```
/*   floq.c
 *   do a Floquet calculation for a single spin 1/2
 *   with ZCW powder average over the whole sphere
 *   Use spherical tensors as much as possible
 *
 *   Basic references
 *    Herzfeld and Berger, J. Chem. Phys. 73, 6021-6030 (1980)
 *    Maricq and Waugh,  J. Chem. Phys. 70, 3300-3316 (1978)
 *    Vega, Olejniczak and Griffin, J. Chem. Phys. 80, 4832-4840 (1984)
 *   Useful modern review
 *    Hodgkinson and Emsley, Progress NMR Spect. 36, 201-239 (2000)
 *   Our Concepts paper
 *    Bain and Dumont, Concepts in Magnetic Resonance 13 159-170 (2001)
 *
 *   Numerical stuff is handled by a package of routines call Meschach,
 *   written in C, and available from Netlib (among other places)
 *
 *   Copyright (C) 2002 Alex D. Bain
 *   This program is free software; you can redistribute it and/or
 *   modify it under the terms of the GNU General Public License
 *   as published by the Free Software Foundation; either version 2
 *   of the License, or (at your option) any later version.
 *
 *   This program is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *   GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *   along with this program; if not, write to the Free Software
 *   Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
 */
#include <math.h>
#include <stdio.h>
#include "zmatrix2.h"  /* collection of meschach headers */

#define VERSION "0200315.1"

#define SQR(x) ((x) * (x))

#define PI                3.14159265358979323846
#define TOLER             0.000001
#define NFLOQUET         17
#define NGAMMA           16
#define MAX_FIBONACCI    35
#define N_FIBONACCI      16
#define NPOINTS          (8*1024)
#define START_FREQ       3000
#define DOMEG            (2*PI*2.0*START_FREQ/NPOINTS)
#define RELAX            20.0;
#define SCALE         10000.0
#define ANGLE_ERROR       0.0  /* magic = 0.955316618124509 radians*/
#define ANALYTICAL        0    /* analytical Fourier components */
#define FFT               1    /* numerical Fourier components */
#define H_AND_B           0    /* Herzfeld & Berger expressions */
#define NFOURIER          2    /* number of Fourier components for Floquet */

/* function prototypes */
void gen_freq(double alpha, double beta, int n_floquet, double theta,
               double sigma11, double sigma22, double sigma33,
               double omegar, double nuzero, VEC *freqre, VEC *freqim,
               MAT *flmat_re, MAT *flmat_im, MAT *eigvecr, MAT *eigveci, VEC *eigvals,
               ZVEC *rhozero, ZVEC *Uinv_rho);
void accum(double weight, double omegar, int n_floquet, double centreband,
               VEC *eigvals, VEC *spc, ZVEC *rhozero, ZVEC *Uinv_rho, VEC *sideb);
void mult_d2(double cbt, double sbt, ZVEC *in, ZVEC *out);
void mult_rz(double cal, double sal, ZVEC *in, ZVEC *out);
int makefibon(int *fibon);
void cg_(int nm, int n, double **ar, double **ai, double *wr, double *wi,
               int matz, double **zr, double **eigveci, double *fv1, double *fv2,
               double *fv3, int *ierr);
```

```
/*********************************************************************/
/********************** START OF MAIN PROGRAM ************************/
/*********************************************************************/
void main()

{
   int i, j, k;
   int n_floquet, fibon_order;

   double sigma11, sigma22, sigma33;
   double alpha, beta;
   double fac, f0, f1;
   double omegar, nuzero, weight, centreband, freq, theta;

   int fibon[MAX_FIBONACCI];

   /*
    *  these are all data structures for the meschach package of
    *  numerical subroutines
    */

   MAT *flmat_re, *flmat_im;
   MAT *eigvecr, *eigveci;
   VEC *eigvals;
   VEC *spc;
   VEC *sideb;
   VEC *freqre, *freqim;

   ZVEC *rhozero, *Uinv_rho;

   FILE *outp;   /* will contain the spectrum */

    /* set up the parameters */
    fibon_order = N_FIBONACCI;
    n_floquet = NFLOQUET;

    /* in Herzfeld & Berger, these give mu = 13.0, rho = 0.2 */
    sigma11 = -15.34;   sigma22 = -0.364;   sigma33 = 22.1;
    nuzero = 2*PI*50.0;
    omegar = 2*PI*144.0;

    centreband = nuzero * (sigma11 + sigma22 + sigma33)/3.0;

    theta = acos(1.0/sqrt(3.0)) + ANGLE_ERROR;   /* rotor angle */

    /*  allocate and initalize some memory */
    freqre = v_get(NGAMMA);
    freqim = v_get(NGAMMA);

    flmat_re = m_get(2*n_floquet+1, 2*n_floquet+1);
    flmat_im = m_get(2*n_floquet+1, 2*n_floquet+1);
    eigvecr = m_get(2*n_floquet+1, 2*n_floquet+1);
    eigveci = m_get(2*n_floquet+1, 2*n_floquet+1);
    eigvals = v_get(2*n_floquet+1);
    rhozero = zv_get(2*n_floquet+1);
    Uinv_rho = zv_get(2*n_floquet+1);
    sideb = v_get(2*n_floquet+1);
    v_zero(sideb);

    spc = v_get(NPOINTS);
    v_zero(spc);

    /* set up Fibonnacci numbers for ZCW average */
    makefibon(fibon);

    if ((outp=fopen("spectrum.txt","wt")) == NULL)
      {
        printf("cannot open file.\n");
        exit(0);
      }
```

```
    /*
     *  calculate the powder average over the whole sphere
     *  use Zaremba Conroy Wolfsberg grid - this code
     *  derived from the SIMPSON program
     *  Bak, Rasmussen and Neilsen, J. Mag. Res 147, 296-330 (2000)
     */
    printf("Calculating %d angles\n", fibon[fibon_order]-1);
    for (i=1; i<fibon[fibon_order]; i++)
      {
        fac = i/(double)fibon[fibon_order];
        f0 = fac * 1.0;
        f1 = fac * (double)fibon[fibon_order-2];

        f0 -= floor(f0);
        f1 -= floor(f1);
        alpha = 2 * PI * f0;   /* azimuthal */
        beta = PI * f1;        /* polar */

        weight = (1.0/(double)fibon[fibon_order]) * sin(beta);;
        gen_freq(alpha, beta, n_floquet, theta,
                        sigma11, sigma22, sigma33, omegar, nuzero, freqre, freqim,
                        flmat_re, flmat_im, eigvecr, eigveci, eigvals, rhozero, Uinv_rho);
        accum(weight, omegar, n_floquet, centreband, eigvals, spc, rhozero, Uinv_rho, sideb);
      }

    /* print out the spectrum */
    freq = START_FREQ;
    for (j=0; j<NPOINTS; j++)
       {
          fprintf(outp, "%10.2f %16.6f\n", freq, spc->ve[j]);
          freq -= DOMEG/(2*PI);
       }

    fclose(outp);

    for(i=0; i<2*n_floquet+1; i++)
      {
        printf("%4d %16.4f\n", (i-n_floquet), 100*sideb->ve[i]/sideb->ve[n_floquet]);
      }

    printf("Done\n");

}   /* end of main program */

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

void gen_freq(double alpha, double beta, int n_floquet, double theta,
                double sigma11, double sigma22, double sigma33,
                double omegar, double nuzero, VEC *freqre, VEC *freqim,
                MAT *flmat_re, MAT *flmat_im, MAT *eigvecr, MAT *eigveci, VEC *eigvals,
                ZVEC *rhozero, ZVEC *Uinv_rho)
/*
 *  set up and diagonalize the Floquet matrix
 *  return eigenvalues and eigenvectors, then to
 *  be used elsewhere for spectrum calculation
 */
{
   int i, j, m;
   int matz, ierr;

   double a1, b1, a2, b2;
   double sal, cal, sal2, cal2, sbt, cbt, sbt2, cbt2;
   double aa, bb;
   double coss, sins;
   double gamma, phi, sgam, cgam, x, y;
   double sigmaav;

   static VEC *wi = VNULL;
   static VEC *fv1 = VNULL, *fv2 = VNULL, *fv3 = VNULL;
```

```
static ZMAT *U = ZMNULL;
static ZMAT *Ud = ZMNULL;
static PERM *pivot = PNULL;

static ZVEC *sigma = ZVNULL;
static ZVEC *trsig = ZVNULL;
static ZVEC *trsig_r = ZVNULL;
static ZVEC *wksp = ZVNULL;
static ZVEC *Btr = ZVNULL;

     /* allocate temporary storage */
     sigma = zv_resize(sigma, 5);
     trsig = zv_resize(trsig, 5);
     trsig_r = zv_resize(trsig_r, 5);
     wksp = zv_resize(wksp, 5);

     Btr = zv_resize(Btr, 5);
     zv_zero(Btr);

     U = zm_resize(U, 2*n_floquet+1, 2*n_floquet+1);
     Ud = zm_resize(Ud, 2*n_floquet+1, 2*n_floquet+1);

     wi = v_resize(wi, 2*n_floquet+1);
     fv1 = v_resize(fv1, 2*n_floquet+1);
     fv2 = v_resize(fv2, 2*n_floquet+1);
     fv3 = v_resize(fv3, 2*n_floquet+1);

     /* express shift tensor as spherical tensor */
     zv_zero(sigma);
     /* zero order */
     sigmaav = (sigma11+sigma22+sigma33)/3.0;
     /* second order */
     /* use the notation 2+0 to remind us the z component runs from -2 to +2 */
     sigma->ve[2+0].re = sqrt(1.0/6.0) * (2*sigma33 - sigma11 - sigma22);
     sigma->ve[2+2].re = (1.0/2.0) * (sigma11 - sigma22);
     sigma->ve[2-2].re = (1.0/2.0) * (sigma11 - sigma22);

     /* work out the trigonometry for each vertex */
     cbt = cos(beta);                sbt = sin(beta);       /* polar */
     cal = cos(alpha);               sal = sin(alpha);      /* azimuthal angle */

     sal2 = sal*sal;                 cal2 = 1.0 - sal2;
     sbt2 = sbt*sbt;                 cbt2 = 1.0 - sbt2;

     /*
      *  Rotate the magnetic field to the rotor angle (magic?)
      *  since there is only Bz in the lab frame, we need only the
      *  one row of the Wigner matrix
      */
     Btr->ve[2+2].re = sqrt(3.0/8.0)*SQR(sin(theta));
     Btr->ve[2+1].re = sqrt(3.0/8.0)*(sin(2.0*theta));
     Btr->ve[2+0].re = 0.5 * (3.0*SQR(cos(theta))-1);
     Btr->ve[2-1].re = -sqrt(3.0/8.0)*(sin(2.0*theta));
     Btr->ve[2-2].re = sqrt(3.0/8.0)*SQR(sin(theta));

     /* rotate CSA tensor into rotor frame */
     /*  first from some arbitrary frame, to show we can do it */
     aa = 0.4;
     coss = cos(aa);    sins = sin(aa);
     mult_rz(coss, sins, sigma, wksp);
     bb = 1.1;
     coss = cos(bb);    sins = sin(bb);
     mult_rz(coss, sins, wksp, trsig);
     aa = 0.7;
     coss = cos(aa);    sins = sin(aa);
     mult_rz(coss, sins, trsig, sigma);

     /* now for real */
     mult_rz(cal, sal, sigma, wksp);   /* around z */
     mult_d2(cbt, sbt, wksp, trsig);   /* around y */
```

```
        /*
         *  Fourier components for Floquet
         *  These are the analytical results, since this is a simple system
         *  These are the terms that appear at the rotor frequency and
         *  double the rotor frequency. Since the rotation with the gamma
         *  angle corresponds to rotor rotation, these are just the
         *  +1 and +2 spherical tensor components
         */
        a1 = -trsig->ve[2+1].re;  /* minus sign from contraction (-1)^m */
        b1 = -trsig->ve[2+1].im;
        a2 = trsig->ve[2+2].re;
        b2 = trsig->ve[2+2].im;

 #if H_AND_B
        /*
         *  These terms direct from Herzfeld and Berger
         *  These are the sine and cosine terms - in Vega et al,
         *  they use magnitude and phase, but the answer is the same
         */
        a1 = sbt*cbt*(cal2*(sigma11-sigma33)+sal2*(sigma22-sigma33));
        b1 = - sal*cal*sbt*(sigma11-sigma22);
        a2 = 0.5*(cbt2*cal2-sal2)*(sigma11-sigma33) + 0.5*(cbt2*sal2-cal2)*(sigma22-sigma33);
        b2 = - sal*cal*cbt*(sigma11-sigma22);
#endif

#if FFT
        /*
         *  Now get Fourier components the hard way.
         *  explicitly do the gamma rotation and Fourier transform
         */

        phi = 0.0;
        for (i=0; i<NGAMMA; i++)
          {
            gamma = i * (2.0 * PI)/((double) NGAMMA);
            gamma += phi;
            cgam = cos(gamma);    sgam = sin(gamma);
            mult_rz(cgam, sgam, trsig, trsig_r);
            freqre->ve[i] = 0.0;
            freqim->ve[i] = 0.0;
            for(j=-2; j<3; j++)
              {
                x = Btr->ve[2-j].re * trsig_r->ve[2+j].re;
                y = Btr->ve[2-j].re * trsig_r->ve[2+j].im;
                if (j%2)  /* this is the (-1)^m factor */
                  {
                    x = -x;      y = -y;
                  }
                freqre->ve[i] += x;
                freqim->ve[i] += y;
              }
          }
        /*
         *  We have the frequency as a function of angle now,
         *  do the fft, in place
         */
        fft(freqre, freqim);

        /* normalize the FT */
        for (i=0; i<NGAMMA; i++)
          {
            freqre->ve[i] = freqre->ve[i]/((double) NGAMMA);
            freqim->ve[i] = freqim->ve[i]/((double) NGAMMA);
          }
#endif

        /*
         *  Make up the Floquet matrix
         *  zero the Floquet matrix. Keep real and imaginary part separate
         *  only because of the eigenvalue routine
         */
```

```
        m_zero(flmat_re);
        m_zero(flmat_im);

        /*
         *  Put in diagonal (time independent) elements in Floquet matrix
         */
        for (m=1; m<=n_floquet; m++)
          {
            flmat_re->me[n_floquet+m][n_floquet+m] = nuzero * sigmaav + m * omegar;
            flmat_re->me[n_floquet-m][n_floquet-m] = nuzero * sigmaav - m * omegar;
          }
        flmat_re->me[n_floquet][n_floquet] = nuzero * sigmaav;

        /* allow for non-magic angles */
        for (m=0; m<2*n_floquet+1; m++)
          {
            flmat_re->me[m][m] += nuzero * (sqrt(2.0/3.0)) * Btr->ve[2+0].re * trsig->ve[2+0].re;
          }

        /*
         *  Put in the time-dependent terms, either from analytical terms or from numerical
         *  Fourier analysis of explicit rotation about rotor axis
         */

#if ANALYTICAL
        /* first off-diagonals */
        /* sqrt(2.0)/3.0 comes from rotating static field into rotor frame */
        for (m=0; m<2*n_floquet; m++)
          {
            flmat_re->me[m][m+1] =  nuzero * (sqrt(2.0/3.0)) * Btr->ve[2+1].re * a1;
            flmat_im->me[m][m+1] = -nuzero * (sqrt(2.0/3.0)) * Btr->ve[2+1].re * b1;
            flmat_re->me[m+1][m] =  nuzero * (sqrt(2.0/3.0)) * Btr->ve[2+1].re * a1;
            flmat_im->me[m+1][m] =  nuzero * (sqrt(2.0/3.0)) * Btr->ve[2+1].re * b1;    /* c conj */
          }

        /* second off-diagonal */
        for (m=0; m<2*n_floquet-1; m++)
          {
            flmat_re->me[m][m+2] =  nuzero * (sqrt(2.0/3.0)) * Btr->ve[2+2].re * a2;
            flmat_im->me[m][m+2] = -nuzero * (sqrt(2.0/3.0)) * Btr->ve[2+2].re * b2;
            flmat_re->me[m+2][m] =  nuzero * (sqrt(2.0/3.0)) * Btr->ve[2+2].re * a2;
            flmat_im->me[m+2][m] =  nuzero * (sqrt(2.0/3.0)) * Btr->ve[2+2].re * b2;    /* c conj */
          }
#endif

#if FFT
        /* sqrt(2.0)/3.0 comes from rotating static field into rotor frame */
       for (i=0; i<=NFOURIER; i++)   /* loop over Fourier components */
         {
           for (m=0; m<2*n_floquet+1-i; m++)  /* fill in the Floquet matrix */
             {
               flmat_re->me[m][m+i] += nuzero * (sqrt(2.0/3.0)) * freqre->ve[i];
               flmat_im->me[m][m+i] += nuzero * (sqrt(2.0/3.0)) * freqim->ve[i];
               flmat_re->me[m+i][m] =  flmat_re->me[m][m+i];
               flmat_im->me[m+i][m] = -flmat_im->me[m][m+i];  /* complex conj */
             }
         }
#endif

        /*
         *  eispack routine for complex general matrix
         *  the matrix is Hermitian, so this is overkill. However, we
         *  will need it for exchange calculation.
         *  If the shift tensor is cylindrical, the matrix is real
         *  and a simple real eigenvalue routine will work
         */
        matz = 1;
        cg_(2*n_floquet+1, 2*n_floquet+1, flmat_re->me, flmat_im->me,
            eigvals->ve, wi->ve, matz, eigvecr->me, eigveci->me, fv1->ve, fv2->ve, fv3->ve, &ierr);

        /* to work out intensities, allow for fact that matrix may not be hermitian */
```

```
        for (i=0; i<2*n_floquet+1; i++)
          {
            for (j=0; j<2*n_floquet+1; j++)
              {
                U->me[i][j].re = eigvecr->me[i][j];
                U->me[i][j].im = eigveci->me[i][j];
              }
          }

        /*
         *  Calculate the spectrum as a sum of Lorentzian lines
         *  This code for calculating the spectrum is from MEXICO
         *  U is the matrix of eigenvectors of the Liouvillian
         *  You need (Uinv * rho(0)) times (U * detector operator)
         *  All the books say don't take inverses if you can avoid it
         *  Treat uinv as a set of linear equations and solve by LU
         *  decomposition.  This gives the lines in the spectrum in
         *  the eigenvector basis
         */
        zv_zero(rhozero);
        rhozero->ve[n_floquet].re = 1.0;   /* middle is only nozero element */
        pivot = px_resize(pivot, 2*n_floquet+1);

        zLUfactor(U, pivot);   /* decompose */
        zLUsolve(U, pivot, rhozero, Uinv_rho);  /* and solve */

        /* U * detector is trivial */
        for (i=0; i<2*n_floquet+1; i++)
          {
            rhozero->ve[i].re = eigvecr->me[n_floquet][i];
            rhozero->ve[i].im = eigveci->me[n_floquet][i];
          }

}    /* end of gen_freq */

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

void accum(double weight, double omegar, int n_floquet, double centreband,
           VEC *eigvals, VEC *buffer,
           ZVEC *rhozero, ZVEC *Uinv_rho, VEC *sideb)
/* calculate the spectrum for each orientation and accumulate */
{
   int i, j;
   double intenst, harmonic, freq;
   double numer, dnumer, denom, omegav;

   Complex cplxfreq, numerator;
   Complex ints;

   /*
    *  Calculate the intensities of each sideband, as well
    *  as the complete spectrum
    *  this is the square of the coefficient of the eigenvector
    */

   for (i=0; i<(2*n_floquet+1); i++)
     {
       /* accumulate the intensities of the sidebands */
       ints = zmlt(rhozero->ve[i], Uinv_rho->ve[i]);
       intenst = weight * ints.re;
       harmonic = (eigvals->ve[i]-centreband)/(omegar);
       for (j=0; j<2*n_floquet+1; j++)
         {
           if (fabs(harmonic + n_floquet - j) < 0.1)
             sideb->ve[j] += intenst;
         }

       /*
        *  And calculate the lineshape for display
        *  sweep over frequency for each complex
        *  Lorentzian line
```

```
        */
       freq = START_FREQ;

       cplxfreq.im = (2*PI*freq) - eigvals->ve[i];
       cplxfreq.re = RELAX;      /* arbitrary linewidth */
       numerator.re = SCALE*intenst;
       numerator.im = 0.0;
       /* set up for efficient calculation of real part of (numerator/cplxfreq) */
       numer = numerator.re * cplxfreq.re + numerator.im * cplxfreq.im;
       dnumer = numerator.im * DOMEG;
       denom = SQR(cplxfreq.re);
       omegav = cplxfreq.im;
       for (j=0; j<NPOINTS; j++)
         {
            buffer->ve[j] += (numer/(denom + SQR(omegav)));
            numer -= dnumer;
            omegav -= DOMEG;
         }
     }
}

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

void mult_d2(double cbt, double sbt, ZVEC *in, ZVEC *out)
/* apply the second-order Wigner matrix for rotation by angle beta around y */
{
   static ZMAT *d2 = ZMNULL;

        d2 = zm_resize(d2, 5, 5);     /* second order Wigner matrix */
        zm_zero(d2);

           /* define Wigner elements */
        d2->me[2+2][2+2].re = 0.25 * SQR(1+cbt);
        d2->me[2+2][2+1].re = -0.5 *(1+cbt) * sbt;
        d2->me[2+2][2+0].re = sqrt(3.0/8.0) * SQR(sbt);
        d2->me[2+2][2-1].re = -0.5 *(1-cbt) * sbt;
        d2->me[2+2][2-2].re = 0.25 * SQR(1-cbt);

        d2->me[2+1][2+2].re = -d2->me[2+2][2+1].re;
        d2->me[2+1][2+1].re = 0.5 * (cbt-1) + SQR(cbt);
        d2->me[2+1][2+0].re = -sqrt(3.0/2.0) * sbt * cbt;
        d2->me[2+1][2-1].re = 0.5 * (cbt+1) - SQR(cbt);
        d2->me[2+1][2-2].re = d2->me[2+2][2-1].re;

        d2->me[2+0][2+2].re = d2->me[2+2][2+0].re;
        d2->me[2+0][2+1].re = -d2->me[2+1][2+0].re;
        d2->me[2+0][2+0].re = 0.5 * (3 * SQR(cbt) - 1);
        d2->me[2+0][2-1].re = d2->me[2+1][2+0].re;
        d2->me[2+0][2-2].re = d2->me[2+2][2+0].re;

        d2->me[2-1][2+2].re = -d2->me[2+2][2-1].re;
        d2->me[2-1][2+1].re = d2->me[2+1][2-1].re;
        d2->me[2-1][2+0].re = -d2->me[2+1][2+0].re;
        d2->me[2-1][2-1].re = d2->me[2+1][2+1].re;
        d2->me[2-1][2-2].re = d2->me[2+2][2+1].re;

        d2->me[2-2][2+2].re = d2->me[2+2][2-2].re;
        d2->me[2-2][2+1].re = -d2->me[2+2][2-1].re;
        d2->me[2-2][2+0].re = d2->me[2+2][2+0].re;
        d2->me[2-2][2-1].re = -d2->me[2+2][2+1].re;
        d2->me[2-2][2-2].re = d2->me[2+2][2+2].re;

        zmv_mlt(d2, in, out);
}

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

void mult_rz(double cal, double sal, ZVEC *in, ZVEC *out)
/* apply the second-order Wigner matrix for rotation by angle alpha around z */
{
   static ZMAT *rz = ZMNULL;
```

```
        rz = zm_resize(rz, 5, 5);     /* rotation around z, for second order */
        zm_zero(rz);

        rz->me[2+2][2+2].re = 2 * SQR(cal) - 1;   /* cos(2 alpha) */
        rz->me[2+2][2+2].im = 2 * sal * cal;      /* sin(2 alpha) */

        rz->me[2+1][2+1].re = cal;
        rz->me[2+1][2+1].im = sal;

        rz->me[2+0][2+0].re = 1.0;

        rz->me[2-1][2-1].re = cal;
        rz->me[2-1][2-1].im = -sal;

        rz->me[2-2][2-2].re = 2 * SQR(cal) - 1;
        rz->me[2-2][2-2].im = -2 * sal * cal;

        zmv_mlt(rz, in, out);
}

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

int makefibon(int *fibon)
/* set up vector of Fibonacci numbers */
{
   int i;

   fibon[0] = 0;
   fibon[1] = 1;

   for (i=2; i<MAX_FIBONACCI; i++)
       fibon[i] = fibon[i-1] + fibon[i-2];
}

/*****************************************************************/
```