optimization_machine_MDS.c – Analyzes Color Circle Data From Ekman (1954)

```
 434 INDIGO        100  86  42  42  18   6   7   4   2   7   9  12  13  16
 445 BLUE           86 100  50  44  22   9   7   7   2   4   7  11  13  14
 465                42  50 100  81  47  17  10   8   2   1   2   1   5   3
 472 BLUE-GREEN     42  44  81 100  54  25  10   9   2   1   0   1   2   4
 490                18  22  47  54 100  61  31  26   7   2   2   1   2   0
 504 GREEN           6   9  17  25  61 100  62  45  14   8   2   2   2   1
 537                 7   7  10  10  31  62 100  73  22  14   5   2   2   0
 555 YELLOW-GREEN    4   7   8   9  26  45  73 100  33  19   4   3   2   2
 584                 2   2   2   2   7  14  22  33 100  58  37  27  20  23
 600 YELLOW          7   4   1   1   2   8  14  19  58 100  74  50  41  28
 610                 9   7   2   0   2   2   5   4  37  74 100  76  62  55
 628 ORANGE-YELLOW  12  11   1   1   1   2   2   3  27  50  76 100  85  68
 651 ORANGE         13  13   5   2   2   2   2   2  20  41  62  85 100  76
 674 RED            16  14   3   4   0   1   0   2  23  28  55  68  76 100

*/

#include <stdio.h>
#include <stddef.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <time.h>
#include <R_ext/Lapack.h>
#include <R_ext/BLAS.h>
//
#define NDIM 28
#define NMAX 100000
#define ITMAX 2000
#define nrowX 14
#define ncolX 14


…..etc etc etc
```

THE PLUGIN FOR METRIC MDS

```
/*
  Setup for Metric Multidimensional Scaling -- No Constraints on the
  Coordinates
*/

double keithrules(double x[])
{
        int i, j;
        double sum=0;
        double sumsquared=0;
        double circledist, circledisthat;
        for(i=0;i<nrowX;i++)
        {
                for(j=0;j<ncolX;j++)
                {
                     sum=0.0;
                     circledist = (100.0 - X[i+j*nrowX])/50.0;
                     circledisthat = sqrt(pow((x[i+1]-x[j+1]),2.0) +
pow((x[i+1+nrowX]-x[j+1+nrowX]),2.0));
                      sumsquared=sumsquared+pow((circledist-
circledisthat),2.0);
                }
        }
//      printf("%lf\n",-sumsquared);
        return sumsquared;

}
```

AMOEBA DOES NOT ALWAYS WORK!!!

```
//
//  AMOEBA MINIMIZATION IS DONE FIRST
//
/* if i == (j+1) is true then the value of x[j] = 1.0, otherwise =0.0 */
        for (i=1;i<=NDIM+1;i++) {
                for (j=1;j<=NDIM;j++)
                x[j]=p[i][j]=(i == (j+1) ? 1.0 : 2.0*( (double)rand() /
((double)(RAND_MAX)+1))-1.0);
//              x[j]=p[i][j]=(i == (j+1) ? 1.0 : -1.0);
                y[i]=func(x);
                printf("%5d initial function value = %15.6f \n",i,y[i]);
        }
        amoeba(p,y,ndim,FTOL,func,&nfunc);
```

optimization_machine_king.c – Analyzes King Data on U.K. elections (1990)

```
#
#  VOTE SHARES -- CONSERVATIVE, LABOR, 3rd
# .4341994,  .4610934,  .1047072,
# .4797144,  .4877813,  .0325043,
# .4974225,  .4635792,  .0389983,
# .4935235,  .4384425,  .0680339,
# .4339797,  .4413256,  .1246946,
# .4187992,  .4803503,  .1008505,
# .4637579,  .430723,   .1055191,
# .3788168,  .3715876,  .2495956,
# .3584408,  .3925122,  .249047,
# .4387356,  .3693695,  .1918949,
# .4242528,  .2757294,  .3000178,
# .423       .308,      .269
#
#  SEATS -- CONSERVATIVES, LABOR, 3rd
#
#  300,  315,  12,
#  325,  295,  9,
#  345,  277,  8,
#  365,  258,  7,
#  304,  317,  9,
#  253,  364,  13,
#  330,  288,  12,
#  297,  301,  37,
#  277,  319,  39,
#  339,  269,  27,
#  397,  209,  45,
#  376,  229,  45
#
```

THE MLE RESULTS

```
> results
            [,1]       [,2]       [,3]         [,4]
[1,]  1.14217375 0.08810331  12.964027 1.187315e-06
[2,] -0.04587087 0.02420965  -1.894735 9.473289e-02
[3,] -1.60468315 0.09017014 -17.796169 1.017642e-07
> model
$par
[1]  1.14217375 -0.04587087 -1.60468315

$value
[1] 6118.279

$counts
function gradient
     264       NA

$convergence
[1] 0

$message
NULL
```

```
$hessian
           [,1]       [,2]      [,3]
[1,]  254.92549  -45.61907 -171.2843
[2,]  -45.61907 1866.32262 -112.0308
[3,] -171.28432 -112.03083  249.0333

*/
```

THE PLUGIN FOR KING PROBLEM

```
double keithrules(double x[])
{
        int i, j, nelection;
        double sum=0;
        double *lambda;
        double sumsquared=0;
        double rho_0, rhox;
        nelection = nrowX/2;
        lambda = (double *) malloc ((ncolX+1)*sizeof(double));
        rho_0 = x[1];
        lambda[1] = 0.0;
        lambda[2] = x[2];
        lambda[3] = x[3];
/**/
        for(i=0;i<nelection;i++)
        {
            rhox = rho_0;
            sum = 0.0;
           for(j=0;j<ncolX;j++)
            {
/*
#  Calculate constant term -- TT = Vote Share -- first 12 rows of
                                   king_data.txt
                             -- T  = Seats -- rows 13 - 24 of
                                   king_data.txt
*/
              sum = sum + exp(lambda[j+1] + rhox*log(X[i+j*nrowX]));
            }
            for(j=0;j<ncolX;j++)
            {
/*
#  Calculate Row
*/
                sumsquared = sumsquared + X[i+nelection+j*nrowX]*(lambda[j+1]
+ rhox*log(X[i+j*nrowX]) - log(sum));
            }
        }
        free(lambda);
         return(-sumsquared);
```