## Xamoeba_general.c

```
/**/
/* xamoeba_general.c  -- C code is from NUMERICAL RECIPES IN C --
 *                       Additional code written by Keith Poole
 *                       September - October 2009
 *
   Uses Nedler-Mead downhill simplex method to find a minimum/maximum
   of a multi-dimensional function
                 |->amotry->func
Main -> amoeba --|
                 |->func

                 Amoeba initializes the simplex and calls func for each
                 point in the simplex.  The simplex starts at the usual
                 triangluar definition plus the origin -- ndim +1.

                 func is the user defined function being
                 minimized/maximized;

                 Amotry -- (from description in NUMERICAL RECIPES)
                 Extrapolates by a factor, fac, through the face of the
                 simplex across from a high point, tries, and replaces
                 the high point if the new point is better.
*/

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

#define MP 26
#define NP 25
#define nrowX 1000
#define ncolX 25
#define FTOL 0.0000001
#define TINY 1.0e-10
#define NMAX 100000

void amoeba(double **p, double y[], int ndim, double ftol, double Y[], double
X[],
            double (*funk)(double [],double[],double[]), int *nfunk);
double func(double *,double *,double *);
double amotry(double **p, double y[], double psum[], int ndim, double Y[],
double X[],
             double (*funk)(double [],double[],double[]), int ihi, double
fac);

double func(double x[],double Y[], double X[])
{
   int i, j;
   double sum=0;
   double sumsquared=0;
   for(i=0;i<nrowX;i++)
```

```
   {
   sum=0.0;
   for(j=0;j<ncolX;j++)
   {
          sum=sum+X[i+j*nrowX]*x[j+1];
   }
   sumsquared = sumsquared + (Y[i]-sum)*(Y[i]-sum);
   }
   printf("%lf\n",sumsquared);
return sumsquared;

}
FILE *fp;
FILE *jp;

int main(void)
{
       int i,nfunc,j,ndim=NP;
// **p is a pointer to p[0][0] -- p is a matrix
       double *x,*y,**p;
       double *X, *Y;

       X = (double *) malloc (nrowX*ncolX*sizeof(double));
       Y = (double *) malloc (nrowX*sizeof(double));
       x = (double *) malloc ((NP+1)*sizeof(double));
       y = (double *) malloc ((MP+1)*sizeof(double));
/* Dynamic Allocation of a Matrix: */
/* First, allocate pointers to rows */
/* Second, allocate rows and set pointers to them -- note that p[0] is
 * a pointer to p[0][0:n] -- that is a row -- so the syntax below is
 * p[0][0:nrow*ncol] */
/* Third, form the matrix from step two -- the pointer to the kth row,
 * p[k] is set equal to the memory
 * location of the number of nrows-1 times the number of columns plus
 * the memory location of p[0] */
       p= (double **) malloc((MP+1)*sizeof(double*));
       p[0]=(double *) malloc((NP+1)*(MP+1)*sizeof(double));
       for (i=0;i<MP+1;i++)p[i]=p[0]+i*(NP+1);
//
       jp = fopen("c:/docs_c_summer_course/data_AMOEBA.txt","w");
       if((fp = fopen("c:/docs_c_summer_course/data_ols.txt","r"))==NULL)
       {
              printf("\nUnable to open file OLS_DATA.TXT: %s\n",
strerror(errno));
              exit(EXIT_FAILURE);
       }
       else {

              fprintf(jp," Y and X = \n");
              for(i=0;i<nrowX;i++)
              {
                     fscanf(fp,"%lf",&Y[i]);
                     for(j=0;j<ncolX;j++)
                     {
                            fscanf(fp,"%lf",&X[i+j*nrowX]);
```

```
                        }
                        fprintf(jp,"%10d %12.6f", i,Y[i]);
                        for(j=0;j<ncolX;j++)
                        {
                                fprintf(jp,"%12.6f",X[i+j*nrowX]);
                        }
                        fprintf(jp,"\n");
                }
        }
        for (i=1;i<=MP;i++) {
                for (j=1;j<=NP;j++)
/* if i == (j+1) is true then the value of x[j] = 1.0, otherwise =0.0 */
                        x[j]=p[i][j]=(i == (j+1) ? 1.0 : 0.0);

                y[i]=func(x,Y,X);
        }
        amoeba(p,y,ndim,FTOL,Y,X,func,&nfunc);
        printf("\nNumber of function evaluations: %3d\n",nfunc);
        fprintf(jp,"\nNumber of function evaluations: %3d\n",nfunc);
        printf("Vertices of final 3-d simplex and\n");
        fprintf(jp,"Vertices of final 3-d simplex and\n");
        printf("function values at the vertices:\n\n");
        fprintf(jp,"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);
                fprintf(jp,"%3d ",i);
                for (j=1;j<=NP;j++) {
//                      printf("%12.6f ",p[i][j]);
                        fprintf(jp,"%12.6f ",p[i][j]);
                }
                printf("%12.6f\n",y[i]);
                fprintf(jp,"%12.6f\n",y[i]);
        }
//      printf("\nTrue minimum is at (0.5,0.6,0.7)\n");
        free(X);
        free(Y);
        free(x);
        free(y);
        free(p);
        return 0;
}

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

        psum = (double *) malloc ((ndim+1)*sizeof(double));
        *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,Y,X,funk,ihi,-1.0);
                if (ytry <= y[ilo])
                        ytry=amotry(p,y,psum,ndim,Y,X,funk,ihi,2.0);
                else if (ytry >= y[inhi]) {
                        ysave=y[ihi];
                        ytry=amotry(p,y,psum,ndim,Y,X,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,Y,X);
                                        }
                                }
                                *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(psum);
}
```

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

        ptry = (double *) malloc ((ndim+1)*sizeof(double));
        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,Y,X);
        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(ptry);
        return ytry;
}
```

## xamoeba_test.c

```
/**/
/* xamoeba_test.c  -- C code is from NUMERICAL RECIPES IN C --
 *                      Additional code written by Keith Poole
 *                      September - October 2009
 *
   Uses Nedler-Mead downhill simplex method to find a minimum/maximum
   of a multi-dimensional function
                |->amotry->func
Main -> amoeba --|
                |->func

                Amoeba initializes the simplex and calls func for each
                point in the simplex.  The simplex starts at the usual
                triangluar definition plus the origin -- ndim +1.

                func is the user defined function being
                minimized/maximized;

                Amotry -- (from description in NUMERICAL RECIPES)
                Extrapolates by a factor, fac, through the face of the
                simplex across from a high point, tries, and replaces
                the high point if the new point is better.
*/

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

#define MP 26
#define NP 25
#define nrowX 1000
#define ncolX 25
#define FTOL 0.0000001
#define TINY 1.0e-10
#define NMAX 100000
#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 keithrules(double x[]);

double func(double x[])
{
        return keithrules(x);
}
static double *Y,*X;
FILE *fp;
```

```
FILE *jp;

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

        X = (double *) malloc (nrowX*ncolX*sizeof(double));
        Y = (double *) malloc (nrowX*sizeof(double));
        x = (double *) malloc ((NP+1)*sizeof(double));
        y = (double *) malloc ((MP+1)*sizeof(double));
/* Dynamic Allocation of a Matrix: */
/* First, allocate pointers to rows */
/* Second, allocate rows and set pointers to them -- note that p[0] is
 * a pointer to p[0][0:n] -- that is a row -- so the syntax below is
 * p[0][0:nrow*ncol] */
/* Third, form the matrix from step two -- the pointer to the kth row,
 * p[k] is set equal to the memory
 * location of the number of nrows-1 times the number of columns plus
 * the memory location of p[0] */
        p= (double **) malloc((MP+1)*sizeof(double*));
        p[0]=(double *) malloc((NP+1)*(MP+1)*sizeof(double));
        for (i=0;i<MP+1;i++)p[i]=p[0]+i*(NP+1);
//
        jp = fopen("c:/docs_c_summer_course/data_AMOEBA.txt","w");
        if((fp = fopen("c:/docs_c_summer_course/data_ols.txt","r"))==NULL)
        {
                printf("\nUnable to open file OLS_DATA.TXT: %s\n",
strerror(errno));
                exit(EXIT_FAILURE);
        }
        else {

                fprintf(jp," Y and X = \n");
                for(i=0;i<nrowX;i++)
                {
                        fscanf(fp,"%lf",&Y[i]);
                        for(j=0;j<ncolX;j++)
                        {
                                fscanf(fp,"%lf",&X[i+j*nrowX]);
                        }
                        fprintf(jp,"%10d %12.6f", i,Y[i]);
                        for(j=0;j<ncolX;j++)
                        {
                                fprintf(jp,"%12.6f",X[i+j*nrowX]);
                        }
                        fprintf(jp,"\n");
                }
        }
/* if i == (j+1) is true then the value of x[j] = 1.0, otherwise =0.0 */
        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);
        fprintf(jp,"\nNumber of function evaluations: %3d\n",nfunc);
        printf("Vertices of final 3-d simplex and\n");
        fprintf(jp,"Vertices of final 3-d simplex and\n");
        printf("function values at the vertices:\n\n");
        fprintf(jp,"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);
                fprintf(jp,"%3d ",i);
                for (j=1;j<=NP;j++) {
                        fprintf(jp,"%12.6f ",p[i][j]);
                }
                printf("%12.6f\n",y[i]);
                fprintf(jp,"%12.6f\n",y[i]);
        }
        free(X);
        free(Y);
        free(x);
        free(y);
        free(p);
        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 = (double *) malloc ((ndim+1)*sizeof(double));
        *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(psum);
}


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 = (double *) malloc ((ndim+1)*sizeof(double));
        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(ptry);
        return ytry;
}

double keithrules(double x[])
{
        int i, j;
        double sum=0;
        double sumsquared=0;
        for(i=0;i<nrowX;i++)
        {
                sum=0.0;
                for(j=0;j<ncolX;j++)
                {
                        sum=sum+X[i+j*nrowX]*x[j+1];
                }
                sumsquared = sumsquared + (Y[i]-sum)*(Y[i]-sum);
        }
        printf("%lf\n",sumsquared);
        return sumsquared;

}
```

```
/*
xpowell_test.c -- C code is from NUMERICAL RECIPES IN C --
 *                       Additional code written by Keith Poole
 *                       September - October 2009
 *
   Uses Powell's Quadratically Convergent Method to find a minimum/maximum
   of a multi-dimensional function
                 |->func
Main -> powell --|          | ->mnbrak|
                 |->linmin->|         |
                            |->brent->f1dim->func

                 Powell minimizes a function of n variables.  The
                 starting values are in the vector p[1:n] and an n by n
                 matrix of directions -- normally a simplex.

                 func is the user defined function being
                 minimized/maximized;

                 linmin -- Finds the minimum on a line joining p and xi.

                    mnbrak -- Used by linmin.  It finds three points that bracket the
                 minimum.

                    brent -- Given the 3 points from mnbrak, it finds the
                    minimum
*/

#include <stdio.h>
#include <stddef.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#define NRANSI
#define ITMAX 2000
#define NDIM 25
#define nrowX 1000
#define ncolX 25
#define FTOL 1.0e-6
#define TOL 2.0e-4
#define GOLD 1.618034
#define GLIMIT 100.0
#define TINY 1.0e-20
#define SHFT(a,b,c,d) (a)=(b);(b)=(c);(c)=(d);
#define SIGN(a,b) ((b) >= 0.0 ? fabs(a) : -fabs(a))
static double maxarg1,maxarg2;
#define FMAX(a,b) (maxarg1=(a),maxarg2=(b),(maxarg1) > (maxarg2) ? (maxarg1) : (maxarg2))

#define CGOLD 0.3819660
#define ZEPS 1.0e-10


/* Driver for routine powell */

void powell(double p[], double **xi, int n, double ftol, int *iter, double *fret,
            double (*func)(double []));
void linmin(double p[], double xi[], int n, double *fret, double (*func)(double []));
void mnbrak(double *ax, double *bx, double *cx, double *fa, double *fb, double *fc,
            double (*func)(double));
double brent(double ax, double bx, double cx, double (*f)(double), double tol,
            double *xmin);
double f1dim(double x);
double keithrules(double x[]);

double func(double x[])
{
        return keithrules(x);
```

```
}
int ncom;
double *pcom,*xicom,(*nrfunc)(double []);
static double *Y,*X;
FILE *fp;
FILE *jp;

int main(void)
{
         int i,iter,j;
         double fret,**xi;
         static double
p[]={0.0,0.9,1.5,1.5,2.5,1.5,1.5,2.5,1.5,1.5,2.5,1.5,1.5,2.5,1.5,1.5,2.5,1.5,1.5,2.5,1.5,1.5,2.5,
1.5,1.5,2.5};
         X = (double *) malloc (nrowX*ncolX*sizeof(double));
         Y = (double *) malloc (nrowX*sizeof(double));
/* Dynamic Allocation of a Matrix: */
/* First, allocate pointers to rows */
/* Second, allocate rows and set pointers to them -- note that p[0] is
 * a pointer to p[0][0:n] -- that is a row -- so the syntax below is
 * p[0][0:nrow*ncol] */
/* Third, form the matrix from step two -- the pointer to the kth row,
 * p[k] is set equal to the memory
 * location of the number of nrows-1 times the number of columns plus
 * the memory location of p[0] */
         xi= (double **) malloc((NDIM+1)*sizeof(double*));
         xi[0]=(double *) malloc((NDIM+1)*(NDIM+1)*sizeof(double));
         for (i=0;i<NDIM+1;i++)xi[i]=xi[0]+i*(NDIM+1);
//
//
         jp = fopen("c:/docs_c_summer_course/data_POWELL.txt","w");
         if((fp = fopen("c:/docs_c_summer_course/data_ols.txt","r"))==NULL)
         {
                  printf("\nUnable to open file OLS_DATA.TXT: %s\n", strerror(errno));
                  exit(EXIT_FAILURE);
         }
         else {

                  fprintf(jp," Y and X = \n");
                  for(i=0;i<nrowX;i++)
                  {
                           fscanf(fp,"%lf",&Y[i]);
                           for(j=0;j<ncolX;j++)
                           {
                                    fscanf(fp,"%lf",&X[i+j*nrowX]);
                           }
                           fprintf(jp,"%10d %12.6f", i,Y[i]);
                           for(j=0;j<ncolX;j++)
                           {
                                    fprintf(jp,"%12.6f",X[i+j*nrowX]);
                           }
                           fprintf(jp,"\n");
                  }
         }
         for (i=1;i<=NDIM;i++)
                  for (j=1;j<=NDIM;j++)
                           xi[i][j]=(i == j ? 1.0 : 0.0);
/* if i == j is true then the value of x[i][j] = 1.0, otherwise =0.0 */
         powell(p,xi,NDIM,FTOL,&iter,&fret,func);
         fprintf(jp,"Iterations: %3d\n\n",iter);
         printf("Iterations: %3d\n\n",iter);
         fprintf(jp,"Minimum found at: \n");
         printf("Minimum found at: \n");
         for (i=1;i<=NDIM;i++) {
                  fprintf(jp,"%5d %12.6f\n",i, p[i]);
                  printf("%5d %12.6f\n",i, p[i]);
         }
         fprintf(jp,"\n\nMinimum function value = %12.6f \n\n",fret);
         printf("\n\nMinimum function value = %12.6f \n\n",fret);
```

```
	free(xi);
	free(Y);
	free(X);
	return 0;
}
void powell(double p[], double **xi, int n, double ftol, int *iter, double *fret,
	    double (*func)(double []))
{
	int i,ibig,j;
	double del,fp,fptt,t,*pt,*ptt,*xit;

	pt = (double *) malloc ((n+1)*sizeof(double));
	ptt = (double *) malloc ((n+1)*sizeof(double));
	xit = (double *) malloc ((n+1)*sizeof(double));
	*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);
			if (fptt-(*fret) > del) {
				del=fptt-(*fret);
				ibig=i;
			}
		}
		if (2.0*(fp-(*fret)) <= ftol*(fabs(fp)+fabs(*fret))+TINY) {
			free(xit);
			free(ptt);
			free(pt);
			return;
		}
		if (*iter == ITMAX)
		{
			printf("powell exceeding maximum iterations.");
			exit(EXIT_FAILURE);
		}
		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)*(fp-(*fret)-del)*(fp-(*fret)-del)-
del*(fp-fptt)*(fp-fptt);
			if (t < 0.0) {
				linmin(p,xit,n,fret,func);
				for (j=1;j<=n;j++) {
					xi[j][ibig]=xi[j][n];
					xi[j][n]=xit[j];
				}
			}
		}
	}
}


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

	ncom=n;
	pcom = (double *) malloc ((n+1)*sizeof(double));
	xicom = (double *) malloc ((n+1)*sizeof(double));
```

```
        nrfunc=func;
        for (j=1;j<=n;j++) {
                pcom[j]=p[j];
                xicom[j]=xi[j];
        }
        ax=0.0;
        xx=1.0;
        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(xicom);
        free(pcom);
}


void mnbrak(double *ax, double *bx, double *cx, double *fa, double *fb, double *fc,
            double (*func)(double))
{
        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(FMAX(fabs(q-r),TINY),q-r));
                ulim=(*bx)+GLIMIT*(*cx-*bx);
                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)
        }
}
```

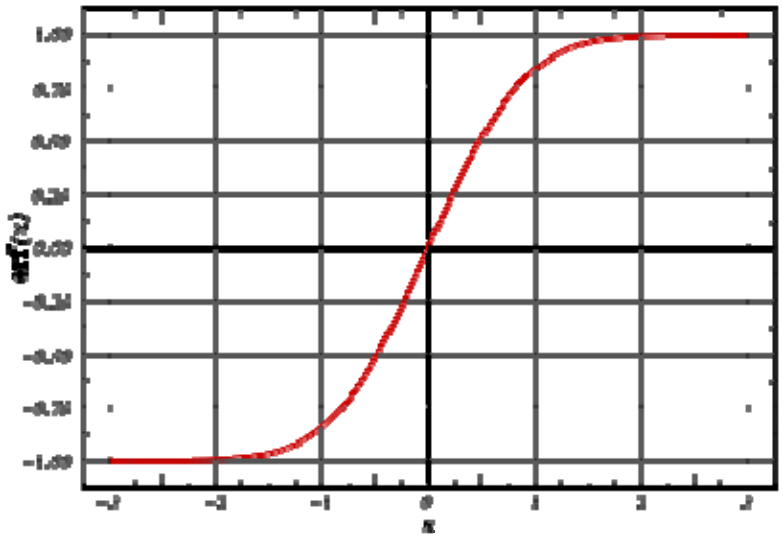
```
double brent(double ax, double bx, double cx, double (*f)(double), double tol,
             double *xmin)
{
         int iter;
         double a,b,d,etemp,fu,fv,fw,fx,p,q,r,tol1,tol2,u,v,w,x,xm;
         double e=0.0;

         a=(ax < cx ? ax : cx);
         b=(ax > cx ? ax : cx);
         x=w=v=bx;
         fw=fv=fx=(*f)(x);
         for (iter=1;iter<=ITMAX;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;
                           }
                  }
         }
         printf("Too many iterations in brent");
         exit(EXIT_FAILURE);
         *xmin=x;
         return fx;
}

double f1dim(double x)
{
         int j;
         double f,*xt;
```

```
        xt = (double *) malloc ((ncom+1)*sizeof(double));
        for (j=1;j<=ncom;j++) xt[j]=pcom[j]+x*xicom[j];
        f=(*nrfunc)(xt);
        free(xt);
        return f;
}

double keithrules(double x[])
{
        int i, j;
        double sum=0;
        double sumsquared=0;
        for(i=0;i<nrowX;i++)
        {
                sum=0.0;
                for(j=0;j<ncolX;j++)
                {
                        sum=sum+X[i+j*nrowX]*x[j+1];
                }
                sumsquared = sumsquared + (Y[i]-sum)*(Y[i]-sum);
        }
        printf("%lf\n",sumsquared);
        return sumsquared;

}
```

# Error function

## From Wikipedia, the free encyclopedia

Jump to: navigation, search



Plot of the error function

In mathematics, the **error function** (also called the **Gauss error function**) is a special function (non-elementary) of sigmoid shape which occurs in probability, statistics, materials science, and partial differential equations. It is defined as:

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}\,dt.$$

The **complementary error function**, denoted *erfc*, is defined in terms of the error function:

$$\begin{aligned} \operatorname{erfc}(x) &= 1 - \operatorname{erf}(x) \\ &= \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-t^2}\,dt. \end{aligned}$$

The **complex error function**, denoted *w*(*x*), (also known as the Faddeeva function) is also defined in terms of the error function:

$$w(x) = e^{-x^2} \operatorname{erfc}(-ix).$$

The error function is essentially identical to the standard normal cumulative distribution function, denoted Φ, as they differ only by scaling and translation. Indeed,

$$\Phi(x) = \frac{1}{2}\left[1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)\right] = \frac{1}{2}\operatorname{erfc}\left(-\frac{x}{\sqrt{2}}\right)$$

or rearranged for erf and erfc:

$$\operatorname{erf}(x) = 2\Phi\left(x\sqrt{2}\right) - 1$$
$$\operatorname{erfc}(x) = 2\left[1 - \Phi\left(x\sqrt{2}\right)\right].$$

## xamoeba_test_bush_probit.c

```
/**/
/* xamoeba_test.c  -- C code is from NUMERICAL RECIPES IN C --
 *                      Additional code written by Keith Poole
 *                      September - October 2009
 *
   Uses Nedler-Mead downhill simplex method to find a minimum/maximum
   of a multi-dimensional function
                 |->amotry->func
Main -> amoeba --|
                 |->func

                 Amoeba initializes the simplex and calls func for each
                 point in the simplex.  The simplex starts at the usual
                 triangluar definition plus the origin -- ndim +1.

                 func is the user defined function being
                 minimized/maximized;

                 Amotry -- (from description in NUMERICAL RECIPES)
                 Extrapolates by a factor, fac, through the face of the
                 simplex across from a high point, tries, and replaces
                 the high point if the new point is better.
*/

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

#define MP 9
#define NP 8
#define nrowX 432
```

```
#define ncolX 8
#define FTOL 0.0000001
#define TINY 1.0e-10
#define NMAX 10000
```

ETC ETC ETC

## The Function the Calculates the probit

```
double keithrules(double x[])
{
        int i, j, k1, k0, l1, l0;
        double sum=0;
        double phi=0.0;
        double xphi=0.0;
        double sumsquared=0;
        k1=0;
        k0=0;
        for(i=0;i<nrowX;i++)
        {
                sum=0.0;
                for(j=0;j<ncolX;j++)
                {
                        sum=sum+X[i+j*nrowX]*x[j+1];
                }
                phi = (erf(fabs(sum)/sqrt(2.0)))/2.0 + 0.5;
                xphi = phi;
                if(sum < 0.0)phi=1.0-xphi;
                xphi = phi;
                if(xphi > 0.99999999)phi=0.9999999;
                if(xphi < 0.00000001)phi=0.0000001;
// Voted for Bush
                if(Y[i] >= 50.0){
                        l1=1;
                        l0=0;
                        k1=k1+1;
                        sumsquared = sumsquared + log(phi);
                }
// Voted for Gore
                if(Y[i] < 50.0){
                        l1=0;
                        l0=1;
                        k0=k0+1;
                        sumsquared = sumsquared + log(1.0 - phi);
                }
        }
        printf("%lf\n",-sumsquared);
        return -sumsquared;

}
```

## STATA Probit Output

```
file C:\docs_c_summer_course\hdmg106_2009_fixed.dta saved


. probit ybush black00 south hispanic00 income owner00 dwnom1n dwnom2n

Iteration 0:   log likelihood = -299.27289
Iteration 1:   log likelihood = -155.22888
Iteration 2:   log likelihood = -134.72283
Iteration 3:   log likelihood = -130.45346
Iteration 4:   log likelihood =  -130.1108
Iteration 5:   log likelihood = -130.10873
Iteration 6:   log likelihood = -130.10873

Probit estimates                                  Number of obs   =        432
                                                  LR chi2(7)      =     338.33
                                                  Prob > chi2     =     0.0000
Log likelihood = -130.10873                       Pseudo R2       =     0.5653

------------------------------------------------------------------------------
       ybush |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
     black00 |  -.0285248   .0098015    -2.91   0.004    -.0477354   -.0093142
       south |   .7685989   .2545868     3.02   0.003      .269618     1.26758
  hispanic00 |  -.0089129   .0069185    -1.29   0.198    -.0224729     .004647
      income |  -.0240403   .0126564    -1.90   0.058    -.0488463    .0007658
     owner00 |   .0236413   .0143807     1.64   0.100    -.0045444    .0518271
     dwnom1n |   2.760012   .2621742    10.53   0.000     2.24616    3.273864
     dwnom2n |   1.136143   .2339367     4.86   0.000     .6776357    1.594651
       _cons |  -.9799419   1.079705    -0.91   0.364    -3.096126    1.136242
------------------------------------------------------------------------------

note: 1 failure and 0 successes completely determined.

. probit ygore black00 south hispanic00 income owner00 dwnom1n dwnom2n

Iteration 0:   log likelihood = -296.05574
Iteration 1:   log likelihood = -149.55538
Iteration 2:   log likelihood = -128.23315
Iteration 3:   log likelihood = -123.81516
Iteration 4:   log likelihood = -123.48492
Iteration 5:   log likelihood = -123.48326
Iteration 6:   log likelihood = -123.48326

Probit estimates                                  Number of obs   =        432
                                                  LR chi2(7)      =     345.14
                                                  Prob > chi2     =     0.0000
Log likelihood = -123.48326                       Pseudo R2       =     0.5829

------------------------------------------------------------------------------
       ygore |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
     black00 |    .037454   .0096159     3.90   0.000     .0186072    .0563007
       south |  -.5341551   .2623903    -2.04   0.042    -1.048431   -.0198796
  hispanic00 |   .0142289   .0068103     2.09   0.037      .000881    .0275768
      income |   .0331017   .0128455     2.58   0.010      .007925    .0582783
     owner00 |  -.0321204   .0135469    -2.37   0.018    -.0586718    -.005569
     dwnom1n |  -2.590046   .2538889   -10.20   0.000    -3.087659   -2.092433
     dwnom2n |  -.8584056   .2325336    -3.69   0.000    -1.314163   -.4026481
       _cons |   .6039696   1.017087     0.59   0.553    -1.389484    2.597424
------------------------------------------------------------------------------

note: 0 failures and 1 success completely determined.
```