N[3] = sqrt(2*z[3]*H2/(b*(2*z[3]-sin(2*z[3]))));

   printf("N1 %e \n",N[1]);
   printf("N2 %e \n",N[2]);
   printf("N3 %e \n",N[3]);

   return(0);
}
```

```
/*****************************************************************/
/* Name: cgj                                                     */
/*                                                               */
/* Purpose: This function calculates the groups velocities       */
/*          using the formulas given by Hsieh and Mysak          */
/*          (JPO 1980).                                          */
/*                                                               */
/* Written: Feb 9, 1992                                          */
/* By: Francois Primeau                                          */
/*                                                               */
/*****************************************************************/
#include <stdio.h>
#include <math.h>
int cgj( double cg[4],        /*group velocites*/
         const double c[4],   /*phase speeds*/
         const double H2,     /*cross shelf profile parameter*/
         const double N[4],   /*normalization constants*/
         const double k[4],   /*wave numbers*/
         const double z[4]    /*roots*/
       )
{
   double gamma[4];

   gamma[1] = (N[1]*N[1]/H2)*(0.5-sin(2*z[1])/(4*z[1]) +
                              sin(z[1])*sin(z[1])/(2*_ABS(k[1])));
   gamma[2] = (N[2]*N[2]/H2)*(0.5-sin(2*z[2])/(4*z[2]) +
                              sin(z[2])*sin(z[2])/(2*_ABS(k[2])));
   gamma[3] = (N[3]*N[3]/H2)*(0.5-sin(2*z[3])/(4*z[3]) +
                              sin(z[3])*sin(z[3])/(2*_ABS(k[3])));

   cg[1] = c[1]*(1+2*gamma[1]*c[1]*k[1]*k[1]);
   cg[2] = c[2]*(1+2*gamma[2]*c[2]*k[2]*k[2]);
   cg[3] = c[3]*(1+2*gamma[3]*c[3]*k[3]*k[3]);

   printf("cg1 %e \n",cg[1]);
   printf("cg2 %e \n",cg[2]);
   printf("cg3 %e \n",cg[3]);

   return(0);
}
```

```
/*****************************************************************/
/* Name: mu0j                                                    */
/*                                                               */
/* Purpose: This function calculates the alongshore              */
/*          topogaraphy coefficients.                            */
/*          coeffs.                                              */
/*                                                               */
/* Written: Feb 9, 1992                                          */
/* By: Francois Primeau                                          */
/*                                                               */
/*****************************************************************/
#include <stdio.h>
#include <math.h>
int mu0j( double mu[4],     /*alongshore topography coeffs*/
          const double b,  /*cross shelf profile parameter*/
          const double H2, /*cross shelf profile parameter*/
          const double N[4],/*normalization constants*/
          const double k[4],/*wave numbers*/
          const double omega[4],/*frequencies*/
          const double z[4]     /*roots*/
        )
{
   int j;

   for(j = 1; j<= 3;j++)
      mu[j] = (N[j]*N[j]*omega[j]/(4*H2*H2))*(
              (b*b-k[j]*k[j]-z[j]*z[j])/b +
              exp(2*b)*(-b*b+k[j]*k[j]-z[j]*z[j])/b +
              ( b*exp(2*b)*(b*b-k[j]*k[j]-3*z[j]*z[j]) +
               b*(-b*b+k[j]*k[j]+3*z[j]*z[j])*cos(2*z[j]) +
               z[j]*(3*b*b-k[j]*k[j]-z[j]*z[j])*sin(2*z[j])
              )/(b*b+z[j]*z[j]));

   return(0);
}
```

```
/*************************************************************/
/* Name: muj                                                 */
/*                                                           */
/* Purpose: This function calculates the alongshore topog    */
/*          topography coefficient.                          */
/*                                                           */
/* Written: Feb 9, 1992                                      */
/* By: Francois Primeau                                      */
/*                                                           */
/*************************************************************/
#include <stdio.h>
#include <math.h>
int muj( double *mu0,        /*alongshore topography coeff*/
         const double b,     /*cross shelf profile parameter*/
         const double H2     /*cross shelf profile parameter*/
         const double k[4], /*wave numbers*/
         const double omega[4],/*frequencies*/
         const double z[4]  /*roots*/
       )
{
  int i;
  double c[4];  /*phase velocities*/
  double cg[4]; /*group velocities*/
  double N[4];  /*Normalization constants*/
  double mu[4]; /*alongshore topography coefficients*/

  i = Nj(N,b,H2,z);
  i = cj(c,k,omega);
  i = cgj(cg,c,H2,N,k,z);
  i = mu0j(mu,b,H2,N,k,omega,z);

  *mu0 = mu[1]/cg[1] + mu[2]/cg[2] + mu[3]/cg[3];

  return(0);
}
```

```
/*************************************************************/
/* Name: K0j                                                 */
/*                                                           */
/* Purpose: This function calculates the interaction         */
/*          coefficients for the three wave interaction      */
/*          equations, using the formulas given by           */
/*          Hsieh and Mysak (JPO 1980). The function returns*/
/*          the value 0 in  all cases.                       */
/*                                                           */
/* Written: Feb 9, 1992                                      */
/* By: Francois Primeau                                      */
/*                                                           */
/*************************************************************/
#include <stdio.h>
#include <math.h>
/*interaction coefficients*/
static double K[4][4][4] = {
          {{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0}},
          {{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0}},
          {{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0}},
          {{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0}}};
int K0j( double K0[4],      /*interaction coefficient*/
         const double b,    /*cross shelf profile parameter*/
         const double H2,   /*cross shelf profile parameter*/
         const double k[4], /*wave numbers*/
         const double omega[4],/*frequencies*/
         const double z[4]     /*roots*/
       )
{
  int i;
  double Ks[4]; /*Kj as defined by Hsieh and Mysak*/
  double c[4];  /*phase velocities*/
  double cg[4]; /*group velocities*/
  double N[4];  /*Normalization constants*/

  i = Nj(N,b,H2,z);
  i = cj(c,k,omega);
  i = cgj(cg,c,H2,N,k,z);
  i = Kjlm(K,b,H2,N,k,z);

  Ks[1] = c[1]*(K[1][2][3] + K[1][3][2]);
  Ks[2] = c[2]*(K[2][3][1] + K[2][1][3]);
  Ks[3] = c[3]*(K[3][1][2] + K[3][2][1]);

  K0[1] = Ks[1]/cg[1];
  K0[2] = Ks[2]/cg[2];
  K0[3] = Ks[3]/cg[3];
```

```
  printf("K1 = %e \n",Ks[1]);
  printf("K2 = %e \n",Ks[2]);
  printf("K3 = %e \n",Ks[3]);

  printf("K01 = %e \n",K0[1]);
  printf("K02 = %e \n",K0[2]);
  printf("K03 = %e \n",K0[3]);

  return(0);
}
```

```
/*************************************************************/
/* Name: Kjlm                                                */
/*                                                           */
/* Purpose: This function calculates the Kjlm coefficients   */
/*          using the formulas given by Hsieh and Mysak      */
/*          (JPO 1980). The function returns the value 0 in  */
/*           all cases.                                      */
/*                                                           */
/* Written: Feb 9, 1992                                      */
/* By: Francois Primeau                                      */
/*                                                           */
/*************************************************************/
#include <stdio.h>
#include <math.h>
const int permute[6][3] = {{1,2,3},{1,3,2},{2,3,1},{2,1,3},
                           {3,1,2},{3,2,1}};
int Kjlm( double K[4][4][4],   /*interaction coefficients*/
          const double b,   /*cross shelf profile prmtr*/
          const double H2,  /*cross shelf profile prmtr*/
          const double N[4],/*normalization constants*/
          const double k[4],/*wave numbers*/
          const double z[4] /*roots*/
        )
{
   double b1,b2,b3,b4,A1,A2,A3,A4,B1,B2,B3,
          B4,C1,C2,C3,C4,I1,I2,I3;
   int i,j,l,m;

   for ( i = 0; i<6; i++) {
      j = permute[i][0];
      l = permute[i][1];
      m = permute[i][2];

      b1 = z[j]-z[l]-z[m];
      b2 = z[j]-z[l]+z[m];
      b3 = z[j]+z[l]-z[m];
      b4 = z[j]+z[l]+z[m];

      A1 = sin(b1)/(b*b+b1*b1);
      A2 = sin(b2)/(b*b+b2*b2);
      A3 = sin(b3)/(b*b+b3*b3);
      A4 = sin(b4)/(b*b+b4*b4);

      B1 = cos(b1)/(b*b+b1*b1);
      B2 = cos(b2)/(b*b+b2*b2);
      B3 = cos(b3)/(b*b+b3*b3);
      B4 = cos(b4)/(b*b+b4*b4);
```

```
        C1 = exp(b)/(b*b+b1*b1);
        C2 = exp(b)/(b*b+b2*b2);
        C3 = exp(b)/(b*b+b3*b3);
        C4 = exp(b)/(b*b+b4*b4);

        I1 = 0.25*((b*A1+b1*B1-b1*C1)-(b*A2+b2*B2-b2*C2)
                  -(b*A3+b3*B3-b3*C3)+(b*A4+b4*B4-b4*C4));
        I2 = 0.25*((b1*A1-b*B1+b*C1)-(b2*A2-b*B2+b*C2)
                  +(b3*A3-b*B3+b*C3)-(b4*A4-b*B4+b*C4));
        I3 = 0.25*(-(b1*A1-b*B1+b*C1)+(b2*A2-b*B2+b*C2)
                  +(b3*A3-b*B3+b*C3)-(b4*A4-b*B4+b*C4));

        K[j][l][m] = N[j]*N[l]*N[m]*(k[m]/(H2*H2))
          *(
            b*(-3*b*b-z[j]*z[j]-2*z[l]*z[l]-
            3*k[l]*k[l]-k[m]*k[m])*I1
            +z[l]*(-b*b-z[j]*z[j]+k[l]*k[l]-k[m]*k[m])*I2
            +2*z[j]*(-b*b-z[l]*z[l])*I3
            +sin(z[j])*sin(z[l])*sin(z[m])/
            ( ABS(k[j])+ ABS(k[l])+ ABS(k[m]))
            *(k[l]*k[l]*(-2* ABS(k[j])- ABS(k[l]))+
             ABS(k[l])*(k[m]*k[m]-k[j]*k[j]))
           );
        printf("K%d%d%d = %e \n",j,l,m,K[j][l][m]);
    }
    return(0);
}
```

```
/*************************************************************/
/* Name: invsn                                               */
/*                                                           */
/* Purpose: This function is the inverse Jacobi sn(*|*)      */
/*          elliptic function.                               */
/*                                                           */
/* Written: Feb 9, 1992                                      */
/* By: Francois Primeau                                      */
/*                                                           */
/*************************************************************/
#include <stdio.h>
#include <math.h>
int invsn(double *F,
          const double u,
          const double k
                    )
{
  int ierr;
  double l,phi,psi;
  double EE,FF;
  if (u > 1 ) return(4);
  phi = asin(u);
  psi = (M_PI/2) - phi;
  if (k > 1 ) return(1);
  l = sqrt(1 - k*k);
  dellpi_(&phi,&psi,&k,&l,&FF,&EE,&ierr);
  *F = FF;
  printf("%.14e\n",*F);
  return(ierr);
}
```

```
    i = sort(y);
    if (y[2] > 0) {
      printf("WARNING: %le   yhat[2] set to zero.\n",
             y[2]); y[2] = 0;}
    return(0);
}
```

```
/*****************************************************************/
/* Name: cubic                                                   */
/*                                                               */
/* Purpose: This function calculates the roots of the cubic      */
/*          polynomial.                                          */
/*                                                               */
/* Written: Feb 9, 1992                                          */
/* By: Francois Primeau                                          */
/*                                                               */
/*****************************************************************/
#include <stdio.h>
#include <math.h>
extern struct global;
int cubic(double y[4], /*roots*/
          const double b10,/*initial condition for wave1*/
          const double b20,/*initial condition for wave2*/
          const double b30,/*initial condition for wave3*/
          const double Gamma,/*constant of the motion*/
          const double mu0,/*alongshore
                             topographic coefficient*/
          const double H /*height of the top hat topographic
                           configuration*/
       )
{
   double d1,d2,d3; /*coefficients of the cubic polynomial*/
   double Q,R,D,M,theta;
   int i;

   /*initialize the coefficient of y to the power 2*/
   d1 = -b10*b10 + b20*b20 + b30*b30 + mu0*mu0*H*H/4;

   /*initialize the coefficient of y to the power 1*/
   d2 = -b10*b10*b20*b20 - b10*b10*b30*b30 + b20*b20*b30*b30 +
        Gamma*mu0*H - b10*b10*mu0*mu0*H*H/2;
   /*initialize the coefficient of y to the power 0*/
   d3 = -b10*b10*b20*b20*b30*b30 + Gamma*Gamma -
         b10*b10*Gamma*H*mu0 +
         b10*b10*b10*b10*mu0*mu0*H*H/4;

   Q = (3.0*d2-d1*d1)/9.0;
   R = (9.0*d1*d2-27*d3-2.0*d1*d1*d1)/54.0;
   D = Q*Q*Q+R*R ;
   if ( D>=0) return(1); /*roots are not all real*/
   theta = acos(R/sqrt(-Q*Q*Q));
   y[1] = 2*sqrt(-Q)*cos(theta/3) - d1/3;
   y[2] = 2*sqrt(-Q)*cos(theta/3 + 2*M_PI/3) - d1/3;
   y[3] = 2*sqrt(-Q)*cos(theta/3 + 4*M_PI/3) - d1/3;
```

```
/*****************************************************************/
/* Name: sort                                                    */
/*                                                               */
/* Purpose: This function sorts three numbers in descending      */
/*          order.                                               */
/*                                                               */
/* Written: Feb 9, 1992                                          */
/* By: Francois Primeau                                          */
/*                                                               */
/*****************************************************************/
#include <stdio.h>
#include <math.h>
int sort(double y[4]
        )
{
   if (y[2] > y[1]) swap(&y[1],&y[2]);
   if (y[3] > y[2]) swap(&y[2],&y[3]);
   if (y[2] > y[1]) swap(&y[1],&y[2]);
   return(0);
}

int swap(double *a,
         double *b)
{
double temp;
temp = *a;
*a = *b;
*b = temp;
return(0);
}
```

```
/*****************************************************************/
/* Name: solve                                                   */
/*                                                               */
/* Purpose: This function computes the amplitudes and the        */
/*          phases for the   resonant triad of waves.            */
/*                                                               */
/* Written: Feb 19, 1992                                         */
/* By: Francois Primeau                                          */
/*                                                               */
/*****************************************************************/
#include <stdio.h>
#include <math.h>
static struct global { double root[4];
                       double b0hat[4];
                       double khat;
                       double s[4];
                       double S;
                       double thetahat;
                       double Gamma;
                       double mu0;
                       double H;
                       double PHI;
                       double SN;
                       double CN;
                       double DN;
                     } common;

int solve( double bhat[4],
           double phi[4],
           double s[4],
           const double S,
           double *PHI,
           double *TEST,
           const double root[4],
           const double b0hat[4],
           const double phi0[4],
           const double PHI0,
           const double Gamma,
           const double mu0,
           const double H,
           const double khat,
           const double thetahat,
           const double zeta,
           const double zetanot,
           double *SN,
           double *CN,
           double *DN
```

```
        )
{
  extern double f1();
  extern double f2();
  extern double f3();
  extern double f4();
  double aerr = 10e-12;
  double rerr = 10e-12;
  double z;
  double err;
  int num;
  int ierr;
  int l = 1000;
  int m = 4000;
  int n;
  int iwk[1000];
  double wk[4000];
  double aa,bb;
  double ee;
  double er;
  {
    int i;
    for (i=1; i<4; i++)
    {
       common.root[i] = root[i];
       common.b0hat[i] = b0hat[i];
       common.s[i] = s[i];
    }
    common.khat = khat;
    common.thetahat = thetahat;
    common.Gamma = Gamma;
    common.mu0 = mu0;
    common.H = H;
    common.S = S;
  }
  aa = zetanot;
  bb = zeta;

  dqags_(f1,&aa,&bb,&aerr,&rerr,&z,&err,&num,
        &ierr,&l,&m,&n,iwk,wk);
  if (ierr !=0 ) printf("ierr1: %d %.14e \n",ierr,bb);
  phi[1] = phi0[1] + z;

  dqags_(f2,&aa,&bb,&aerr,&rerr,&z,&err,&num,
        &ierr,&l,&m,&n,iwk,wk);
  if (ierr != 0) printf("ierr2: %d bb: %.14e  \n",ierr,bb);
  phi[2] = phi0[2] + z;
```

```
    dqags_(f3,&aa,&bb,&aerr,&rerr,&z,&err,&num,
           &ierr,&l,&m,&n,iwk,wk);
    if (ierr !=0) printf("ierr3: %d bb: %.14e \n",ierr,bb);
    phi[3] = phi0[3] + z;
    dqags_(f4,&aa,&bb,&aerr,&rerr,&z,&err,&num,
           &ierr,&l,&m,&n,iwk,wk);
    if (ierr != 0) printf("ierr4: %d bb: %.14e \n",ierr,bb);
    *PHI = PHI0 - common.mu0*common.H*(zeta-zetanot) + z;


    if ( b(bhat,zeta,zetanot) == 1 ) return(1);
    *TEST= acos((Gamma-0.5*mu0*H*bhat[1]*bhat[1])/
    (bhat[1]*bhat[2]*bhat[3]));
    *SN = common.SN;
    *CN = common.CN;
    *DN = common.DN;
    s[1] = common.s[1];
    s[2] = common.s[2];
    s[3] = common.s[3];
    return(0);
}
double f1(double *Y)
{ double bhat[4];
  double value;
  int i;
  if ( (i=b(bhat,*Y,0)) != 0 ) error(100+i);
  return(
    -(2*common.Gamma-common.mu0*common.H*bhat[1]*bhat[1])/
     (2*bhat[1]*bhat[1]));
}
double f2(double *Y)
{ double bhat[4];
  double value;
  int i;
  if ( (i=b(bhat,*Y,0)) !=0 ) error(200+i);
  return(
    (2*common.Gamma-common.mu0*common.H*bhat[1]*bhat[1])/
    (2*bhat[2]*bhat[2]));
}
double f3(double *Y)
{ double bhat[4];
  double value;
  int i;
  if ( (i=b(bhat,*Y,0)) != 0 ) error(300+i);
  return(
    (2*common.Gamma-common.mu0*common.H*bhat[1]*bhat[1])/
```

```
    (2*bhat[3]*bhat[3]));
}
double f4(double *Y)
{ double bhat[4];
  double value;
  int i;
  if ( (i=b(bhat,*Y,0)) != 0 ) error(400+i);
  return(
    (common.Gamma - common.mu0*common.H*bhat[1]*bhat[1]*0.5)*
    (-1/(bhat[1]*bhat[1]) +
        1/(bhat[2]*bhat[2]) + 1/(bhat[3]*bhat[3])));
}
int b( double bhat[4],
       const double zeta,
       const double zetanot
     )
{
  double yhat;
  double sn,cn,dn,s,c,d;
  double temp;
  double u,u1,u2,u3;
  double k;
  double l;
  double psi,cpsi;
  double FF,EE;
  double flip1 =1;
  double flip2 =1;
  double flip3 =1;
  int ierr;

  u = common.S*sqrt(common.root[1]-common.root[3])*
      (zeta-zetanot)+common.thetahat;
  k = common.khat;

  if (k > 1) return(1);
  l = sqrt(1 - k*k);
  psi = M_PI/2;
  cpsi = 0.0;
  dellpi_(&psi,&cpsi,&k,&l,&FF,&EE,&ierr);
  while ( u > 4*FF) u = u - 4*FF;
  while ( u < 0.0)u = u + 4*FF;
  u1 = u;
  flip1 =1;
  if (u1 > 2*FF ){
     u1 = u1 -2*FF;
     flip1 = -1*flip1;
  }
```

```
if ( u1 > FF ) u1 = -1*u1 + 2*FF;
u2 = u;
flip2 = 1;
if (u2 > 2*FF) u2 = -u2 + 4*FF;
if (u2 > FF ) {
   u2 = -1*u2 + 2*FF;
   flip2 = -1*flip2;
}
u3 = u;
flip3 =1;
if ( u3 > 2*FF) u3 = u3 -2*FF;
if ( u3 > FF) u3 = -1*u3 + 2*FF;

djcbyf_(&u1,&k,&s,&c,&d,&ierr);
if (ierr != 0) return(ierr);
sn = flip1*s;
djcbyf_(&u2,&k,&s,&c,&d,&ierr);
if (ierr != 0) return(ierr);
cn = flip2*c;
djcbyf_(&u3,&k,&s,&c,&d,&ierr);
if (ierr != 0) return(ierr);
dn = flip3*d;
common.SN = sn;
common.CN = cn;
common.DN = dn;
if ( (_ABS(common.root[1] -
           common.b0hat[1]*common.b0hat[1]) < 1e-13 )
    & (sn < 0.0 )) common.s[1] = -1.0;
if ( (_ABS(common.root[1] -
           common.b0hat[1]*common.b0hat[1]) < 1e-13 )
    & (sn > 0.0 )) common.s[1] = 1.0;
if ( (_ABS(common.root[1] -
           common.b0hat[1]*common.b0hat[1]) < 1e-13 )
    & (cn < 0.0 )) common.s[3] = -1.0;
if ( (_ABS(common.root[1] -
           common.b0hat[1]*common.b0hat[1]) < 1e-13 )
    & (cn > 0.0 )) common.s[3] = 1.0;
yhat = (common.root[2]-common.root[1])*sn*sn+common.root[1];

bhat[1] = common.s[1]*
          sqrt(common.b0hat[1]*common.b0hat[1]-yhat);
bhat[2] = common.s[2]*
          sqrt(common.b0hat[2]*common.b0hat[2]+yhat);
bhat[3] = common.s[3]*
          sqrt(common.b0hat[3]*common.b0hat[3]+yhat);

return(0);
}
```

```
/*************************************************************/
/* Name: unscale                                             */
/*                                                           */
/* Purpose: This function unscales the wave amplitudes.      */
/*                                                           */
/* Written: Feb 19, 1992                                     */
/* By: Francois Primeau                                      */
/*                                                           */
/*************************************************************/
#include <stdio.h>
#include <math.h>

int unscale(double b[4],
            const double bhat[4],
            const double K0[4]
           )
{
   b[1] = bhat[1]/sqrt(_ABS(K0[2]*K0[3]));
   b[2] = bhat[2]/sqrt(_ABS(K0[3]*K0[1]));
   b[3] = bhat[3]/sqrt(_ABS(K0[1]*K0[2]));

   return(0);
}
```

```
/*********************************************************/
/* Name: unscale                                         */
/*                                                       */
/* Purpose: This function unscales the wave amplitudes.*/
/*                                                       */
/* Written: Feb 19, 1992                                 */
/* By: Francois Primeau                                  */
/*                                                       */
/*********************************************************/
#include <stdio.h>
#include <math.h>

int unscale(double b[4],
            const double bhat[4],
            const double K0[4]
           )
{
   b[1] = bhat[1]/sqrt(_ABS(K0[2]*K0[3]));
   b[2] = bhat[2]/sqrt(_ABS(K0[3]*K0[1]));
   b[3] = bhat[3]/sqrt(_ABS(K0[1]*K0[2]));

   return(0);
}
```

```
/*************************************************************/
/* Name: output                                              */
/*                                                           */
/* Purpose: This function ouputs the results and plots the   */
/*          solutions.                                       */
/*                                                           */
/* Written: Feb 9, 1992                                      */
/* By: Francois Primeau                                      */
/*                                                           */
/*************************************************************/
#include <stdio.h>
#include <math.h>
extern FILE *out;
int output(double YY,
           double h,
           double b[4],
           double phi[4],
           double PHI,
           double TEST,
           double SN,
           double CN,
           double DN
          )
{

fprintf(out,"%e %e %e %e %e %e %e %e %e %e\n",
             (float)YY,(float)h,(float)b[1],(float)b[2],
             (float)b[3],(float)phi[1],(float)phi[2],
             (float)phi[3],(float)PHI,(float)TEST);
return(0);
}
```