Read_dwnom_2.c

I added this nested sort:

```
int structnameidnocomparison(const void *s1, const void *s2)
{
        kpsorter *p1=(kpsorter *)s1;
        kpsorter *p2=(kpsorter *)s2;
/* The "->" is special to structures -- it is "aiming" at the data
 * element of the structure */
        int res;
        res = strcmp(p1->lastname, p2->lastname);
        if(res != 0)
                return res;
        else
                if(p1->icpsrid < p2->icpsrid)
                        return -1;
                else if(p1->icpsrid == p2->icpsrid)
                        return 0;
                else
                        return 1;
}
```

Read_dwnom_3.c

I added this nested sort:

```
int structnameidnocongresscomparison(const void *s1, const void *s2)
{
        kpsorter *p1=(kpsorter *)s1;
        kpsorter *p2=(kpsorter *)s2;
        int res;
        res = strcmp(p1->lastname, p2->lastname);
        if(res != 0) return res;
        else
                if(p1->icpsrid < p2->icpsrid)
                        return -1;
                else if(p1->icpsrid > p2->icpsrid)
                        return 1;
                 else
                        if(p1->congress < p2->congress)
                                return -1;
                        else if(p1->congress > p2->congress)
                                return 1;
                         else
                                return 0;

}
```

Read_dwnom_3A.c

Same as above but this produces a Compiler *Warning* but runs the same:

```
int structnameidnocongresscomparison(const void *s1, const void *s2)
{
        kpsorter *p1=(kpsorter *)s1;
        kpsorter *p2=(kpsorter *)s2;
        int res;
        res = strcmp(p1->lastname, p2->lastname);
        if(res != 0) return res;
        else {
                if(p1->icpsrid < p2->icpsrid)
                        return -1;
                if(p1->icpsrid > p2->icpsrid)
                        return 1;
                if(p1->icpsrid == p2->icpsrid)
                {
                        if(p1->congress < p2->congress)
                                return -1;
                        else if(p1->congress > p2->congress)
                                return 1;
                        if(p1->congress == p2->congress)
                                 return 0;
                }
        }
}
```

From Wikipedia, the free encyclopedia

Jump to: navigation, search

In mathematics, **Bessel functions**, first defined by the mathematician Daniel Bernoulli and generalized by Friedrich Bessel, are canonical solutions $y(x)$ of Bessel's differential equation:

$$x^2\frac{d^2y}{dx^2} + x\frac{dy}{dx} + (x^2 - \alpha^2)y = 0$$

for an arbitrary real or complex number α (the *order* of the Bessel function). The most common and important special case is where α is an integer *n*.

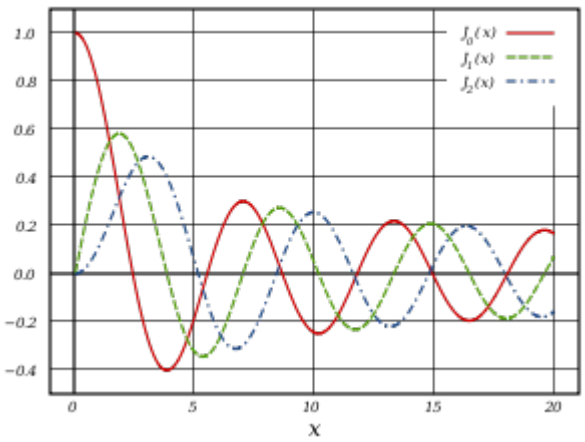
Although α and −α produce the same differential equation, it is conventional to define different Bessel functions for these two orders (e.g., so that the Bessel functions are mostly smooth functions of α). Bessel functions are also known as **cylinder functions** or **cylindrical harmonics** because they are found in the solution to Laplace's equation in cylindrical coordinates.

Bessel functions of the first kind: J_α

Bessel functions of the first kind, denoted as $J_\alpha(x)$, are solutions of Bessel's differential equation that are finite at the origin ($x = 0$) for non-negative integer α, and diverge as x approaches zero for negative non-integer α. The solution type (e.g. integer or non-integer) and normalization of $J_\alpha(x)$ are defined by its properties below. It is possible to define the function by its Taylor series expansion around $x = 0$:

$$J_\alpha(x) = \sum_{m=0}^{\infty} \frac{(-1)^m}{m!\Gamma(m+\alpha+1)}\left(\frac{x}{2}\right)^{2m+\alpha}$$

where $\Gamma(z)$ is the gamma function, a generalization of the factorial function to non-integer values. The graphs of Bessel functions look roughly like oscillating sine or cosine functions that decay proportionally to $1/\sqrt{x}$ (see also their asymptotic forms below), although their roots are not generally periodic, except asymptotically for large x. (The Taylor series indicates that $-J_1(x)$ is the derivative of $J_0(x)$, much like $-\sin(x)$ is the derivative of $\cos(x)$; more generally, the derivative of $J_n(x)$ can be expressed in terms of $J_{n\pm1}(x)$ by the identities below.)



Plot of Bessel function of the first kind, $J_\alpha(x)$, for integer orders α=0,1,2.

For non-integer α, the functions $J_\alpha(x)$ and $J_{-\alpha}(x)$ are linearly independent, and are therefore the two solutions of the differential equation. On the other hand, for integer order α, the following relationship is valid (note that the Gamma function becomes infinite for negative integer arguments):

$$J_{-n}(x) = (-1)^n J_n(x).$$

This means that the two solutions are no longer linearly independent. In this case, the second linearly independent solution is then found to be the Bessel function of the second kind, as discussed below.

Xamoeba_double.c

```
/* Driver for routine amoeba */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define NRANSI

#define MP 4
#define NP 3
//#define FTOL 1.0e-6
#define FTOL 0.0000000001
#define TINY 1.0e-10
#define NMAX 5000
#define NR_END 1
#define FREE_ARG char*

void amoeba(double **p, double y[], int ndim, double ftol,
            double (*funk)(double []), int *nfunk);
double func(double *);
double amotry(double **p, double y[], double psum[], int ndim,
             double (*funk)(double []), int ihi, double fac);
double bessj0(double x);
double *vector(long nl, long nh);
double **matrix(long nrl, long nrh, long ncl, long nch);
void free_vector(double *v, long nl, long nh);
void free_matrix(double **m, long nrl, long nrh, long ncl, long nch);

double func(double x[])
{
        return 0.8-bessj0((x[1]-0.535)*(x[1]-0.535)+(x[2]-0.646)*(x[2]-
0.646)+(x[3]-0.717)*(x[3]-0.717));
}

int main(void)
{
        int i,nfunc,j,ndim=NP;
        double *x,*y,**p;

        x=vector(1,NP);
        y=vector(1,MP);
        p=matrix(1,MP,1,NP);
        for (i=1;i<=MP;i++) {
                for (j=1;j<=NP;j++)
                        x[j]=p[i][j]=(i == (j+1) ? 1.0 : 0.0);
                y[i]=func(x);
        }
        amoeba(p,y,ndim,FTOL,func,&nfunc);
        printf("\nNumber of function evaluations: %3d\n",nfunc);
        printf("Vertices of final 3-d simplex and\n");
        printf("function values at the vertices:\n\n");
        printf("%3s %10s %12s %12s %14s\n\n",
               "i","x[i]","y[i]","z[i]","function");
        for (i=1;i<=MP;i++) {
```

```
                printf("%3d ",i);
                for (j=1;j<=NP;j++) printf("%12.6f ",p[i][j]);
                printf("%12.6f\n",y[i]);
        }
        printf("\nTrue minimum is at (0.5,0.6,0.7)\n");
        free_matrix(p,1,MP,1,NP);
        free_vector(y,1,MP);
        free_vector(x,1,NP);
        return 0;
}

void amoeba(double **p, double y[], int ndim, double ftol,
            double (*funk)(double []), int *nfunk)
{
        int i,ihi,ilo,inhi,j,mpts=ndim+1;
        double rtol,sum,swap,ysave,ytry,*psum;

        psum=vector(1,ndim);
        *nfunk=0;
                        for (j=1;j<=ndim;j++) {
                for (sum=0.0,i=1;i<=mpts;i++) sum += p[i][j];
                psum[j]=sum;}
                        for (;;) {
                ilo=1;
                ihi = y[1]>y[2] ? (inhi=2,1) : (inhi=1,2);
                for (i=1;i<=mpts;i++) {
                        if (y[i] <= y[ilo]) ilo=i;
                        if (y[i] > y[ihi]) {
                                inhi=ihi;
                                ihi=i;
                        } else if (y[i] > y[inhi] && i != ihi) inhi=i;
                }
                rtol=2.0*fabs(y[ihi]-y[ilo])/(fabs(y[ihi])+fabs(y[ilo])+TINY);
                if (rtol < ftol) {
                        swap= y[1];
                        y[1]=y[ilo];
                        y[ilo]=swap;
                        for (i=1;i<=ndim;i++)
                        {
                                swap=p[1][i];
                                p[1][i]=p[ilo][i];
                                p[ilo][i]=swap;
                        }
                                        break;
                }
                if (*nfunk >= NMAX)
                {
                        printf("NMAX exceeded");
                        exit(EXIT_FAILURE);
                }
                *nfunk += 2;
                ytry=amotry(p,y,psum,ndim,funk,ihi,-1.0);
                if (ytry <= y[ilo])
                        ytry=amotry(p,y,psum,ndim,funk,ihi,2.0);
                else if (ytry >= y[inhi]) {
```

```
                        ysave=y[ihi];
                        ytry=amotry(p,y,psum,ndim,funk,ihi,0.5);
                        if (ytry >= ysave) {
                                for (i=1;i<=mpts;i++) {
                                        if (i != ilo) {
                                                for (j=1;j<=ndim;j++)

        p[i][j]=psum[j]=0.5*(p[i][j]+p[ilo][j]);
                                                y[i]=(*funk)(psum);
                                        }
                                }
                                *nfunk += ndim;
                                                for (j=1;j<=ndim;j++) {
                                        for (sum=0.0,i=1;i<=mpts;i++) sum +=
p[i][j];
                                                psum[j]=sum;}
                        }
                } else --(*nfunk);
        }
        free_vector(psum,1,ndim);
}


double amotry(double **p, double y[], double psum[], int ndim,
             double (*funk)(double []), int ihi, double fac)
{
        int j;
        double fac1,fac2,ytry,*ptry;

        ptry=vector(1,ndim);
        fac1=(1.0-fac)/ndim;
        fac2=fac1-fac;
        for (j=1;j<=ndim;j++) ptry[j]=psum[j]*fac1-p[ihi][j]*fac2;
        ytry=(*funk)(ptry);
        if (ytry < y[ihi]) {
                y[ihi]=ytry;
                for (j=1;j<=ndim;j++) {
                        psum[j] += ptry[j]-p[ihi][j];
                        p[ihi][j]=ptry[j];
                }
        }
        free_vector(ptry,1,ndim);
        return ytry;
}

double bessj0(double x)
{
        double ax,z;
        double xx,y,ans,ans1,ans2;

        if ((ax=fabs(x)) < 8.0) {
                y=x*x;
                ans1=57568490574.0+y*(-13362590354.0+y*(651619640.7
                        +y*(-11214424.18+y*(77392.33017+y*(-184.9052456)))));
                ans2=57568490411.0+y*(1029532985.0+y*(9494680.718
```

```
                        +y*(59272.64853+y*(267.8532712+y*1.0))));
                ans=ans1/ans2;
        } else {
                z=8.0/ax;
                y=z*z;
                xx=ax-0.785398164;
                ans1=1.0+y*(-0.1098628627e-2+y*(0.2734510407e-4
                                                +y*(-0.2073370639e-
5+y*0.2093887211e-6)));
                ans2 = -0.1562499995e-1+y*(0.1430488765e-3
                                           +y*(-0.6911147651e-
5+y*(0.7621095161e-6
                        -y*0.934945152e-7)));
                ans=sqrt(0.636619772/ax)*(cos(xx)*ans1-z*sin(xx)*ans2);
        }
        return ans;
}
double *vector(long nl, long nh)
/* allocate a double vector with subscript range v[nl..nh] */
{
        double *v;

        v=(double *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(double)));
        if (!v)
        {
                printf("allocation failure in vector()");
                exit(EXIT_FAILURE);
        }
        return v-nl+NR_END;
}
double **matrix(long nrl, long nrh, long ncl, long nch)
/* allocate a double matrix with subscript range m[nrl..nrh][ncl..nch] */
{
        long i, nrow=nrh-nrl+1,ncol=nch-ncl+1;
        double **m;

        /* allocate pointers to rows */
        m=(double **) malloc((size_t)((nrow+NR_END)*sizeof(double*)));
        if (!m)
        {
                printf("allocation failure 1 in matrix()");
                exit(EXIT_FAILURE);
        }
        m += NR_END;
        m -= nrl;

        /* allocate rows and set pointers to them */
        m[nrl]=(double *) malloc((size_t)((nrow*ncol+NR_END)*sizeof(double)));
        if (!m[nrl])
        {
                printf("allocation failure 2 in matrix()");
                exit(EXIT_FAILURE);
        }
        m[nrl] += NR_END;
        m[nrl] -= ncl;
```

```
        for(i=nrl+1;i<=nrh;i++) m[i]=m[i-1]+ncol;

        /* return pointer to array of pointers to rows */
        return m;
}
void free_vector(double *v, long nl, long nh)
/* free a double vector allocated with vector() */
{
        free((FREE_ARG) (v+nl-NR_END));
}
void free_matrix(double **m, long nrl, long nrh, long ncl, long nch)
/* free a double matrix allocated by matrix() */
{
        free((FREE_ARG) (m[nrl]+ncl-NR_END));
        free((FREE_ARG) (m+nrl-NR_END));
}
```