Minimax Approximation

The problem is to find a set of (m) coefficients on a set of (n) independent variables such that the absolute difference between an approximating function and the observed data (|data set| = q) is minimized.

OTHER NORMS

This problem may be solved for other norms (i.e. L1, L2, ..., L_∞). In general,

$$L_i = \sqrt[i]{[f(v_1,...,v_n)_k - p(v_1,...,v_n)_k]^i} \; ; \; k=1,...,q$$

when i = ∞ we have that

$$L_\infty = \sqrt[\infty]{[f(v_1,...,v_n)_k - p(v_1,...,v_n)_k]^\infty} = \max(f(v_1,...,v_n)_k - p(v_1,...,v_n)_k) \; ; \; k=1,...,q$$

where

$f(v_1,...,v_n)$ ($F(\mathbf{v})$) is the original (unknown) function from which the data proceeds
$p(v_1,...,v_n)$ ($P(\mathbf{v})$) is the inferred function

It is common to assume that $P(\mathbf{v})$ is a polynomial in which case

$$P(\mathbf{v}) = \sum_{v_1=0}^{d_1} \cdots \sum_{v_n=0}^{d_n} c_{i_1 \cdots i_n} v_1^{i_1} \cdots v_n^{i_n} \qquad (0)$$

We shall use the following example throughout. Generalization is straightforward.

READ THE DATA

CODE

```
import sys
import random
MaximumNumberOfIndependentVariables=5
DEBUG=False

def check_integer_ge1(i):
    while True:
        print("Give me the degree of independent variable "+str(i))
        try:
            while True:
                x=float(input())
                x=int(x)
                if (x>=1):
                    return x
                #endif
                print("Degree must be \"1\" or more")
                print("Try again")
            #endwhile
        except:
            print("Must be a number")
            print("Try again")
```

```
    ##endWhile
##endFunction

def NumCols(Linea,Linelen):
    cols=1
    for i in range(0,LineLen):
        if (Linea[i]=="\t"):
            cols=cols+1
        #endIf
    #endFor
    return cols
#endFunction

def getNum(Linea,numVar):
    countVar=-1
    first=0
    for i in range(0,numVar+1):
        x=""
        for j in range(first,len(Linea)):
            if (Linea[j]=="\t" or Linea[j]=="\n"):
                first=j+1
                countVar=countVar+1
                break
            else:
                x=x+Linea[j]
            #endif
        #endFor
        if (countVar==numVar):
            return float(x)
        #endIf
    #endFor
#endFunction

#
#               MAIN
#
DTA=input("Give me the name of the Data to Analyze: \t")
try:
    FN = open(DTA,"r")
    print("\t***\""+DTA+"\" was opened" )
except:
    print("\t***\""+DTA+"\" was not found" )
    sys.exit("**** End of Program ****")
#endTrydat
Tuples=FN.readlines()
n=len(Tuples)
print("Tuples:     "+str(n))
LineLen=len(Tuples[0])
Atribs=NumCols(Tuples[0],LineLen)
m=Atribs
print("Attributes: "+str(m))
print("INDEPENDENT VARIABLES: "+str(m-1))
IV=m-1
if (IV>MaximumNumberOfIndependentVariables):
    print("Exceeded number of maximum allowed independent variables")
    sys.out("***\t\tPROGRAM WILL NOW END\t\t***")
#endif
```

```
print("\tDEPENDENT VARIABLE IS IN COLUMN "+str(m))
print()
dummy=input("\"ENTER\" to continue\n")
FN = open(DTA,"r")              # REWIND
xy=list(range(0,n+1))
for i in range(0,n+1):
    xy[i]=list(range(0,m+1))
#
#       PON LOS DATOS EN xy[i][j] EN PUNTO FLOTANTE
#
for i in range(0,n):             # n tuples  (i=0,1,...,n-1)
    for j in range(0,m):         # m columns (j=0,1,...,m-1)
        L=Tuples[i]
        Num=getNum(L,j)          #"Num" is Floating point
        xy[i+1][j+1]=Num
    #endFor
#endFor
FN.close()
print("ORIGINAL DATA TUPLES")
for i in range(1,n+1):
    Tuple_i=""
    for j in range(1,m+1):
        Tuple_i=Tuple_i+("%6.3f\t" % (xy[i][j]))
    #endfor
    print(Tuple_i)
#endfor
print()
```

RUN

```
"ENTER" to continue

ORIGINAL DATA TUPLES
 2.000   3.000   2.000  -11.000
 1.000  -2.000   3.000   30.500
 2.000   4.000   1.000  -21.000
 2.500  -1.000   4.000   80.000
 3.500  -2.000   2.000   53.500
 3.000   2.000   3.000   15.500
 5.000   4.000   1.000  -37.500
 9.000   6.000   1.000 -101.500
 2.000   8.000   2.000  -56.000
 5.000   4.000   3.000  -17.500
 9.000   6.000   4.000  -61.000
 2.000   8.000  -1.000  -17.000
 1.000   1.000   1.000   -0.500
 1.000   2.000   1.000   -5.500
 2.000   1.000   2.000    7.000
-2.000   3.000   3.000  -12.000
-1.000   2.000   4.000  -19.000
 1.500   2.000   3.000    5.750
 2.000   1.500   1.000   -3.500
-1.000  -1.500  -2.000    1.000
 1.250  -2.000  -2.000   14.250
 2.000   2.500  -1.000   -0.500
```

The original data is displayed in the next table. There are three independent variables.

Data0

V01	V02	V03	F
-2.0122377610	-2.9190930940	-2.0426922110	0.9395592270
-1.0406407350	-0.8646951770	-2.6058760410	1.4606832120
2.0733753260	-0.1129651390	-1.5962931050	0.9994078010
1.6338680820	-2.5136014250	0.9995281440	0.7166767470
-1.5771060590	-1.4906475010	1.5502652410	1.0136229110
1.1202277810	-0.9750042970	1.6886113700	0.9036104460
2.3696771920	-0.1323339380	1.9990939530	1.0170010520
-1.4051188310	1.5959250860	1.6376465650	1.0658522860
2.2233661810	2.9472279450	2.3937435500	1.0561271550
-2.8907702070	-2.9862389890	-1.4770551610	0.9872026620
-0.6498412240	-2.8226975850	-2.8770917220	1.2877591530
-0.6656991750	-2.5636340340	-2.8422699990	1.2346941390
-0.9479532760	-2.5811643430	-2.6956741590	1.4326872580
-2.3688476990	-1.1852915940	-2.5528598730	0.5551251630
-2.3103407450	-2.5884950280	-1.3664166890	1.0000000000
-2.2491356080	-1.4928414990	-2.7483471230	1.2392240550
-2.8069015990	-1.3126406780	-2.8461536360	1.2601623730
-2.5338168170	-1.3470004690	-2.3836474680	1.0420081510
-0.8177116800	-1.4784499750	-2.4402897540	0.3379039350

For this example, we select the simple case where degrees $d_1 = d_2 = d_3 = 1$.

Degrees

Variable	Degree
v01	1
v02	1
v03	1

The sought for expression is of the form:

$$P(X,Y,Z)=C_{000} + C_{001}Z + C_{010}Y + C_{011}YZ+C_{100}X+C_{101}XZ+C_{110}XY+C_{111}XYZ \quad (1)$$

MAPPING

In order to find the approximant of the desired form all one has to do is map the original data to the values of the monomials. In the case of a full polynomial expansion as on (0) there is an actual mapping from R^n into R^D where $D = (d_1+1)x. \; . \; .x(d_n+1)$. The matrix mapping from the original space into the polynomial space is shown. The subindices indicate the degrees to which the independent variables have been raised. We have mapped the original X-Y-Z-F data from R^4 into R^9. The column reading "Sign" corresponds to the values of the signs needed to make this system the set of conditions to fit X-Y-Z into (1) in the minimax sense.

Data1

Sign	T000	T001	T010	T011	T100	T101	T110	T111	F
	1.0000000000	-2.0426922110	-2.9190930940	5.9628087263	-2.0122377610	4.1103824011	5.8739093516	-11.9985888807	0.9395592270
	1.0000000000	-2.6058760410	-0.8646951770	2.2532884445	-1.0406407350	2.7117807586	0.8998370245	-2.3448637431	1.4606832120
	1.0000000000	-1.5962931050	-0.1129651390	0.1803254725	2.0733753260	-3.3097147370	-0.2342191319	0.3738823853	0.9994078010
	1.0000000000	0.9995281440	-2.5136014250	-2.5124153671	1.6338680820	1.6330971315	-4.1068931392	-4.1049552770	0.7166767470
	1.0000000000	1.5502652410	-1.4906475010	-2.3108990074	-1.5771060590	-2.4449327046	2.3509092057	3.6445328263	1.0136229110
	1.0000000000	1.6886113700	-0.9750042970	-1.6464033417	1.1202277810	1.8916293680	-1.0922269001	-1.8443467621	0.9036104460
	1.0000000000	1.9990939530	-0.1323339380	-0.2645479752	2.3696771920	4.7372073451	-0.3135887146	-0.6268933031	1.0170010520
	1.0000000000	1.6376465650	1.5959250860	2.6135612351	-1.4051188310	-2.3010880270	-2.2424643912	-3.6723641074	1.0658522860
	1.0000000000	2.3937435500	2.9472279450	7.0549078837	2.2233661810	5.3221684551	6.5527669406	15.6856435987	1.0561271550

The minimum maximum absolute error [$\varepsilon_\theta = max(\varepsilon_i)$] in this example results from solving

$$\varepsilon_i + C_{000} + C_{001}Z_i + C_{010}Y_i + C_{011}Y_iZ_i + C_{100}X_i + C_{101}X_iZ_i + C_{110}X_iY_i + C_{111}X_iY_iZ_i = F_i \quad i=1,\ldots,9 \qquad (2)$$

The C_{ijk}'s will be those of the minimax approximant if a) $\varepsilon_i = \varepsilon_j \ \forall i,j$ (i.e. all the errors have the same absolute size) and b) The signs of the ε_i ($sgn(\varepsilon_i) = \sigma_i$) are adequately chosen. The system of (2) may be written as

$$\begin{pmatrix} \sigma_1 & 1 & Z_1 & Y_1 & Y_1Z_1 & \ldots & X_1Y_1Z_1 \\ \sigma_2 & 1 & Z_2 & Y_2 & Y_2Z_2 & \ldots & X_2Y_2Z_2 \\ . & . & . & . & . & . & . \\ \sigma_9 & 1 & Z_9 & Y_9 & Y_9Z_9 & \ldots & X_9Y_9Z_9 \end{pmatrix} \begin{pmatrix} \varepsilon_\theta \\ C_{000} \\ . \\ C_{111} \end{pmatrix} = \begin{pmatrix} F_1 \\ F_2 \\ . \\ F_9 \end{pmatrix} \qquad (3)$$

ε_θ will be minimized if $\sum_{i=1}^{9} \sigma_i C_i$ is maximized (where C_i is the cofactor of the i-th row). Therefore, solving (3) will yield the minimax coefficients if $\sigma_i = sign(C_i)$ $\forall i$.

MAP THE DATA

CODE

```
Deg=list(range(0,m))
for i in range(1,m):
    Deg[i]=check_integer_ge1(i)
#endfor
Terms=1
for i in range(1,m):
    Terms=Terms*(Deg[i]+1)
#endFor
#                X
map=list(range(0,n+1))
for i in range(0,n+1):
    map[i]=list(range(0,Terms+3))       # error+Terms+Dependent variable
#endFor
iv=list(range(m))
for i in range(1,n+1):
    for j in range(1,m):
        iv[j]=xy[i][j]
    #endfor
    k=1
    while k<=Terms:
        for p1 in range(0,Deg[1]+1):
```

```
            if (IV==1):
                map[i][k+1]=pow(iv[1],p1)
                k=k+1
                continue
            #endif
            for p2 in range(0,Deg[2]+1):
                if (IV==2):
                    map[i][k+1]=pow(iv[1],p1)*pow(iv[2],p2)
                    k=k+1
                    continue
                #endif
                for p3 in range(0,Deg[3]+1):
                    if (IV==3):
                        map[i][k+1]=pow(iv[1],p1)*pow(iv[2],p2)*pow(iv[3],p3)
                        k=k+1
                        continue
                    #endif
                    for p4 in range(0,Deg[4]+1):
                        if (IV==4):
                           map[i][k+1]=pow(iv[1],p1)*pow(iv[2],p2)
                                       *pow(iv[3],p3)*pow(iv[4],p4)
                            k=k+1
                            continue
                        #endfor
                        for p5 in range(0,Deg[5]+1):
                            map[i][k+1]=pow(iv[1],p1)*pow(iv[2],p2)*pow(iv[3],p3)
                                        *pow(iv[4],p4)*pow(iv[5],p5)
                            k=k+1
                        #endfor
                    #endfor
                #endfor
            #endfor
        #endfor
    #endfor
    map[i][k+1]=xy[i][m]
    DEBUG=False
    if (DEBUG):
        print("map["+str(i)+"]["+str(k+1)+"]="+str(map[i][k+1]))
    #endif
#endFor
DEBUG=True
print("There are "+str(Terms+2)+" attributes, "+str(n)+" tuples in MAP")
dummy=input("\"ENTER\" to continue\n")
```

RUN

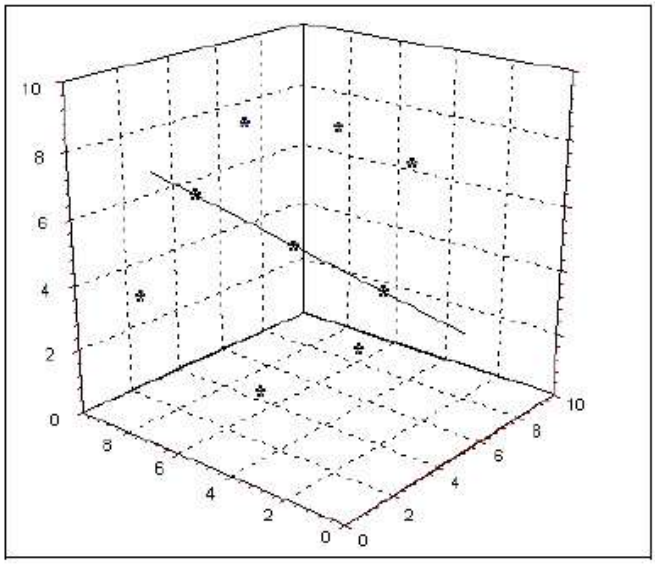
```
1
Give me the degree of independent variable 2

1
Give me the degree of independent variable 3

1
There are 10 attributes, 22 tuples in MAP

"ENTER" to continue
```

DATA MAPPED INTO THE POWERS OF THE INDEPENDENT VARIABLES

1.000000	1.000000	2.000000	3.000000	6.000000	2.000000	4.000000	6.000000	12.000000	-11.000000
1.000000	1.000000	3.000000	-2.000000	-6.000000	1.000000	3.000000	-2.000000	-6.000000	30.500000
1.000000	1.000000	1.000000	4.000000	4.000000	2.000000	2.000000	8.000000	8.000000	-21.000000
1.000000	1.000000	4.000000	-1.000000	-4.000000	2.500000	10.000000	-2.500000	-10.000000	80.000000
1.000000	1.000000	2.000000	-2.000000	-4.000000	3.500000	7.000000	-7.000000	-14.000000	53.500000
1.000000	1.000000	3.000000	2.000000	6.000000	3.000000	9.000000	6.000000	18.000000	15.500000
1.000000	1.000000	1.000000	4.000000	4.000000	5.000000	5.000000	20.000000	20.000000	-37.500000
1.000000	1.000000	1.000000	6.000000	6.000000	9.000000	9.000000	54.000000	54.000000	-101.500000
1.000000	1.000000	2.000000	8.000000	16.000000	2.000000	4.000000	16.000000	32.000000	-56.000000
1.000000	1.000000	3.000000	4.000000	12.000000	5.000000	15.000000	20.000000	60.000000	-17.500000
1.000000	1.000000	4.000000	6.000000	24.000000	9.000000	36.000000	54.000000	216.000000	-61.000000
1.000000	1.000000	-1.000000	8.000000	-8.000000	2.000000	-2.000000	16.000000	-16.000000	-17.000000
1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	-0.500000
1.000000	1.000000	1.000000	2.000000	2.000000	1.000000	1.000000	2.000000	2.000000	-5.500000
1.000000	1.000000	2.000000	1.000000	2.000000	2.000000	4.000000	2.000000	4.000000	7.000000
1.000000	1.000000	3.000000	3.000000	9.000000	-2.000000	-6.000000	-6.000000	-18.000000	-12.000000
1.000000	1.000000	4.000000	2.000000	8.000000	-1.000000	-4.000000	-2.000000	-8.000000	-19.000000
1.000000	1.000000	3.000000	2.000000	6.000000	1.500000	4.500000	3.000000	9.000000	5.750000
1.000000	1.000000	1.000000	1.500000	1.500000	2.000000	2.000000	3.000000	3.000000	-3.500000
1.000000	1.000000	-2.000000	-1.500000	3.000000	-1.000000	2.000000	1.500000	-3.000000	1.000000
1.000000	1.000000	-2.000000	-2.000000	4.000000	1.250000	-2.500000	-2.500000	5.000000	14.250000
1.000000	1.000000	-1.000000	2.500000	-2.500000	2.000000	-2.000000	5.000000	-5.000000	-0.500000

PERTURBATION AND STABILITY

It may be seen that it is easy to produce (from the above procedure) a set of data where a set of rows or columns may be linearly dependent.



Linear Dependence

In such case the system of equations which will be formulated may become numerically unstable or simply have no solution. Therefore, we may replace the data in the matrix above by

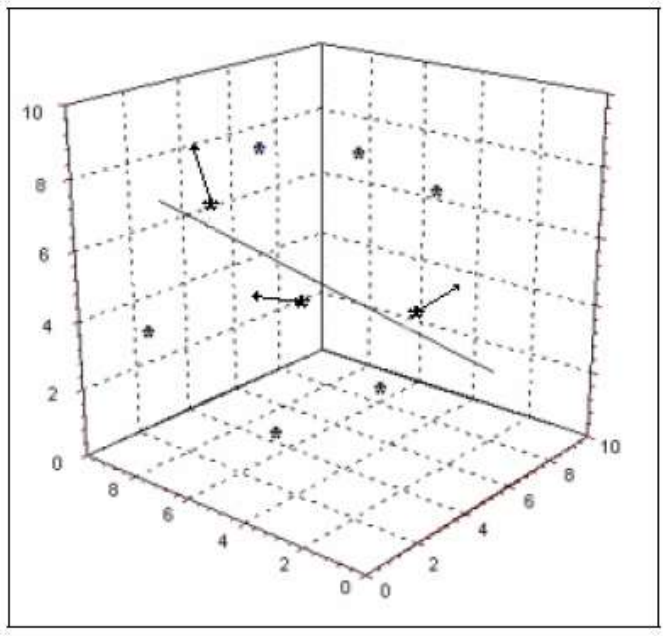
Data1

Sign	T000	T001	T010	T011	T100	T101	T110	T111	F
	1.0000002947	-2.0426934542	-2.9190948439	5.9628128318	-2.0122379805	4.1103846734	5.8739107193	-11.9985909431	0.9395592270
	1.0000001451	-2.6058774823	-0.8646953813	2.2532896358	-1.0406410073	2.7117814473	0.8998373202	-2.3448653968	1.4606832120
	1.0000003730	-1.5962934333	-0.1129651401	0.1803255071	2.0733767056	-3.3097176445	-0.2342191988	0.3738825663	0.9994078010
	1.0000006821	0.9995282326	-2.5136023547	-2.5124160510	1.6338689682	1.6330976019	-4.1068941060	-4.1049591569	0.7166767470
	1.0000008097	1.5502658941	-1.4906477637	-2.3108995339	-1.5771072808	-2.4449334470	2.3509109827	3.6445329588	1.0136229110
	1.0000001695	1.6886125900	-0.9750050670	-1.6464047853	1.1202284282	1.8916307211	-1.0922271731	-1.8443474373	0.9036104460
	1.0000000567	1.9990954077	-0.1323339689	-0.2645481611	2.3696789383	4.7372102085	-0.3135888350	-0.6268934077	1.0170010520
	1.0000004153	1.6376475750	1.5959257420	2.6135637822	-1.4051194020	-2.3010902786	-2.2424657148	-3.6723643647	1.0658522860
	1.0000001685	2.3937448790	2.9472291029	7.0549126621	2.2233672264	5.3221722778	6.5527710748	15.6856544940	1.0561271550

where the elements of the data matrix **X** have been replaced by **X*** as follows:

$$X_i^* = \begin{cases} X_i \times (1 + \rho_u \cdot \delta_H) & if\ X_i \neq 0 \\ \rho_u \cdot \delta_H & if\ X_i = 0 \end{cases}$$

where ρ_u denotes a uniformly distributed random variable such that $0 \leq \rho_u < 1$ and $\delta_H = 10^{-6}$. As may be observed from the illustration above, in **X*** all the elements of the first row (corresponding to $X^0Y^0Z^0$ (and, therefore originally identical to 1) have been slightly disturbed so that they are all different but very closely preserve the original value. This tends to replace linearly dependent vectors by linearly independent ones.



Random Displacement

It is even possible to have several identical vectors in X without disabling the algorithm.

PERTURBATION AND STABILITY

CODE

```
#
#            STABILIZATION
#
def stabilize():
    for i in range(1,n+1):
        for j in range(1,Terms+3):
            if (abs(map[i][j])<10e-6):
                map[i][j]=random.uniform(0,+10e-6)                #ALMOST 0
            else:
                map[i][j]=map[i][j]*(1+random.uniform(0,+10e-6))  #ALMOST XY[i][1]
            #endif
        #endfor
    #endfor
    return
#endStabilize
```

EXAMPLE. Consider the simple polynomial

$$Z = X^2 + XY - 2Y + 1$$

in which C00=+1; C01=-2; C11=+1 and C20=+1. We generate 200 vectors by calculating the values of Z for

$$X_i = 6\rho_u - 3$$

and

$$Y_i = 6\rho_u - 3$$

However, we make sure that there are *several* copies of every (X, Y, Z) vector as shown in the next figure.

V01	V02	F
-2.1073677329	-1.3512793216	10.9911998454
-2.1073677329	-1.3512793216	10.9911998454
-2.1073677329	-1.3512793216	10.9911998454
-2.1073677329	-1.3512793216	10.9911998454
-2.1073677329	-1.3512793216	10.9911998454
-2.1073677329	-1.3512793216	10.9911998454
-2.1073677329	-1.3512793216	10.9911998454
-2.1073677329	-1.3512793216	10.9911998454
-2.1073677329	-1.3512793216	10.9911998454
-2.1073677329	-1.3512793216	10.9911998454

Trying to solve this problem for $d_1 = d_2 = 2$ will lead the algorithm to select an initial internal data set as follows:

Sign	T00	T01	T02	T10	T11	T12	T20	T21	T22	F
	1.0000000000	-1.3512793216	1.8259558050	-2.1073677329	2.8476424405	-3.8479603451	4.4409987617	-6.0010297939	8.1090674688	10.9911998454
	1.0000000000	-1.3512793216	1.8259558050	-2.1073677329	2.8476424405	-3.8479603451	4.4409987617	-6.0010297939	8.1090674688	10.9911998454
	1.0000000000	-1.3512793216	1.8259558050	-2.1073677329	2.8476424405	-3.8479603451	4.4409987617	-6.0010297939	8.1090674688	10.9911998454
	1.0000000000	-1.3512793216	1.8259558050	-2.1073677329	2.8476424405	-3.8479603451	4.4409987617	-6.0010297939	8.1090674688	10.9911998454
	1.0000000000	-1.3512793216	1.8259558050	-2.1073677329	2.8476424405	-3.8479603451	4.4409987617	-6.0010297939	8.1090674688	10.9911998454
	1.0000000000	-1.3512793216	1.8259558050	-2.1073677329	2.8476424405	-3.8479603451	4.4409987617	-6.0010297939	8.1090674688	10.9911998454
	1.0000000000	-1.3512793216	1.8259558050	-2.1073677329	2.8476424405	-3.8479603451	4.4409987617	-6.0010297939	8.1090674688	10.9911998454
	1.0000000000	-1.3512793216	1.8259558050	-2.1073677329	2.8476424405	-3.8479603451	4.4409987617	-6.0010297939	8.1090674688	10.9911998454
	1.0000000000	-1.3512793216	1.8259558050	-2.1073677329	2.8476424405	-3.8479603451	4.4409987617	-6.0010297939	8.1090674688	10.9911998454
	1.0000000000	-1.3512793216	1.8259558050	-2.1073677329	2.8476424405	-3.8479603451	4.4409987617	-6.0010297939	8.1090674688	10.9911998454

Notice that we have mapped a problem originally in $\mathbf{R}^3$ to one in $\mathbf{R}^{10}$. Our first aim is to find the minimax signs which are indicated in the above matrix under "Signs". But, of course, this is impossible when linear dependence between the vectors has been induced, as here. When required to solve the system of equations of the cofactor theorem (see following paragraphs) the system responds with the following warning

PERTURBATION AND STABILITY

RUN

```
Give me the name of the Data to Analyze:    Perturbation and Stability exampe.txt
        ***"Perturbation and Stability exampe.txt" was opened
Tuples:     56
Attributes: 3
INDEPENDENT VARIABLES: 2
        DEPENDENT VARIABLE IS IN COLUMN 3


"ENTER" to continue

ORIGINAL DATA TUPLES
 1.109   2.196   0.272
 1.109   2.196   0.272
 1.109   2.196   0.272
 1.109   2.196   0.272
 1.109   2.196   0.272
 1.109   2.196   0.272
 1.109   2.196   0.272
 1.109   2.196   0.272
-2.527   0.568   4.815
 0.977  -1.415   3.402
 1.058  -1.572   3.600
-1.962  -0.499   6.828
 2.148   0.563   5.697
 0.552   0.887   0.021
 1.861  -1.159   4.626
-2.456  -0.507   9.292
 0.598   1.215  -0.347

• • •
 0.853  -2.804   4.944
-1.645  -1.686   9.855
 0.772   0.706   0.730
 2.289  -2.627   5.481
-2.622  -1.232  13.571
 1.010   2.500  -0.457
 2.306   2.325   7.030


"ENTER" to continue
```

```
"ENTER" to continue

DATA MAPPED INTO THE POWERS OF THE INDEPENDENT VARIABLES
  1.000000    1.000000    2.195600    4.820659    1.108500    2.433823    5.343701    1.228772    2.697892    5.923492    0.271500
  1.000000    1.000000    2.195600    4.820659    1.108500    2.433823    5.343701    1.228772    2.697892    5.923492    0.271500
  1.000000    1.000000    2.195600    4.820659    1.108500    2.433823    5.343701    1.228772    2.697892    5.923492    0.271500
  1.000000    1.000000    2.195600    4.820659    1.108500    2.433823    5.343701    1.228772    2.697892    5.923492    0.271500
  1.000000    1.000000    2.195600    4.820659    1.108500    2.433823    5.343701    1.228772    2.697892    5.923492    0.271500
  1.000000    1.000000    2.195600    4.820659    1.108500    2.433823    5.343701    1.228772    2.697892    5.923492    0.271500
  1.000000    1.000000    2.195600    4.820659    1.108500    2.433823    5.343701    1.228772    2.697892    5.923492    0.271500
  1.000000    1.000000    2.195600    4.820659    1.108500    2.433823    5.343701    1.228772    2.697892    5.923492    0.271500

• • •
    1.000000    1.000000   -2.765500    7.647990    0.207800   -0.574671    1.589252    0.043181   -0.119417    0.330247    5.999500
    1.000000    1.000000   -2.734400    7.476943    2.388900   -6.532208   17.861670    5.706843  -15.604792   42.669743    5.643300
    1.000000    1.000000   -2.804400    7.864659    0.852900   -2.391873    6.707768    0.727438   -2.040028    5.721055    4.944400
    1.000000    1.000000   -1.686300    2.843608   -1.645500    2.774807   -4.679156    2.707670   -4.565944    7.699552    9.855100
    1.000000    1.000000    0.705800    0.498154    0.772300    0.545089    0.384724    0.596447    0.420972    0.297122    0.729800
    1.000000    1.000000   -2.627300    6.902705    2.289300   -6.014678   15.802363    5.240894  -13.769402   36.176350    5.480900
    1.000000    1.000000   -1.231900    1.517578   -2.622400    3.230535   -3.979696    6.876982   -8.471754   10.436354   13.570900
    1.000000    1.000000    2.499700    6.248500    1.009500    2.523447    6.307861    1.019090    2.547420    6.367786   -0.456700
    1.000000    1.000000    2.325500    5.407950    2.306200    5.363068   12.471815    5.318558   12.368308   28.762499    7.030300

  "ENTER" to continue

  Unstable system
  An exception has occurred
```

Now we introduce random perturbations, as discussed, to get

```
DATA MAPPED INTO THE POWERS OF THE INDEPENDENT VARIABLES
  1.000000    1.000009    2.195600    4.820687    1.108501    2.433844    5.343713    1.228782    2.697907    5.923500    0.271500
  1.000009    1.000006    2.195605    4.820669    1.108505    2.433832    5.343709    1.228774    2.697918    5.923493    0.271502
  1.000002    1.000002    2.195602    4.820682    1.108502    2.433833    5.343702    1.228780    2.697907    5.923542    0.271500
  1.000009    1.000002    2.195601    4.820662    1.108503    2.433832    5.343710    1.228779    2.697899    5.923534    0.271502
  1.000009    1.000000    2.195603    4.820680    1.108509    2.433824    5.343715    1.228777    2.697909    5.923514    0.271500
  1.000002    1.000003    2.195609    4.820693    1.108511    2.433847    5.343747    1.228775    2.697910    5.923531    0.271502
  1.000010    1.000000    2.195605    4.820703    1.108502    2.433845    5.343746    1.228777    2.697906    5.923524    0.271501
  1.000001    1.000003    2.195609    4.820681    1.108506    2.433832    5.343728    1.228778    2.697913    5.923495    0.271500
  1.000003    1.000001    0.568305    0.322968   -2.527413   -1.436327   -0.816266    6.387778    3.630160    2.063031    4.814638
  1.000009    1.000007   -1.415411    2.003372    0.976901   -1.382716    1.957086    0.954336   -1.350770    1.911872    3.402423
  1.000007    1.000001   -1.571805    2.470577    1.057700   -1.662499    2.613109    1.118733   -1.758433    2.763884    3.599914
  1.000003    1.000006   -0.499402    0.249401   -1.961908    0.979774   -0.489302    3.849088   -1.922217    0.959959    6.827812

• • •
```

Sign	T00	T01	T02	T10	T11	T12	T20	T21	T22	F
	1.0000006654	-1.3512805087	1.8259563269	-2.1073687529	2.8476443829	-3.8479606863	4.4410004043	-6.0010314274	8.1090718673	10.9911998454
	1.0000002880	-1.3512796397	1.8259575308	-2.1073694392	2.8476436401	-3.8479610231	4.4409997735	-6.0010344429	8.1090699310	10.9911998454
	1.0000007559	-1.3512793707	1.8259561145	-2.1073692555	2.8476446895	-3.8479637191	4.4410013274	-6.0010340865	8.1090694954	10.9911998454
	1.0000003661	-1.3512793982	1.8259571337	-2.1073682244	2.8476444420	-3.8479631807	4.4410014460	-6.0010320983	8.1090688214	10.9911998454
	1.0000004153	-1.3512801550	1.8259565556	-2.1073697867	2.8476435977	-3.8479641103	4.4410013829	-6.0010302143	8.1090688355	10.9911998454
	1.0000005552	-1.3512798525	1.8259570417	-2.1073687237	2.8476444859	-3.8479627728	4.4410018464	-6.0010304615	8.1090719114	10.9911998454
	1.0000007591	-1.3512796690	1.8259566566	-2.1073694982	2.8476430822	-3.8479605248	4.4410003824	-6.0010334220	8.1090675432	10.9911998454
	1.0000001773	-1.3512794206	1.8259559074	-2.1073692913	2.8476445279	-3.8479618668	4.4409997344	-6.0010299872	8.1090698043	10.9911998454
	1.0000001579	-1.3512802443	1.8259575173	-2.1073698107	2.8476441292	-3.8479609591	4.4410016050	-6.0010298376	8.1090706155	10.9911998454
	1.0000008149	-1.3512803992	1.8259567211	-2.1073686070	2.8476440715	-3.8479609759	4.4410001633	-6.0010322265	8.1090743317	10.9911998454

In this new set no two vectors are alike. Therefore we are able to find the minimax signs:

Sign	T00	T01	T02	T10	T11	T12	T20	T21	T22	F
1	1.0000006654	-1.3512805087	1.8259563269	-2.1073687529	2.8476443829	-3.8479606863	4.4410004043	-6.0010314274	8.1090718673	10.9911998454
1	1.0000002880	-1.3512796397	1.8259575308	-2.1073694392	2.8476436401	-3.8479610231	4.4409997735	-6.0010344429	8.1090699310	10.9911998454
-1	1.0000007559	-1.3512793707	1.8259561145	-2.1073692555	2.8476446895	-3.8479637191	4.4410013274	-6.0010340865	8.1090694954	10.9911998454
-1	1.0000003661	-1.3512793982	1.8259571337	-2.1073682244	2.8476444420	-3.8479631807	4.4410014460	-6.0010320983	8.1090688214	10.9911998454
1	1.0000004153	-1.3512801550	1.8259565556	-2.1073697867	2.8476435977	-3.8479641103	4.4410013829	-6.0010302143	8.1090688355	10.9911998454
1	1.0000005552	-1.3512798525	1.8259570417	-2.1073687237	2.8476444859	-3.8479627728	4.4410018464	-6.0010304615	8.1090719114	10.9911998454
-1	1.0000007591	-1.3512796690	1.8259566566	-2.1073694982	2.8476430822	-3.8479605248	4.4410003824	-6.0010334220	8.1090675432	10.9911998454
-1	1.0000001773	-1.3512794206	1.8259559074	-2.1073692913	2.8476445279	-3.8479618668	4.4409997344	-6.0010299872	8.1090698043	10.9911998454
-1	1.0000001579	-1.3512802443	1.8259575173	-2.1073698107	2.8476441292	-3.8479609591	4.4410016050	-6.0010298376	8.1090706155	10.9911998454
-1	1.0000008149	-1.3512803992	1.8259567211	-2.1073686070	2.8476440715	-3.8479609759	4.4410001633	-6.0010322265	8.1090743317	10.9911998454

Solving the corresponding system of equations yields:

Error	0.0000066910
C00	0.9999992574
C01	-2.0000002653
C02	0.0000009330
C10	0.0000023226
C11	0.9999978573
C12	0.0000007425
C20	0.9999996189
C21	-0.0000008437
C22	0.0000005361

```
Minimax Signs

s[  0 ] =   1.000000
s[  1 ] =   1.000000
s[  2 ] =  -1.000000
s[  3 ] =   1.000000
s[  4 ] =   1.000000
s[  5 ] =  -1.000000
s[  6 ] =   1.000000
s[  7 ] =  -1.000000
s[  8 ] =  -1.000000
s[  9 ] =  -1.000000

DATA FOR MINIMAX APPROXIMATION
  1.000000       1.000009       2.195600       4.820687       1.108501       2.433844       5.343713       1.228782       2.697907       5.923500       0.271500
  1.000000       1.000006       2.195605       4.820669       1.108505       2.433832       5.343709       1.228774       2.697918       5.923493       0.271502
 -1.000000       1.000002       2.195602       4.820682       1.108502       2.433833       5.343702       1.228780       2.697907       5.923542       0.271500
  1.000000       1.000002       2.195601       4.820662       1.108503       2.433832       5.343710       1.228779       2.697899       5.923534       0.271502
  1.000000       1.000000       2.195603       4.820680       1.108509       2.433824       5.343715       1.228777       2.697909       5.923514       0.271500
 -1.000000       1.000003       2.195609       4.820693       1.108511       2.433847       5.343747       1.228775       2.697910       5.923531       0.271502
  1.000000       1.000000       2.195605       4.820703       1.108502       2.433845       5.343746       1.228777       2.697906       5.923524       0.271501
 -1.000000       1.000003       2.195609       4.820681       1.108506       2.433832       5.343728       1.228778       2.697913       5.923495       0.271500
 -1.000000       1.000001       0.568305       0.322968      -2.527413      -1.436327      -0.816266       6.387778       3.630160       2.063031       4.814638
 -1.000000       1.000007      -1.415411       2.003372       0.976901      -1.382716       1.957086       0.954336      -1.350770       1.911872       3.402423


Epsi =-7.08986988202e-05
c[00]=+1.00021701471       ≈ +1
c[01]=-1.99995996095       ≈ -2
c[02]=-0.00015405042
c[10]=+0.00011859347
c[11]=+0.99998990101       ≈ +1
c[12]=-2.73792759378e-05
c[20]=+0.99996098039       ≈ +1
c[21]=-1.15181200849e-05
c[22]=+3.54162838214e-05
```

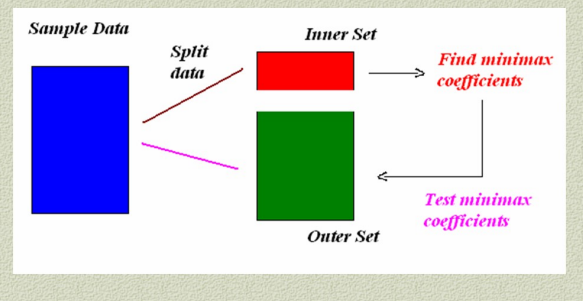
Compare these coefficients with those of the original equation: $Z = X^2 + XY - 2Y + 1$ (C00=+1; C01=-2; C11=+1 and C20=+1).

The minimax error is very closely zero [with an error of O(10^{-5})] and the approximation coefficients are very closely right [also with an error of O(10^{-5g})] and no further iterations are needed. It may be shown that the relative approximation error is of order O(δ_H). More precisely:

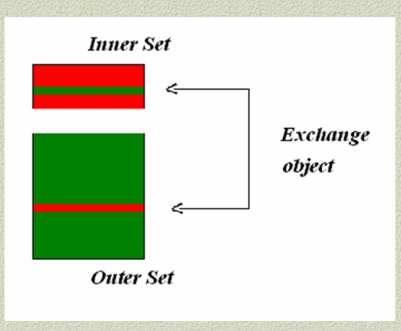
$$\frac{\left|F_\tau^* - F_\tau\right|}{F_\tau} < \delta_H \sum_i d_i \tag{4}$$

where F_τ^* is the value of the function approximating the disturbed vectors at the convergence (target) step τ of the algorithm, F_τ is the value of function approximating the original undisturbed vectors at the convergence (target) step τ of the algorithm, δ_H is the size of the perturbation constant and d_i denotes the highest allowed degree of the *i-th* variable.

The Ascent Algorithm, Phase 1



The Ascent Algorithm, Phase 2



The Minimax Polynomial for the Internal Set

We have 2M unknowns ($s_1, \ldots, s_M$; ε_θ, $c_1, \ldots, c_m$) and only M conditions.

However, if we solve for ε_θ using Cramer's rule, we may put

(Now $|X|$ represents the determinant of X).

$$\varepsilon_\theta = \frac{\begin{vmatrix} f_1 & X_{11} & X_{12} & \cdots & X_{1m} \\ f_2 & X_{21} & X_{22} & \cdots & X_{2m} \\ & & \cdots & & \\ f_M & X_{M1} & X_{M2} & \cdots & X_{Mm} \end{vmatrix}}{\begin{vmatrix} s_1 & X_{11} & X_{12} & \cdots & X_{1m} \\ s_2 & X_{21} & X_{22} & \cdots & X_{2m} \\ & & \cdots & & \\ s_M & X_{M1} & X_{M2} & \cdots & X_{Mm} \end{vmatrix}}$$

The Minimax Polynomial for the Internal Set

From the last system we may write

$$\varepsilon_\theta = \frac{\begin{vmatrix} f_1 & X_{11} & X_{12} & \cdots & X_{1m} \\ f_2 & X_{21} & X_{22} & \cdots & X_{2m} \\ & & \cdots & & \\ f_M & X_{M1} & X_{M2} & \cdots & X_{Mm} \end{vmatrix}}{s_1 \begin{vmatrix} X_{21} & \cdots & X_{2m} \\ \cdots & & \\ X_{M1} & \cdots & X_{Mm} \end{vmatrix} - s_2 \begin{vmatrix} X_{11} & \cdots & X_{1m} \\ \cdots & & \\ X_{M1} & \cdots & X_{Mm} \end{vmatrix} + \cdots + s_M \begin{vmatrix} X_{21} & \cdots & X_{2m} \\ \cdots & & \\ X_{m1} & \cdots & X_{mm} \end{vmatrix}}$$

The Minimax Polynomial for the Internal Set

Using "$X_{i'}$" to mean that the 1st column and i-th row of matrix X have been deleted we have

$$\varepsilon_\theta = \frac{\Delta}{+s_1|X_{1'}| - s_2|X_{2'}| + s_3|X_{3'}| - \ldots} \qquad (4)$$

Or,

$$\varepsilon_\theta = \frac{\Delta}{+s_1\Delta_{1'} - s_2\Delta_{2'} + s_3\Delta_{3'} - \ldots}$$

The Minimax Polynomial for the Internal Set

It is easy to see that for ε_θ to be minimized the quotient of (4) [call it D4] has to be maximized.

This implies that:

a) All its products must be maximized, and

b) All its addends must be of the same sign

This simple observation leads to an algorithm which allows us to solve (4) minimizing ε_θ and thus find the coefficients of (1) in the minimax sense.

Maximizing the Products

Since $\varepsilon_i = s_i\,\varepsilon_\theta$ it follows that $s_i \le 1$ where $s_\theta = 1$.

Then, for the $s_i \Delta_i$ to be maximized, we have to take the largest values of the s_i. Therefore, we must make

$$s_i = 1 \quad \forall i$$

In other words, for the absolute approximation errors to be minimized they must all be of the same absolute size.

The Signs of the Addends

To maximize D4 the signs of the i-th addend (σ_i) of D4 must be made to coincide with those of the products

$$(-1)^{i+1}\,\mathrm{sgn}(\Delta_{i'})$$

where sgn(x) denotes the sign of x. Thus, we must make

$$\sigma_i = (-1)^{i+1}\,\mathrm{sgn}(\Delta_{i'}) \quad \forall i$$

FINDING THE MINIMAX SIGNS

There are several ways to calculate the minimax signs. Each one varies in its computational cost. A brief discussion follows. In what follows "M" stands for the number of equations in (3).

1) DIRECT FULL (DETERMINANT) CALCULATION.

The signs of the data matrix are calculated by finding the cofactors, thus.

a) The *i*-th row and the 1st column for *i*=1 to M is selected.
b) The resulting determinant is computed
c) The appropriate sign is attached to this determinant

EXECUTION TIME 1: $O(M^4)$ -----> $\{M \times M^3\}$

COFACTOR THEOREM

If the elements of a column of a determinant are multiplied by the cofactors of a different column and summed the result is zero.

EXAMPLE:

The cofactors of the first column are denoted by K1, K2 and K3. Hence,

```
| +1 +2 +3 |      K1 = (+) | +1 +2 | = + [(1)-(-2)] = +3
| +2 +1 +2 | ==>           | -1 +1 |
| +1 -1 +1 |
                  K2 = (-) | +2 +3 | = - [(2)-(-3)] = -5
                           | -1 +1 |

                  K3 = (+) | +2 +3 | = + [(4)-(+3)] = +1
                           | +1 +2 |

Column 2:  (+3 x +2) + (-5 x +1) + (+1 x - 1) = +6 - 5 - 1 = + 6 -  6 = 0
Column 3:  (+3 x +3) + (-5 x +2) + (+1 x + 1) = +9 -10 + 1 = +10 - 10 = 0
```

2) DIRECT FAST (COFACTOR THEOREM) CALCULATION

a) The last row and first column are eliminated.
b) The corresponding determinant is calculated
c) The appropriate sign is attached.
d) The remaining cofactors are determined from the cofactor theorem.

EXECUTION TIME 2: $O(2M^3)$ ----->$\{M^3 + M^3\}$

3) INDIRECT FAST CALCULATION (TYPE 1)

The extra condition "the summation of the cofactors is equal to 1" is added to the conditions stemming from the cofactor theorem. The system of equations is solved. This is the case *Proportional to summation-equal-to-1*. Note that the value of any of the cofactors may be chosen arbitrarily. This is so because we only need the sign and, therefore, the size of the cofactor is irrelevant. On the other hand, since we are minimizing the *absolute* value of the approximation error (ε_θ) it does not matter which sign we pick for the extra condition. The signs will be all equal or all different from the actual cofactors' signs. In both cases ε_θ will be minimized.

EXECUTION TIME 3: $O(M^3)$

4) INDIRECT FAST CALCULATION (TYPE 2)

The M-th cofactor is arbitrarily set to -1. From this and the remaining conditions stemming from the cofactor theorem a set of equations is determined. The system of equations is solved. This is *Proportional to cofactor-equal to-(-1)*.

Example:

```
*
*     For M = 5
*
*           M x M                          M-1 x M
*
*     ------X-------               -------------XY-----------
*
*     s1 12 13 14 15                |12   22    32    42    52|
*     s2 22 23 24 25                |13   23    33    43    53|
*     s3 32 33 34 35          ==>   |14   24    34    44    54|
*     s4 42 43 44 45                |15   25    35    45    55|
*     s5 52 53 54 55
*
*     K5 = -1 =>
*           s5=-1
*     12 13 14 15                   |12   22    32    42| |K1| |52|
*     22 23 24 25                   |13   23    33    43| |K2| |53|
*     32 33 34 35             ==>   |14   24    34    44| |K3| |54|
*     42 43 44 45                   |15   25    35    45| |K4| |55|
```

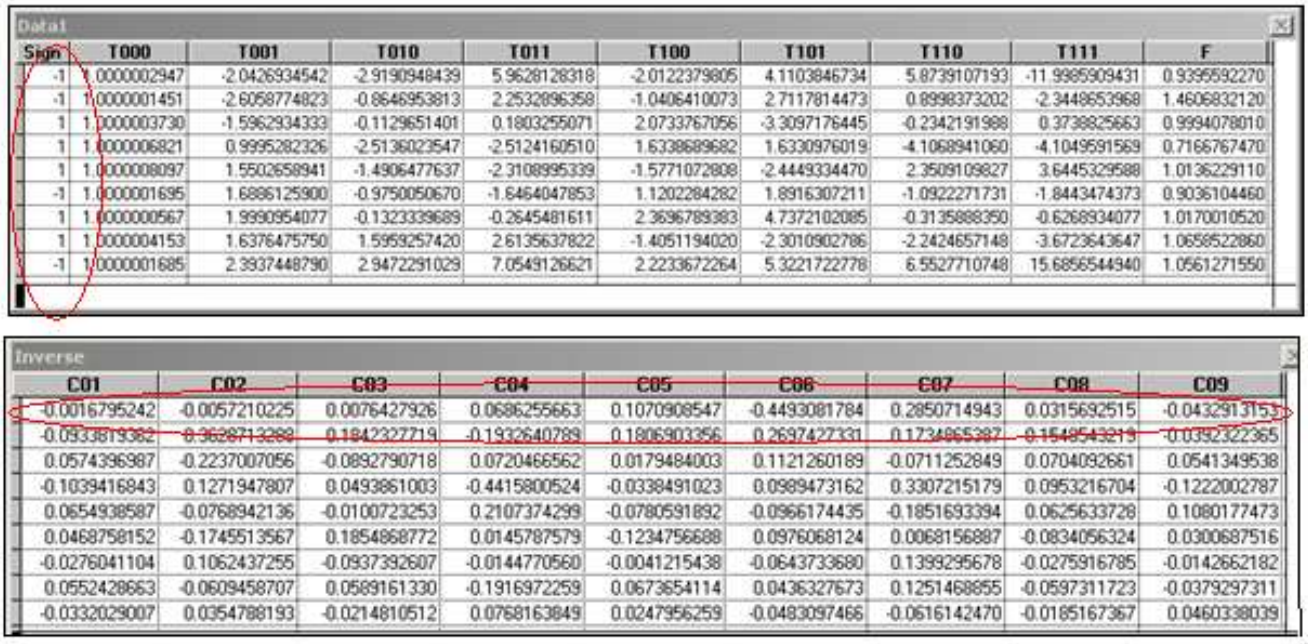
EXECUTION TIME 4: $O(M^3)$

5) INVERSE'S FIRST ROW CALCULATION

The 1st row of the inverse matrix **B** is proportional to the cofactors. This is **B1** (*from-row-1-of-**B***).

EXECUTION TIME 5: $O(M^2)$

Notice that the signs of B^1 are those of A_1.



Data1

Sign	T000	T001	T010	T011	T100	T101	T110	T111	F
-1	1.0000002947	-2.0426934542	-2.9190948439	5.9628128318	-2.0122379805	4.1103846734	5.8739107193	-11.9985909431	0.9395592270
-1	1.0000001451	-2.6058774823	-0.8646953813	2.2532896358	-1.0406410073	2.7117814473	0.8998373202	-2.3448653968	1.4606832120
1	1.0000003730	-1.5962934333	-0.1129651401	0.1803255071	2.0733767056	-3.3097176445	-0.2342191988	0.3738825663	0.9994078010
1	1.0000006821	0.9995282326	-2.5136023547	-2.5124160510	1.6338689682	1.6330976019	-4.1068941060	-4.1049591569	0.7166767470
1	1.0000008097	1.5502658941	-1.4906477637	-2.3108995339	-1.5771072808	-2.4449334470	2.3509109827	3.6445329588	1.0136229110
-1	1.0000001695	1.6886125900	-0.9750050670	-1.6464047853	1.1202284282	1.8916307211	-1.0922271731	-1.8443474373	0.9036104460
1	1.0000000567	1.9990954077	-0.1323339689	-0.2645481611	2.3696789383	4.7372102085	-0.3135888350	-0.6268934077	1.0170010520
1	1.0000004153	1.6376475750	1.5959257420	2.6135637822	-1.4051194020	-2.3010902786	-2.2424657148	-3.6723643647	1.0658522860
-1	1.0000001685	2.3937448790	2.9472291029	7.0549126621	2.2233672264	5.3221722778	6.5527710748	15.6856544940	1.0561271550

Inverse

C01	C02	C03	C04	C05	C06	C07	C08	C09
0.0016795242	-0.0057210225	0.0076427926	0.0686255663	0.1070908547	-0.4493081784	0.2850714943	0.0315692515	-0.0432913153
-0.0933819382	-0.3628713289	0.1842327719	-0.1932640789	0.1806903356	0.2697427331	0.1734965387	-0.1549543219	-0.0392322365
0.0574396987	-0.2237007056	-0.0892790718	0.0720466562	0.0179484003	0.1121260189	-0.0711252849	0.0704092661	0.0541349538
-0.1039416843	0.1271947807	0.0493861003	-0.4415800524	-0.0338491023	0.0989473162	0.3307215179	0.0953216704	-0.1222002787
0.0654938587	-0.0768942136	-0.0100723253	0.2107374299	-0.0780591892	-0.0966174435	-0.1851693394	0.0625633728	0.1080177473
0.0468758152	-0.1745513567	0.1854868772	0.0145787579	-0.1234756688	0.0976068124	0.0068156887	-0.0834056324	0.0300687516
-0.0276041104	0.1062437255	-0.0937392607	-0.0144770560	-0.0041215438	-0.0643733680	0.1399295678	-0.0275916785	-0.0142662182
0.0552428663	-0.0609458707	0.0589161330	-0.1916972259	0.0673654114	0.0436327673	0.1251468855	-0.0597311723	-0.0379297311
-0.0332029007	0.0354788193	-0.0214810512	0.0768163649	0.0247956259	-0.0483097466	-0.0616142470	-0.0185167367	0.0460338039

Take, for example, the following data matrix.

Data for Matrix M

V01	V02	V03	V04	V05	Y
3.6815653006	0.2812312330	3.0000027641	3.0000005318	2.0000008109	-10.8622927645
-2.6965476317	0.7647076107	1.0000006830	-2.0000013230	4.0000012685	-32.4712248948
2.2786657567	1.3204686828	2.0000010419	-1.0000001535	5.0000018199	5.0934600507
-3.5364790431	0.2175320614	-1.0000008250	-1.0000000321	3.0000014212	-55.3829436187
-1.0077438125	4.7913778871	-3.0000008112	-4.0000012812	2.0000001105	1.0343035761

We find the signs via *ALL* the methods mentioned.

	Finding every determinant explicitly	Finding one determinant; then using the cofactor theorem	Making the extra condition *the summation equals 1*; solving for the rest	Picking the sign of K5=-1; solving for the rest	Taking the first row of B
sgn(K1)	**-**	**-**	**+**	**-**	**-**
sgn(K2)	**-**	**-**	**+**	**-**	**-**
sgn(K3)	**+**	**+**	**-**	**+**	**+**
sgn(K3)	**-**	**-**	**+**	**-**	**-**
sgn(K3)	**-**	**-**	**+**	**-**	**-**
Factor	**1**	**1**	**-79**	**14**	**518**
Cost (FLOPS)	**M^4**	**M^3**	**M^3**	**M^3**	**M^2** *(from 2nd iteration on)*

Cofactors for M

___	Reales_d	Reales_t	Propor_s1	Propor_k5	B1
K1	-67.3887814385	-67.3887814386	0.8480302827	-4.7082953102	-0.1300257125
K2	-147.1515755317	-147.1515755320	1.8517769505	-10.2811337166	-0.2839269095
K3	151.9225908175	151.9225908178	-1.9118161047	10.6144733083	0.2931325169
K4	-2.5345216054	-2.5345216054	0.0318947906	-0.1770810502	-0.0048903240
K5	-14.3127771303	-14.3127771303	0.1801140810	-1.0000000000	-0.0276163035
Factor	1.0000000000	1.0000000000	-79.0000000000	14.0000000000	518.0000000000

"Factor" ROW

The next to the last row of "Cofactors for M" displays the proportionality factor between the different "cofactors". Only the first 2 are the real cofactors. The rest are proportional to these.

MINIMAX SIGNS AND MATRIX INVERSION

Notice that the signs in every case are either identical to the signs of the real cofactors or their exact opposites. In any case the set of signs may be taken to fulfill the minimax condition.

DATA MAPPED INTO THE POWERS OF THE INDEPENDENT VARIABLES

1.000006	1.000000	2.000014	4.000017	3.000028	6.000026	12.000028	2.000007	4.000037	8.000015	6.000040	12.000065	24.000081	-11.000102
1.000005	1.000001	3.000015	9.000053	-2.000016	-6.000004	-18.000138	1.000003	3.000022	9.000045	-2.000015	-6.000014	-18.000040	30.500043
1.000004	1.000004	1.000007	1.000003	4.000039	4.000039	4.000035	2.000018	2.000018	2.000018	8.000048	8.000024	8.000069	-21.000202
1.000004	1.000001	4.000001	16.000074	-1.000000	-4.000010	-16.000106	2.500011	10.000076	40.000059	-2.500005	-10.000066	-40.000284	80.000604
1.000002	1.000001	2.000009	4.000036	-2.000019	-4.000012	-8.000050	3.500032	7.000029	14.000020	-7.000050	-14.000128	-28.000240	53.500135
1.000003	1.000001	3.000003	9.000088	2.000016	6.000056	18.000094	3.000006	9.000033	27.000155	6.000042	18.000059	54.000229	15.500090
1.000002	1.000008	1.000007	1.000008	4.000028	4.000010	4.000036	5.000013	5.000015	5.000042	20.000199	20.000185	20.000194	-37.500074
1.000000	1.000001	1.000002	1.000001	6.000025	6.000002	6.000050	9.000031	9.000076	9.000074	54.000399	54.000135	54.000499	-101.500927
1.000008	1.000008	2.000005	4.000003	8.000066	16.000013	32.000082	2.000009	4.000026	8.000044	16.000160	32.000253	64.000008	-56.000381
1.000000	1.000001	3.000019	9.000058	4.000026	12.000005	36.000080	5.000026	15.000148	45.000408	20.000183	60.000566	180.001719	-17.500035
1.000007	1.000008	4.000007	16.000124	6.000055	24.000161	96.000060	9.000066	36.000073	144.000841	54.000253	216.001276	864.002556	-61.000238
1.000003	1.000004	-1.000005	1.000001	8.000039	-8.000016	8.000003	2.000001	-2.000008	2.000015	16.000153	-16.000075	16.000048	-17.000016
1.000008	1.000007	1.000003	1.000010	1.000000	1.000009	1.000010	1.000007	1.000002	1.000008	1.000001	1.000008	1.000009	-0.500004
1.000001	1.000003	1.000007	1.000002	2.000007	2.000020	2.000002	1.000006	1.000004	1.000009	2.000016	2.000006	2.000015	-5.500009
1.000003	1.000004	2.000007	4.000015	1.000006	2.000013	4.000028	2.000002	4.000031	8.000004	2.000014	4.000016	8.000043	7.000049
1.000009	1.000003	3.000000	9.000075	3.000009	9.000075	27.000219	-2.000018	-6.000059	-18.000064	-6.000022	-18.000141	-54.000340	-12.000053
1.000005	1.000004	4.000028	16.000013	2.000016	8.000045	32.000099	-1.000002	-4.000009	-16.000037	-2.000002	-8.000055	-32.000213	-19.000090
1.000004	1.000004	3.000014	9.000034	2.000017	6.000028	18.000016	1.500000	4.500023	13.500092	3.000011	9.000042	27.000250	5.750013
1.000010	1.000009	1.000009	1.000008	1.500008	1.500004	1.500013	2.000016	2.000010	2.000013	3.000015	3.000028	3.000016	-3.500010
1.000009	1.000002	-2.000006	4.000027	-1.500012	3.000006	-6.000055	-1.000001	2.000011	-4.000023	1.500006	-3.000015	6.000001	1.000010
1.000004	1.000003	-2.000013	4.000017	-2.000008	4.000001	-8.000025	1.250011	-2.500007	5.000022	-2.500020	5.000026	-10.000092	14.250048
1.000007	1.000004	-1.000004	1.000008	2.500013	-2.500003	2.500022	2.000000	-2.000007	2.000015	5.000026	-5.000007	5.000039	-0.500004

Minimax Signs

```
s[  0 ] =  -1.000000
s[  1 ] =   1.000000
s[  2 ] =   1.000000
s[  3 ] =  -1.000000
s[  4 ] =  -1.000000
s[  5 ] =   1.000000
s[  6 ] =  -1.000000
s[  7 ] =   1.000000
s[  8 ] =  -1.000000
s[  9 ] =  -1.000000
s[ 10 ] =   1.000000
s[ 11 ] =  -1.000000
s[ 12 ] =  -1.000000
```

MATRIX TO INVERT:

-1.000000	1.000000	2.000014	4.000017	3.000028	6.000026	12.000028	2.000007	4.000037	8.000015	6.000040	12.000065	24.000081
1.000000	1.000001	3.000015	9.000053	-2.000016	-6.000004	-18.000138	1.000003	3.000022	9.000045	-2.000015	-6.000014	-18.000040
1.000000	1.000004	1.000007	1.000003	4.000039	4.000039	4.000035	2.000010	2.000018	2.000018	8.000048	8.000024	8.000069
-1.000000	1.000001	4.000001	16.000074	-1.000000	-4.000010	-16.000106	2.500011	10.000076	40.000059	-2.500005	-10.000066	-40.000284
-1.000000	1.000001	2.000009	4.000036	-2.000019	-4.000012	-8.000050	3.500032	7.000029	14.000020	-7.000050	-14.000128	-28.000240
1.000000	1.000001	3.000003	9.000008	2.000016	6.000056	18.000094	3.000006	9.000033	27.000155	6.000042	18.000059	54.000229
-1.000000	1.000008	1.000007	1.000008	4.000028	4.000010	4.000036	5.000013	5.000015	5.000042	20.000199	20.000185	20.000194
1.000000	1.000001	1.000002	1.000001	6.000025	6.000002	6.000050	9.000031	9.000076	9.000074	54.000199	54.000135	54.000499
-1.000000	1.000008	2.000005	4.000003	8.000066	16.000013	32.000082	2.000009	4.000026	8.000044	16.000160	32.000253	64.000008
-1.000000	1.000001	3.000019	9.000058	4.000026	12.000005	36.000080	5.000026	15.000148	45.000408	20.000183	60.000566	180.001719
1.000000	1.000008	4.000007	16.000124	6.000055	24.000161	96.000060	9.000066	36.000073	144.000841	54.000253	216.001276	864.002556
-1.000000	1.000004	-1.000005	1.000001	8.000039	-8.000016	8.000003	2.000001	-2.000008	2.000015	16.000153	-16.000075	16.000048
-1.000000	1.000007	1.000003	1.000010	1.000000	1.000009	1.000010	1.000007	1.000002	1.000008	1.000001	1.000008	1.000009

INVERTED MATRIX:

-0.198940	0.014719	0.224461	-0.054682	-0.047859	0.259473	-0.011327	0.000860	-0.059664	-0.057654	0.000487	-0.017987	-0.051887
-11.973744	1.452712	-1.509030	-0.299098	2.738469	0.035629	-1.391232	0.589398	5.608434	0.436424	-0.068740	0.059510	5.321205
16.004581	-2.132877	2.423945	0.409599	-3.641952	-0.011003	0.578736	-0.371578	-7.676846	-0.552954	0.091553	-0.102113	-5.019010
-4.130117	0.687500	-0.802646	-0.137836	0.879497	0.105105	0.056848	0.032601	2.038478	0.087712	-0.022571	0.033609	1.171798
1.696568	-0.126861	0.636378	0.038354	-0.466460	-0.281571	0.224025	-0.191882	-0.911899	0.284803	-0.036059	0.038931	-0.904318
-2.024905	0.188884	-0.415197	0.014074	0.679035	0.011163	-0.279524	0.167647	1.251964	-0.280277	0.047497	-0.034842	0.674471
0.469210	-0.072443	0.036480	-0.013696	-0.178667	0.086620	0.084355	-0.038873	-0.297784	0.036312	-0.011783	0.008652	-0.108380
3.124823	-0.534262	0.094708	0.022516	-1.480326	1.058558	2.076350	-0.654508	-2.112960	-0.457422	0.035508	-0.080309	-1.092667
-4.711601	0.822874	-0.431534	-0.179816	1.842764	-0.823466	-1.422590	0.478123	2.770947	0.460721	-0.046005	0.072429	1.167138
1.310361	-0.268153	0.204114	0.081350	-0.428858	0.125276	0.191593	-0.072410	-0.740779	-0.083381	0.011176	-0.017101	-0.313182
-0.424899	0.034563	-0.129199	0.014740	0.262897	-0.079348	-0.341627	0.156256	0.368382	-0.093481	0.017726	0.013620	0.200367
0.640142	-0.066884	0.156286	0.000568	-0.332853	0.047137	0.253256	-0.112722	-0.480842	0.128419	-0.023816	-0.012972	-0.195716
-0.179299	0.029660	-0.039832	-0.005433	0.078586	-0.014636	-0.041522	0.018807	0.123216	-0.024527	0.006001	0.002599	0.046379

PRODUCT OF MATRIX BY ITS INVERTED

1.000000	-0.000000	-0.000000	0.000000	0.000000	0.000000	-0.000000	-0.000000	-0.000000	0.000000	-0.000000	-0.000000	0.000000
-0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	-0.000000	-0.000000	0.000000	0.000000	-0.000000	-0.000000
-0.000000	0.000000	1.000000	-0.000000	0.000000	-0.000000	-0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.000000	-0.000000	0.000000	1.000000	-0.000000	0.000000	0.000000	0.000000	-0.000000	-0.000000	0.000000	0.000000	-0.000000
0.000000	-0.000000	0.000000	0.000000	1.000000	0.000000	0.000000	-0.000000	-0.000000	0.000000	0.000000	-0.000000	-0.000000
-0.000000	0.000000	-0.000000	-0.000000	0.000000	1.000000	-0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
-0.000000	0.000000	0.000000	-0.000000	-0.000000	-0.000000	1.000000	-0.000000	0.000000	0.000000	0.000000	-0.000000	-0.000000
-0.000000	0.000000	0.000000	-0.000000	0.000000	0.000000	-0.000000	1.000000	0.000000	0.000000	0.000000	-0.000000	-0.000000
-0.000000	0.000000	-0.000000	-0.000000	0.000000	-0.000000	-0.000000	0.000000	1.000000	-0.000000	-0.000000	0.000000	0.000000
0.000000	0.000000	-0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000
-0.000000	0.000000	0.000000	-0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	-0.000000
-0.000000	0.000000	0.000000	-0.000000	0.000000	-0.000000	-0.000000	0.000000	0.000000	0.000000	-0.000000	1.000000	0.000000
-0.000000	-0.000000	-0.000000	0.000000	-0.000000	-0.000000	-0.000000	-0.000000	-0.000000	0.000000	0.000000	-0.000000	1.000000

THE (NAÏVE) EXCHANGE ALGORITHM

The Signs of the Addends

Once the signs of the Δ_i' are known, we may determine the σ_i and, since their absolute value is "1", we may easily solve

$$\begin{bmatrix} \sigma_1 & X_{11} & X_{12} & \dots & X_{1m} \\ \sigma_2 & X_{21} & X_{22} & \dots & X_{2m} \\ & & \dots & & \\ \sigma_M & X_{M1} & X_{M2} & \dots & X_{Mm} \end{bmatrix} \begin{bmatrix} \varepsilon_\theta \\ c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ \dots \\ f_M \end{bmatrix} \qquad (3)$$

Cost of finding ε_θ

Solving system (3) also costs $O(m^3)$. Therefore, the cost of finding ε_θ is m^4+m^3 flops.

$$C(\varepsilon_\theta)=O(m^4)$$

This is just the starting point.

Now we must evaluate the y_i's for the remaining M-N vectors in the outer set.

Cost of finding ε_Φ

The evaluation of every

$$y = c_1 X_1 + c_2 X_2 + \ldots + c_m X_m$$

Clearly requires m flops. The full cost of evaluating all M-N vectors comes out to be, therefore, $(M\text{-}N)(m^3)$ flops. Since $M=m+1$

$$C(\varepsilon_\Phi) = m^4\text{-}Nm^3$$

and the calculation of both ε_θ and ε_Φ costs

$$C(\varepsilon_\theta+\varepsilon_\Phi) = 2m^4\text{-}Nm^3$$

$$C(\varepsilon\theta+\varepsilon\Phi) = O(m^4)$$

The Naive Exchange Algorithm

We may now advance the first version of an algorithm which effectively solves the problem of finding the coefficients of system (1).

$$y = c_1 X_1 + c_2 X_2 + \ldots + c_m X_m$$

It relies on the possibility of explicitly finding the signs of the Δ_i's and, accordingly, minimizing the largest possible error for any subset of size M out of the original N objects of the sample.

We call it *naive* because although effective it is not efficient and will be improved considerably.

This does not deter the fact that it will solve any system with the form of (1).

EXAMPLE OF COST FOR THE (NAÏVE) EXCHANGE ALGORITHM

Assume a problem with 5 independent variables, i.e. $F(\vec{V}) = f(x_1, x_2, x_3, x_4, x_5)$ and, furthermore, that we wish to model $F(\vec{V})$ with a linear combination of the 5 independent variables as shown next

$$F(\vec{V}) = \sum_{i1=0}^{1}\sum_{i2=0}^{2}\sum_{i3=0}^{2}\sum_{i4=0}^{1}\sum_{i5=0}^{2} C_{i_1 i_2 i_3 i_4 i_5} x_1^{i_1} x_2^{i_2} x_3^{i_3} x_4^{i_4} x_5^{i_5} \qquad (5)$$

mapping the independent variables into de model of (5) yields 2x3x3x2x3+1 = 109 terms (or *m=109*). Hence, the minimax solution of the approximation polynomial requires $O(109^4) \approx 140$ million floating point operations.

The Exchange Algorithm (1)

a) Set t←1

b) Select M vectors

c) Calculate σ[t, i] for i=1,…,M

d) Solve system (3) thus finding $\varepsilon_\theta(t)$ and $c_i(t)$

e) Find $\varepsilon_\Phi(t)$ (which corresponds to the k-th vector).

f) If $\varepsilon_\theta(t) \geq \varepsilon_\Phi(t)$ END. The c_i's are the coefficients of the best minimax approximating polynomial

g) t ← t+1

The Exchange Algorithm (2)

h) For i=1 to M

i) Replace vector i with vector k

j) Calculate σ[t, j] for j=1,2,…,M

k) Calculate ε_θ[t, i]

l) ε_θ[t, i] > ε_θ[t-1, i] ⟹ (d)

m) Restore vector i

n) endfor

THE FAST ASCENT ALGORITHM

a) Map data to powers of the variables for the indicated degrees.
b) Transform the data via random perturbations to stabilize the behavior of the matrix.
c) Obtain the signs of the errors to achieve minimax (M^3). The completed matrix is **A**.

MATRIX A

Data1

Sign	T000	T001	T010	T011	T100	T101	T110	T111	F
-1	1.0000002947	-2.0426934542	-2.9190948439	5.9628128318	-2.0122379805	4.1103846734	5.8739107193	-11.9985909431	0.9395592270
-1	1.0000001451	-2.6058774823	-0.8646953813	2.2532896358	-1.0406410073	2.7117814473	0.8998373202	-2.3448653968	1.4606832120
1	1.0000003730	-1.5962934333	-0.1129651401	0.1803255071	2.0733767056	-3.3097176445	-0.2342191988	0.3738825663	0.9994078010
1	1.0000006821	0.9995282326	-2.5136023547	-2.5124160510	1.6338689682	1.6330976019	-4.1068941060	-4.1049591569	0.7166767470
1	1.0000008097	1.5502658941	-1.4906477637	-2.3108995339	-1.5771072808	-2.4449334470	2.3509109827	3.6445329588	1.0136229110
-1	1.0000001695	1.6886125900	-0.9750050670	-1.6464047853	1.1202284282	1.8916307211	-1.0922271731	-1.8443474373	0.9036104460
1	1.0000000567	1.9990954077	-0.1323339689	-0.2645481611	2.3696789383	4.7372102085	-0.3135888350	-0.6268934077	1.0170010520
1	1.0000004153	1.6376475750	1.5959257420	2.6135637822	-1.4051194020	-2.3010902786	-2.2424657148	-3.6723643647	1.0658522860
-1	1.0000001685	2.3937448790	2.9472291029	7.0549126621	2.2233672264	5.3221722778	6.5527710748	15.6856544940	1.0561271550

d) Obtain **B**, the inverse of matrix **A**.

MATRIX B = A^{-1}

Inverse

C01	C02	C03	C04	C05	C06	C07	C08	C09
-0.0016795242	-0.0057210225	0.0076427926	0.0686255663	0.1070908547	-0.4493081784	0.2850714943	0.0315692515	-0.0432913153
-0.0933819362	0.3628713288	0.1842327719	-0.1932640789	0.1806903356	0.2697427331	0.1734865387	0.1548543219	-0.0392322365
0.0574396987	-0.2237007056	-0.0892790718	0.0720466562	0.0179484003	0.1121260189	-0.0711252849	0.0704092661	0.0541349538
-0.1039416843	0.1271947807	0.0493861003	-0.4415800524	-0.0338491023	0.0989473162	0.3307215179	0.0953216704	-0.1222002787
0.0654938587	-0.0768942136	-0.0100723253	0.2107374299	-0.0780591892	-0.0966174435	-0.1851693394	0.0625633728	0.1080177473
0.0468758152	-0.1745513567	0.1854868772	0.0145787579	-0.1234756688	0.0976068124	0.0068156887	-0.0834056324	0.0300687516
-0.0276041104	0.1062437255	-0.0937392607	-0.0144770560	-0.0041215438	-0.0643733680	0.1399295678	-0.0275916785	-0.0142662182
0.0552428663	-0.0609458707	0.0589161330	-0.1916972259	0.0673654114	0.0436327673	0.1251468855	-0.0597311723	-0.0379297311
-0.0332029007	0.0354788193	-0.0214810512	0.0768163849	0.0247956259	-0.0483097466	-0.0616142470	-0.0185167367	0.0460338039

THE INVERSE OF A MATRIX

The calculation of the inverse of a matrix may be accomplished from a variation of the Gauss-Jordan method. The cost is O(M^3). The pseudo-code of the algorithm is as follows. In Python we may take advantage of the numpy method "`np.linalg.inv(a)`".

```
#
#        PYTHON INVERT AN MXM MATRIX
#    ON INPUT:
#        "map" ---> MXM row of rows MATRIX TO INVERT
#        M=TUPLES/ATRIBS OF "A"
#        "b"    ---> MXM INVERTED MATRIX

def InvertMatrix():
    import numpy as np
    from numpy import empty
    a = empty([M+1,M+1])
    for i in range(0,M+1):
        for j in range(0,M+1):
            a[i][j]=map[i+1][j+1]
        #endfor
    #endfor
    ai=np.linalg.inv(a)
    for i in range(0,M+1):
        for j in range(0,M+1):
            b[i+1][j+1]=ai[i][j]
        #endfor
    #endfor
    return
#endInvertMatrix
```

```
function Inverse
*
*     On Input:
*         X ==> Original data matrix
*
*     On output:
*         Ix==> Inverted data matrix
*
*
* Scale the rows of the original matrix
priv i,j,k,l,rowmax,scale,big,temp,ipiv,quot
for i=1 to M
      rowmax=abs(X(i,1))
      for j=2 to M
         rowmax=max(rowmax,abs(X(i,j)))
      endfor
      if rowmax=0
         return .F.
      endif
      scale=1/rowmax
*
*         Put original scaled matrix in X  [X  <-- X scaled]
*         Put identity scaled matrix in Ix [Ix <-- I scaled]
*
      for j=1 to M
         X(i,j)=X(i,j)*scale
         if i=j
            Ix(i,j)=scale
         else
            Ix(i,j)=0
         endif
      endfor
endfor
*
*     Put largest element of column in pivot position
*
for k=1 to M-1
      big=0
      for i=k to M
            temp=abs(X(i,k))
            if big<temp
               big=temp
               ipiv=i
           endif
      endfor
      if big=0
            return .F.
      endif
      if ipiv!=k
            for i=k to M
               temp=X(k,i)
               X(k,i)=X(ipiv,i)
               X(ipiv,i)=temp
            endfor
            for i=1 to M
               temp=Ix(k,i)
```

```
                Ix(k,i)=Ix(ipiv,i)
                Ix(ipiv,i)=temp
             endfor
          endif
*
*       Eliminate X(k) from equations k+1, k+2, ..., k+M
*
       for i=k+1 to M
             quot=X(i,k)/X(k,k)
             for j=k+1 to M
                X(i,j)=X(i,j)-quot*X(k,j)
             endfor
             for j=1 to M
                Ix(i,j)=Ix(i,j)-quot*Ix(k,j)
             endfor
       endfor
endfor
if X(M,M)=0
   return .F.
endif
*
*      Back substitution
*
for i=1 to M
   Ix(M,i)=Ix(M,i)/X(M,M)
   for j=M-1 to 1 step -1
      sum=0
      for k=j+1 to M
         sum=sum+X(j,k)*Ix(k,i)
      endfor
      Ix(j,i)=(Ix(j,i)-sum)/X(j,j)
   endfor
endfor
return .T.
```

PRODUCT A x B = I

Ident1

I01	I02	I03	I04	I05	I06	I07	I08	I09
1.0000000000	0.0000000000	0.0000000000	0.0000000000	0.0000000000	0.0000000000	0.0000000000	0.0000000000	0.0000000000
0.0000000000	1.0000000000	0.0000000000	0.0000000000	0.0000000000	0.0000000000	0.0000000000	0.0000000000	0.0000000000
0.0000000000	0.0000000000	1.0000000000	0.0000000000	0.0000000000	0.0000000000	0.0000000000	0.0000000000	0.0000000000
0.0000000000	0.0000000000	0.0000000000	1.0000000000	0.0000000000	0.0000000000	0.0000000000	0.0000000000	0.0000000000
0.0000000000	0.0000000000	0.0000000000	0.0000000000	1.0000000000	0.0000000000	0.0000000000	0.0000000000	0.0000000000
0.0000000000	0.0000000000	0.0000000000	0.0000000000	0.0000000000	1.0000000000	0.0000000000	0.0000000000	0.0000000000
0.0000000000	0.0000000000	0.0000000000	0.0000000000	0.0000000000	0.0000000000	1.0000000000	0.0000000000	0.0000000000
0.0000000000	0.0000000000	0.0000000000	0.0000000000	0.0000000000	0.0000000000	0.0000000000	1.0000000000	0.0000000000
0.0000000000	0.0000000000	0.0000000000	0.0000000000	0.0000000000	0.0000000000	0.0000000000	0.0000000000	1.0000000000

e) Calculate the vector of coefficients **C** from the inverse and the dependent variable [O($\mathbf{M^2}$)]

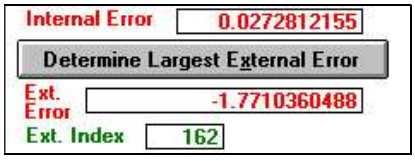
$$\mathbf{C = F\,B}$$

ε_θ	=	0.9395592270	-0.0016795242	-0.0057210225	0.0076427926	0.0686255663	0.1070908547	-0.4493081784	0.2850714943	0.0315692515	-0.0432913153
C000		1.4606832120	-0.0933819362	0.3628713288	0.1842327719	-0.1932640789	0.1806903356	0.2697427331	0.1734865387	0.1548543219	-0.0392322365
C001		0.9994078010	0.0574396987	-0.2237007056	-0.0892790718	0.0720466562	0.0179484003	0.1121260189	-0.0711252849	0.0704092661	0.0541349538
C010		0.7166767470	-0.1039416843	0.1271947807	0.0493861003	-0.4415800524	-0.0338491023	0.0989473162	0.3307215179	0.0953216704	-0.1222002787
C011		1.0136229110	0.0654938587	-0.0768942136	-0.0100723253	0.2107374299	-0.0780591892	-0.0966174435	-0.1851693394	0.0625633728	0.1080177473
C100		0.9036104460	0.0468758152	-0.1745513567	0.1854868772	0.0145787579	-0.1234756688	0.0976068124	0.0068156887	-0.0834056324	0.0300687516
C101		1.0170010520	-0.0276041104	0.1062437255	-0.0937392607	-0.0144770560	-0.0041215438	-0.0643733680	0.1399295678	-0.0275916785	-0.0142662182
C110		1.0658522860	0.0552428663	-0.0609458707	0.0589161330	-0.1916972259	0.0673654114	0.0436327673	0.1251468855	-0.0597311723	-0.0379297311
C111		1.0561271550	-0.0332029007	0.0354788193	-0.0214810512	0.0768163849	0.0247956259	-0.0483097466	-0.0616142470	-0.0185167367	0.0460338039

Coefficients

Name	Value
Error	0.0272812155
C000	1.2148658067
C001	-0.1309839697
C010	0.1850021329
C011	-0.0837992090
C100	-0.1022656708
C101	0.0606803964
C110	0.0156390707
C111	0.0019113237

f) Calculate $P_n(\mathbf{X})$ for all the objects in the external set with **C**. Calculate the largest absolute external error and its index O[**(N-M)xM**].



g) Determine the internal point which will make ε_θ to increase.

Efficient swapping

We denote the i-th row of A with A^i; its j-th column with A_j. From the cofactor theorem:

$$\sum_{i=0}^{m} K_i A^i = 0$$

$$A^j = \sum_{\substack{i=1 \\ i\neq j}}^{m} -\frac{K_i}{K_j} A^i \qquad (5.1)$$

Since, by definition, the rows are linearly independent:

$$A^{m+1} = \sum_{i=1}^{m} \lambda_i A^i$$

$$A^{m+1} - \sum_{i=1}^{m} \lambda_i A^i = 0 \tag{5.2}$$

$$A^{m+1} - \lambda_j A^j - \sum_{\substack{i=1 \\ i \neq j}}^{m} \lambda_i A^i = 0$$

Putting (5.1) in (5.2):

$$A^{m+1} - \lambda_j \left(\sum_{\substack{i=1 \\ i \neq j}}^{m} - \frac{K_i}{K_j} A^i \right) - \sum_{\substack{i=1 \\ i \neq j}}^{m} \lambda_i A^i = 0$$

$$A^{m+1} + \left(\sum_{\substack{i=1 \\ i \neq j}}^{m} \frac{\lambda_j K_i}{K_j} - \lambda_i \right) A^i = 0$$

We require to select *j* thus:

$$\frac{\lambda_j K_i}{K_j} - \lambda_i \geq 0 \tag{5.3}$$

so that **0 is a nonnegative linear combination of $\mathbf{A^1}$, ..., $\mathbf{A^{m+1}}$ with $\mathbf{A^j}$ not appearing**. This is equivalent to

$$\frac{\lambda_j}{K_j} \geq \frac{\lambda_i}{K_i} \tag{5.4}$$

Notice, FIRST, that

$$\boldsymbol{\lambda}\,\mathbf{A} = A^{m+1}$$

so that the λ's are given by

$$\boldsymbol{\lambda} = A^{m+1}\mathbf{B} \tag{5.5}$$

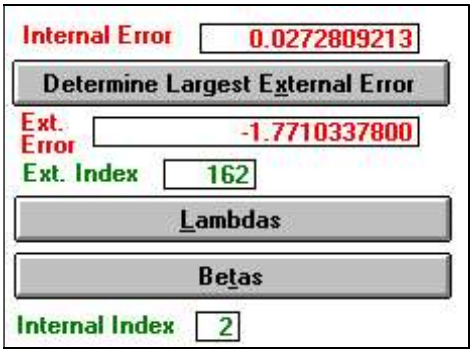
g.1) Calculate the λ's from (5.5). O($\mathbf{M^2}$)

Lambdas

Index	Value
1	-0.7078538304
2	1.4953690192
3	0.4119733527
4	-1.9163564224
5	0.2935209467
6	0.9338514117
7	1.3179471317
8	-0.1070845885
9	-0.7213661803

SECOND, since B = A^{-1} it follows that $B^1 \times A_1 = 1$; $B^1 \times A_2 = 0$; ... ; $B^1 \times A_m = 0$. But this is the set of equations of the cofactor theorem. In other words, B^1 IS the row of cofactors! The first condition corresponds to the free condition ("the summation of the cofactors is equal to 1") established above. Therefore, the ratios needed to determine the j-th element to exchange are those corresponding to the largest $\frac{\lambda_j}{B_j^1}$. Actually we seek for the largest $\sigma_\phi \frac{\lambda_j}{B_j^1}$ so that the sign of the error of the external vector (A^ϕ) is kept.

```
procedure getInternal
* Put external vector as a function of internal vectors
select index(Phi)
Mu=sign(Epsilon_Phi)
copy to array A
for i=1 to M+1
      Sum=Mu*B(1,i)
      for j=2 to M+1
            Sum=Sum+A(j-1)*B(j,i)
      endfor
      Lambda(i)=Sum
endfor
* Determine which internal vector to swap
betaMax=-infinity
for i=1 to M+1
      quot=Mu*Lambda(i)/B(1,i)
      if quot>betaMax
            betaMax=quot
            index(Theta)=i
      endif
endfor
return
```



g.2) Cost of the calculation of M quotients ($\beta = \sigma_\phi \frac{\lambda_j}{B_j^1}$) → O(**M**).

THIRD, Let a) **A** be a nonsingular matrix, b) Let **B** be its inverse and $B_1,...,B_m$ its columns; c) Let $\overline{A}$ be the matrix obtained by replacing the β-th row of **A** by a vector **v**. If $\lambda_\beta \equiv < v, B_\beta > \neq 0$ then $\overline{A}$ is nonsingular and the columns of its inverse are given by $\overline{B}_\beta = \frac{B_\beta}{\lambda_\beta}$ and $\overline{B} = B_j - < v, B_j > \overline{B}_\beta$ for $j \neq \beta$.

This result is very interesting in that the second and all subsequent instances of **B** may be obtained in $O(M^2)$ floating point operations. Since the **λ** and **C** matrices may be obtained from **B** its efficient calculation is very convenient. In fact, every cycle of the exchange algorithm may now be seen to have a cost $O(M^2)$. This represents a remarkable speedup factor of two orders of magnitude vs. the naïve exchange algorithm.

Proof.

To verify that $\overline{A}\overline{B} = I$ we compute the inner product of $\overline{A}^i$ with $\overline{B}_j$. There are four cases.

Case 1. $i = \beta$ *and* $j = \beta$; then

$$\begin{aligned} \langle \overline{A}^\beta . \overline{B}_\beta \rangle &= \langle v, \lambda^{-1} B_\beta \rangle \\ &= \lambda^{-1} \langle v, B_\beta \rangle \\ &= 1 \end{aligned}$$

Case 2. $i \neq \beta$ *and* $j = \beta$; then

$$\begin{aligned} \langle \overline{A}^i . \overline{B}_\beta \rangle &= \langle A^i, \lambda^{-1} B_\beta \rangle \\ &= \lambda^{-1} \langle A^i, B_\beta \rangle \\ &= 0 \end{aligned}$$

Case 3. $i = \beta$ *and* $j \neq \beta$; then

$$\begin{aligned} \langle \overline{A}^\beta . \overline{B}_j \rangle &= \langle v, B_j - \langle v, B_j \rangle \overline{B_\beta} \rangle \\ &= \langle v, B_j \rangle - \langle v, B_j \rangle \lambda^{-1} \langle v, B_\beta \rangle \\ &= 0 \end{aligned}$$

Case 4. $i \neq \beta$ *and* $j \neq \beta$; then

$$\begin{aligned}\langle \overline{A}^{\prime}.\overline{B}_j \rangle &= \langle A^{\prime}.B_j - \langle v.B_j \rangle \overline{B_\beta} \rangle \\ &= \langle A^{\prime}.B_j \rangle - \langle v.B_j \rangle \lambda^{-1} \langle A^{\prime}.B_\beta \rangle \\ &= \langle A^{\prime}.B_j \rangle \\ &= \delta_{ij}\end{aligned}$$

```
procedure NextInverse
for i=1 to M+1
      B(i,index(Theta))=B(i,index(Theta))/Lambda(index(Theta))
endfor
for i=1 to M+1
      for j=1 to Mp1
            if i=index(Theta)
                  exit
            endif
            B(j,i)=B(j,i)-Lambda(i)*B(j,index(Theta))
      endfor
endfor
return
```

The second instance of **B** is shown next. One corollary of the procedure is that we immediately obtain the signs of the errors from the first row of **B**. These signs will not be explicitly needed any more, however.

Newinv

C01	C02	C03	C04	C05	C06	C07	C08	C09
-0.0043877091	-0.0038259678	0.0092191102	0.0612935463	0.1082137538	-0.4457352248	0.2901138694	0.0311595701	-0.0460512486
0.0783882980	0.2426634124	0.0842619414	0.2717655093	0.1094635827	0.0431311144	-0.1463310031	0.1808397778	0.1358169831
-0.0484521695	-0.1495957185	-0.0276496911	-0.2146318860	0.0618578665	0.2518261643	0.1260338411	0.0543898534	-0.0537782924
-0.0437323177	0.0850589728	0.0143440406	-0.2785764480	-0.0588157951	0.0195148255	0.2186180004	0.1044301693	-0.0608415132
0.0290949113	-0.0514215324	0.0111119769	0.1121954720	-0.0629658800	-0.0485974147	-0.1173984934	0.0570569042	0.0709240155
-0.0357505515	-0.1167280733	0.2335757444	-0.2091136991	-0.0892136014	0.2066135416	0.1606570097	-0.0959054344	-0.0541348978
0.0226878294	0.0710484915	-0.1230093400	0.1216771261	-0.0249757678	-0.1307221010	0.0462914340	-0.0199834776	0.0369857547
0.0263932670	-0.0407564860	0.0757066989	-0.2698010175	0.0793282257	0.0816932697	0.1788616523	-0.0640955426	-0.0673300339
-0.0164084915	0.0237258719	-0.0312555029	0.1222835546	0.0178316324	-0.0704662122	-0.0928836119	-0.0159760742	0.0631488299

The Fast Ascent Algorithm

1. Input the data vectors (call them **D**).
2. Input the degrees of each of the variables of the approximating polynomial.
3. Map the original data vectors into the powers of the monomials (call them **P**).
4. Stabilize the vectors of **P** by randomly disturbing the original values as above (call the resulting data **S**).
5. Select a subset of size M from **S**. Call it **I**. Call the remaining vectors **E**.

BOOTSTRAP $O(M^3)$

6. Obtain the minimax signs (call the matrix incorporating the σ's **A**). **M^2**
7. Obtain the inverse of **A** (call it **B**). **M^3**

LOOP $O(M^2)$

8. Calculate the coefficients **C = f B**. The maximum internal ε_θ error is also calculated. **M^2**
9. Calculate the maximum external error ε_ϕ from **C** and **E**. Call its index I_E **$MN-M^2$**
10. $\varepsilon_\theta \geq \varepsilon_\phi$?

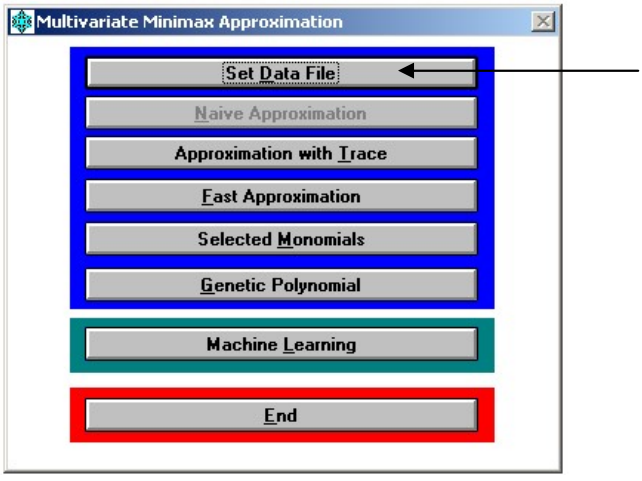
    YES: STOP; the coefficients of **C** are those of the minimax polynomial for the **D** vectors.
11. Calculate the **λ** vector from $\boldsymbol{\lambda} = \mathbf{A}^{m+1}\mathbf{B}$. **$M^2$**
12. Calculate the vector **β** which maximizes $\sigma_\phi \frac{\lambda_j}{B_j}$. Call its index I_I. **M**
13. Interchange vectors I_E and I_I.
14. Calculate the new inverse $\overline{\mathbf{B}}$. Make **B** ← $\overline{\mathbf{B}}$. **M^2**
15. Go to step 8.

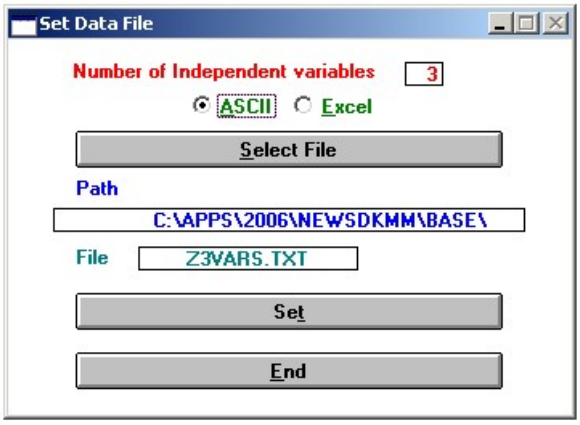
This algorithm costs $O(4M^2)$ FLOPS in every iteration. It is a remarkable improvement over the naïve exchange algorithm. It may be further improved using heuristic techniques.

FULL EXAMPLE

The following example was taken from the program ***NEWSDKMM.EXE***. It runs best on Windows XP.

1. Input the data vectors (call them **D**).

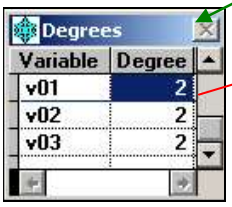




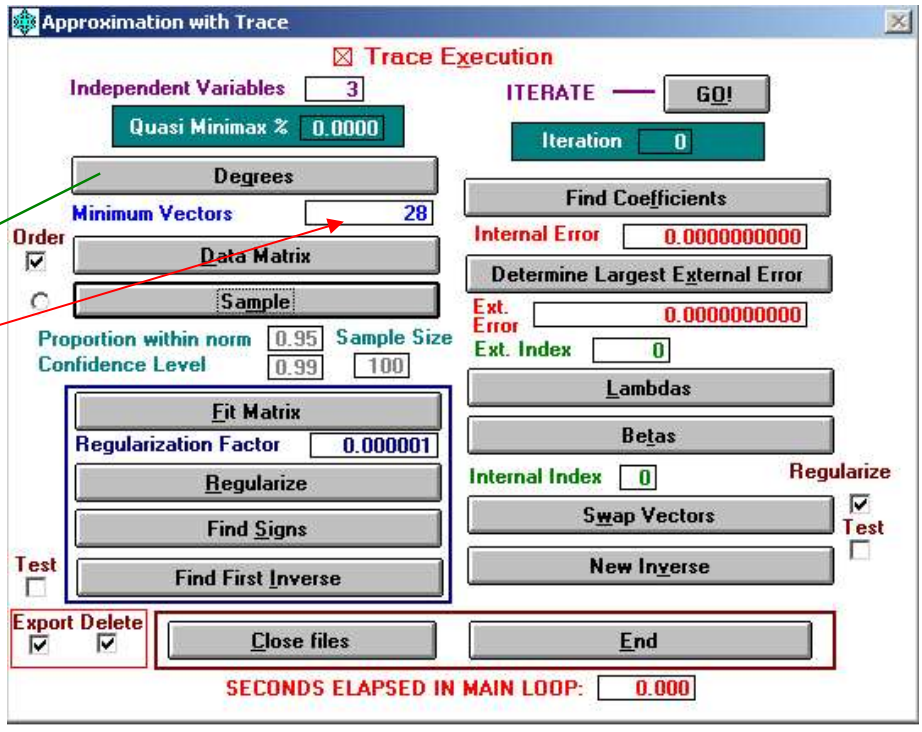
Data File

V01	V02	V03	F
1.4111970760	0.6344440890	-1.8566366910	0.8817407070
1.6939553160	-0.3968748860	1.2390210770	0.8962512570
-0.8177116800	-1.4784499750	-2.4402897540	0.3379039350
2.3101339540	-2.9064715600	1.4107567630	1.0114966370
-2.1201298040	-0.5268444960	-1.1892259540	0.9169912640
0.9752722710	-2.5830607740	-0.8718351180	1.1592536870
0.2168954760	1.3984325580	1.0280978450	1.4975148000
-1.6150987990	-2.6173383990	1.2741184140	1.1242601520
-2.6609439140	-1.8622170130	0.3875441500	1.6419683010
-1.7999534200	-0.0745716210	0.4346738710	1.0045321540
2.8365544370	1.0690850650	0.9000680780	1.3925816520
-1.5405914100	-2.8802782300	0.7523868700	1.1189463380
1.9804730000	-0.5067041680	-1.9445359420	1.0917083620
-1.3097231320	2.4770853030	-1.0736765790	1.0065664620
-2.8640745120	1.1921759700	-1.2772764520	0.9169812640
-0.5539860640	2.5708253580	-2.3718380970	1.1194886140

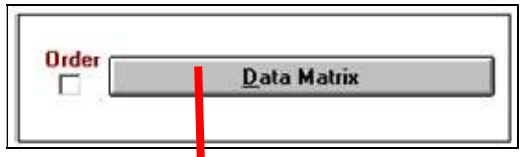
2. Input the degrees of each of the variables of the approximating polynomial



The number of minimum vectors is (d1+1)x(d2+1)x(d3+1)+1 = 3x3x3+1=28



3a. Select M vectors from **D.**



Data0

V01	V02	V03	F
-2.0122377610	-2.9190930940	-2.0426922110	0.9395592270
-2.3426333410	-1.7340805940	-2.2761033090	1.5887017490
-0.9118145360	-1.6625974280	-2.7702046410	1.8966292780
1.3113541940	-1.9225000260	-1.7185578460	1.1374131420
0.3503251080	-2.1891529700	-0.5963667280	0.4790972420
2.5765072060	-2.8547680380	-0.7989707210	1.1770543750
0.6126465590	0.0698140360	-2.3992114160	0.9896233860
2.2063982430	-1.6948918910	0.0312532770	0.5688993630
0.0866813020	0.5650290970	-1.7229590960	0.9809768160
0.8655813480	0.1423767750	-0.8009777400	1.0216220230
-2.2006053730	1.1225983110	-1.9785631370	1.3380020930
-2.1281439430	-1.2337051950	1.2349662860	0.7132836120
2.8457172560	1.0571296570	-2.3977621670	1.2063847890
-1.5982122170	1.2257823730	-0.7645724650	0.8034134300
2.2360279430	1.2983866060	-2.2169872650	1.2224590670
2.9220135490	0.3630220780	-0.1818904340	1.2758104860
-2.3420797740	-2.2975020100	2.1195794680	0.7991904390
-0.0081193170	-0.4718530640	1.4685596090	0.9828817940
-1.6802832620	-0.7939539440	2.0848235160	1.3472481510
-1.7882218860	1.1408243550	1.1305308840	0.9744992850
2.9546131120	2.5233382810	-1.9627648250	0.7921241900
-1.8155557000	2.6954335410	-0.7010572770	1.0396839870
-1.6463721850	2.2467282590	0.2892868000	1.3480518420
1.5763896890	2.9910980740	-1.7509104540	0.9843366400
0.5418244630	0.5260151330	2.3453084070	0.7835607300
1.1669944320	2.3202832250	0.8546888330	1.4695171420
-0.2191561080	2.9358291540	1.1313015870	1.0452031490
2.2233661810	2.9472279450	2.3937435500	1.0561271550

3b. Map the original data vectors into the powers of the monomials (call them **P**).

Sign	T000	T001	T002	T010	T220	T221	T222	F
	1.0000000000	-2.0426922110	4.1725914689	-2.9190930940	34.5028110711	-70.4786234325	143.9661351275	0.9395592270
	1.0000000000	-2.2761033090	5.1806462732	-1.7340805940	16.5024032850	-37.5611747235	85.4931140782	1.5887017490
	1.0000000000	-2.7702046410	7.6740337530	-1.6625974280	2.2981968836	-6.3664756728	17.6364404555	1.8966292780
	1.0000000000	-1.7185578460	2.9534410700	-1.9225000260	6.3558366623	-10.9228729639	18.7715890329	1.1374131420
	1.0000000000	-0.5963667280	0.3556532743	-2.1891529700	0.5881590017	-0.3507584594	0.2091806747	0.4790972420
	1.0000000000	-0.7989707210	0.6383542130	-2.8547680380	54.1008856075	-43.2250235805	34.5355282554	1.1770543750
	1.0000000000	-2.3992114160	5.7562154187	0.0698140360	0.0018293866	-0.0043890852	0.0105303432	0.9896233860
	1.0000000000	0.0312532770	0.0009767673	-1.6948918910	13.9846567029	0.4370663497	0.0136597557	0.5688993630
	1.0000000000	-1.7229590960	2.9685880465	0.5650290970	0.0023987914	-0.0041330194	0.0071210234	0.9809768160
	1.0000000000	-0.8009777400	0.6415653400	0.1423767750	0.0151877725	-0.0121650677	0.0097439484	1.0216220230
	1.0000000000	-1.9785631370	3.9147120871	1.1225983110	6.1028557788	-12.0748854743	23.8909232829	1.3380020930

4. Stabilize (Regularize) the vectors of **P** by randomly disturbing the original values as above (call the resulting data **S**).

$\delta_H = 10^{-6}$

Sign	T000	T001	T002	T010	T220	T221	T222	F
	1.0000002947	-2.0426934542	4.1725939702	-2.9190951039	34.5028346056	-70.4786296826	143.9661883781	0.9395592270
	1.0000002722	-2.2761045436	5.1806477654	-1.7340810022	16.5024148836	-37.5612024031	85.4931657542	1.5887017490
	1.0000003840	-2.7702051031	7.6740369399	-1.6625984534	2.2981974015	-6.3664759702	17.6364468918	1.8966292780
	1.0000006046	-1.7185578618	2.9534415937	-1.9225001668	6.3558403026	-10.9228747546	18.7715949571	1.1374131420
	1.0000004054	-0.5963672327	0.3556534116	-2.1891531680	0.5881593613	-0.3507585260	0.2091808234	0.4790972420
	1.0000005160	-0.7989712525	0.6383543208	-2.8547702920	54.1008950178	-43.2250550988	34.5355288164	1.1770543750
	1.0000001181	-2.3992125092	5.7562160751	0.0698140372	0.0018293879	-0.0043890869	0.0105303529	0.9896233860
	1.0000004127	0.0312533052	0.0009767677	-1.6948920394	13.9846697913	0.4370665319	0.0136597627	0.5688993630
	1.0000001908	-1.7229607247	2.9685905704	0.5650293245	0.0023987924	-0.0041330215	0.0071210273	0.9809768160
	1.0000000058	-0.8009785188	0.6415656204	0.1423767867	0.0151877742	-0.0121650788	0.0097439536	1.0216220230
	1.0000006442	-1.9785651027	3.9147141785	1.1225992323	6.1028588602	-12.0748969501	23.8909338454	1.3380020930

5. Select a subset of size M from **S**. Select the vectors so that the distance from each one is equi-distant to **O**. Call it **I**. Call the remaining vectors **E**.

6. Obtain the minimax signs (call the matrix incorporating the σ's **A**). $\mathbf{M}^3$

Sign	T000	T001	T002	T221	T222	F
1	1.0000002947	-2.0426934542	4.1725939702	-70.4786296826	143.9661883781	0.939559227
-1	1.0000002722	-2.2761045436	5.1806477654	-37.5612024031	85.4931657542	1.588701749
1	1.0000003840	-2.7702051031	7.6740369399	-6.3664759702	17.6364468918	1.896629278
1	1.0000006046	-1.7185578618	2.9534415937	-10.9228747546	18.7715949571	1.137413142
-1	1.0000004054	-0.5963672327	0.3556534116	-0.3507585260	0.2091808234	0.479097242
-1	1.0000005160	-0.7989712525	0.6383543208	-43.2250550988	34.5355288164	1.177054375
-1	1.0000001181	-2.3992125092	5.7562160751	-0.0043890869	0.0105303529	0.989623386
1	1.0000004127	0.0312533052	0.0009767677	0.4370665319	0.0136597627	0.568899363
1	1.0000001908	-1.7229607247	2.9685905704	-0.0041330215	0.0071210273	0.980976816
-1	1.0000000058	-0.8009785188	0.6415656204	-0.0121650788	0.0097439536	1.021622023
-1	1.0000006442	-1.9785651027	3.9147141785	-12.0748969501	23.8909338454	1.338002093
-1	1.0000006967	1.2349666661	1.5251420117	8.5129532821	10.5132081982	0.713283612
-1	1.0000002412	-2.3977634183	5.7492653244	-21.6993397194	52.0298274334	1.206384789
-1	1.0000008414	-0.7645727507	0.5845711099	-2.9343672973	2.2435366570	0.803413430
1	1.0000000344	-2.2169875525	4.9150352964	-18.6864128209	41.4275243401	1.222459067
1	1.0000002501	-0.1818904535	0.0330841465	-0.2046635286	0.0372263424	1.275810486
1	1.0000005739	2.1195801660	4.4926177670	61.3712627521	130.0812604422	0.799190439
1	1.0000005917	1.4685606069	2.1566674349	0.0000215548	0.0000316545	0.982881794
-1	1.0000003723	2.0848252309	4.3464891498	3.7104389273	7.7356110606	1.347248151
-1	1.0000008256	1.1305318741	1.2781008621	4.7050365752	5.3191879951	0.974499285
1	1.0000006835	-1.9627651935	3.8524465275	-109.0990070346	214.1356047481	0.792124190
-1	1.0000002084	-0.7010576051	0.4914814510	-16.7892120819	11.7701937530	1.039683987
1	1.0000005484	0.2892868180	0.0836868726	3.9580909595	1.1450235398	1.348051842
-1	1.0000001085	-1.7509114682	3.0656895372	-38.9271671545	68.1579733746	0.984336640
-1	1.0000009375	2.3453096266	5.5004724869	0.1905082336	0.4468007139	0.783560730
1	1.0000006476	0.8546895013	0.7304937256	6.2665401892	5.3559405582	1.469517142
-1	1.0000007669	1.1313023467	1.2798436574	0.4683248475	0.5298166868	1.045203149
-1	1.0000003576	2.3937449708	5.7300134247	102.7844618688	246.0394477031	1.056127155

7. Obtain the inverse of **A** (call it **B**). **M^3**

Inverse

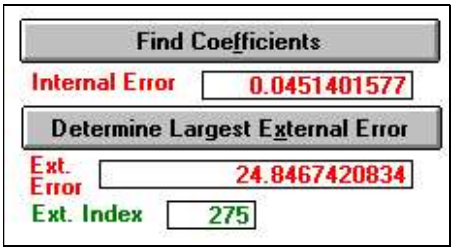
C01	C02	C03	C26	C27	C28
0.0033135870	-0.0049355069	0.0024773966	0.0353475697	-0.0158701659	-0.0037988093
-0.1084934236	0.1055586817	0.0813879072	0.0287944436	0.0203473227	-0.0048983187
0.1187729567	-0.1066531799	-0.0964532056	0.0572614146	-0.0790797955	-0.0042949033
0.0759044025	-0.0704716241	-0.0483188819	0.0203970212	-0.0476050163	-0.0006572578
0.3880975606	-0.3797100978	-0.2359897450	0.1712508150	-0.2032575974	-0.0108137530
0.0699540572	-0.0614241725	-0.0223672785	0.1225401671	-0.0948586076	-0.0110742454
-0.0577809144	0.0676596393	0.0067259922	-0.0058559547	0.0197062994	-0.0006425957
0.1340850090	-0.1241570655	-0.1039988382	0.0327563849	-0.0286051274	-0.0004195547
-0.1390641056	0.1325131376	0.0840579064	-0.0607425417	0.1333043876	0.0048960456
-0.1000363025	0.0891041088	0.0828124138	-0.0455171548	0.0789844796	0.0032134743
0.0011870108	-0.0131836400	-0.0006574818	-0.0661208964	0.0264555116	0.0044210447
-0.1653817997	0.1549658760	0.1248925824	-0.1072957664	0.1072519368	0.0058794847
-0.0675670535	0.0594000693	0.0432685117	-0.0263810583	0.0501520161	0.0004062094
0.0141860831	-0.0049751405	-0.0051892848	0.0814663721	-0.0362858593	-0.0035724080
0.0340649810	-0.0592922427	-0.0037989065	-0.0270579149	0.0006532737	0.0062221348
0.0121206182	-0.0198690518	0.0131250447	-0.0300246274	0.0023533311	0.0037477049
-0.0162697440	0.0149678304	0.0072400339	0.0610330263	-0.0105032076	-0.0040208106
0.0074155653	0.0038623615	-0.0146433851	0.0414930471	-0.0153851274	-0.0002064548

8. Calculate the coefficients **C = f B**. The maximum internal ε_θ error is also calculated. **M^2**

Coefficients

Name	Value
Error	0.0451401577
C000	0.7577224960
C001	0.1999216791
C002	0.1658770194
C010	0.5977731948
C011	0.2350491710
C012	-0.0635942927
C020	0.0955334127
C021	-0.1914127831
C022	-0.1229542431
C100	0.0521686959
C101	-0.4083442433
C102	-0.2375757848
C110	0.0987854662
C111	0.0470563221
C112	0.0170028427
C120	-0.0283681491
C121	0.0233687824

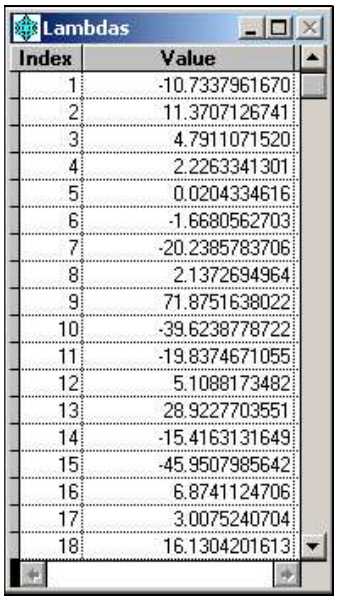
9. Calculate the maximum external error ε_ϕ from **C** and **E**. Call its index I_E **MN-M^2**



10. $\varepsilon_\theta \geq \varepsilon_\phi$?

YES: STOP; the coefficients of **C** are those of the minimax polynomial for lhe **D** vectors.

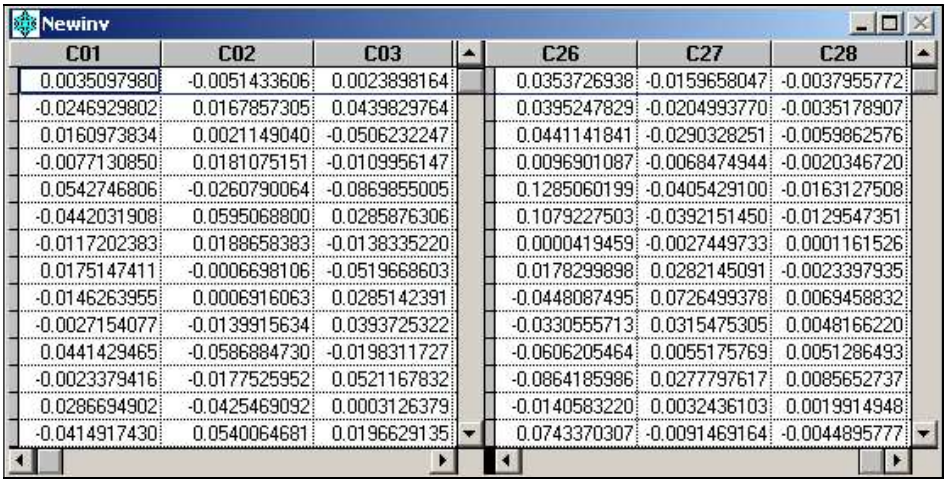
11. Calculate the **λ** vector from $\boldsymbol{\lambda} = \mathbf{A}^{m+1}\mathbf{B}$. **$M^2$**



12. Calculate the vector **β** which maximizes $\sigma_\theta \frac{\lambda_j}{B_j}$. Call its index I_I. **M**

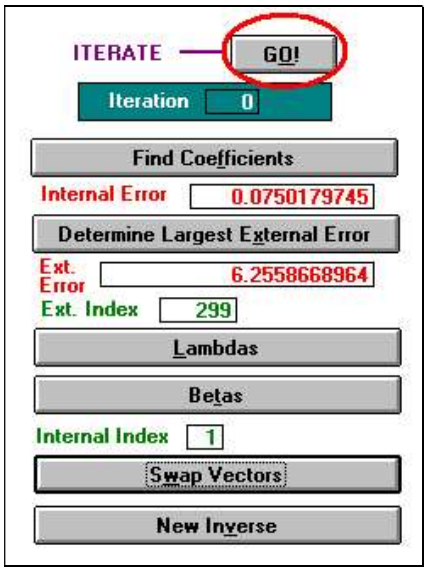
13. Interchange vectors I_E and I_I.

14. Calculate the new inverse $\overline{\mathbf{B}}$. Make **B** ← $\overline{\mathbf{B}}$. **M^2**



15. Go to step 8.

STEPS 8 TO 15 ARE REPEATED UNTIL $\varepsilon_{\theta} \geq \varepsilon_{\varphi}$.



First 10 iterations:

History

Iteration	Epsint	Epsext	Swap	For
1	0.0451401348	24.8467498935	17	275
2	0.0455936706	-9.1789523194	19	240
3	0.0457554911	-11.4391356792	11	250
4	0.0473556764	6.6116659489	8	275
5	0.0530785802	3.9096276300	16	184
6	0.0750172860	6.2558292657	1	299
7	0.0824598244	-16.4126472815	6	29
8	0.0849295773	19.6812235406	27	122
9	0.0864286944	-11.3266006802	12	271
10	0.0915366648	24.6258354576	14	268
11	0.0961549769	9.4177874408	28	96
12	0.0991567406	16.6214219949	14	96

Last 10 iterations:

History

Iteration	Epsint	Epsext	Swap	For
122	0.8157048320	-0.8749006847	5	171
123	0.8157564518	-0.8742434238	14	108
124	0.8157693031	-0.8398627406	14	101
125	0.8157871670	0.9184479573	15	288
126	0.8158028274	-0.8344687442	13	227
127	0.8158308990	0.8633730504	15	242
128	0.8158509421	-0.8515084653	2	260
129	0.8158719332	-1.0077837912	4	171
130	0.8158822664	0.8600771356	4	204
131	0.8158868582	0.8243462367	4	217
132	0.8159055853	0.8223590548	2	260
133	0.8159058474	0.8146019470		

ε_θ ε_φ

MINIMAX COEFFICIENTS

Error	0.814602	C111	-0.001523
C000	0.839653	C112	-0.011812
C001	0.065212	C120	0.019597
C002	-0.040531	C121	0.001878
C010	0.028232	C122	-0.009392
C011	0.051248	C200	0.020988
C012	-0.022493	C201	0.003893
C020	0.100721	C202	0.013053
C021	-0.016508	C210	0.016817
C022	-0.005530	C211	-0.007795
C100	-0.116207	C212	0.002654
C101	-0.003137	C220	-0.019028
C102	0.028792	C221	-0.002560
C110	0.035447	C222	0.001214

Acceleration of the Algorithm Via Heuristics

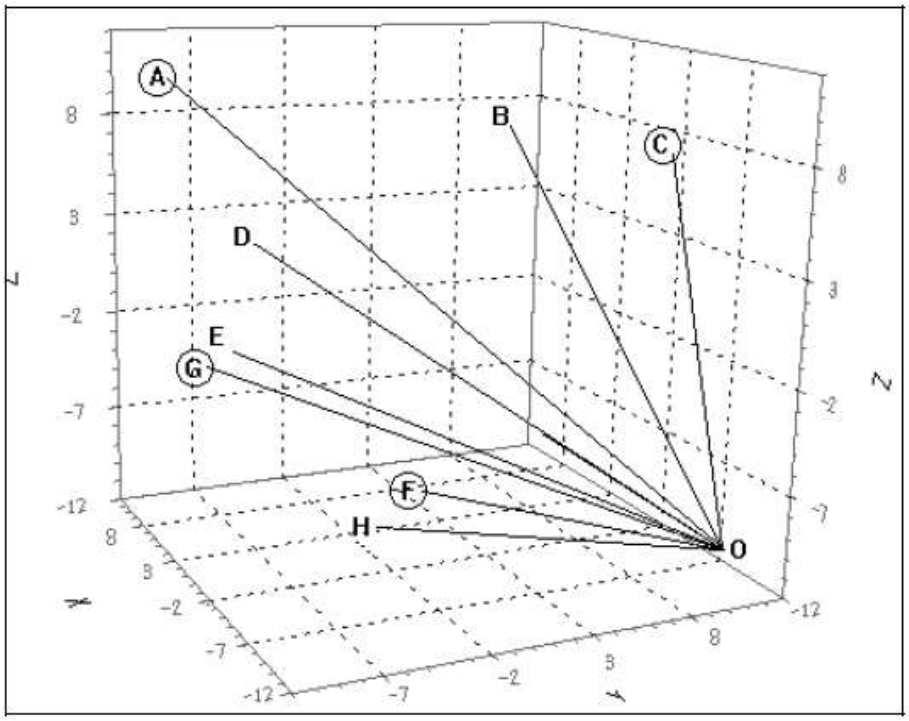
The Fast Ascent algorithm may be further accelerated from heuristic considerations, as follows.

Heuristic 1. Data Spread.

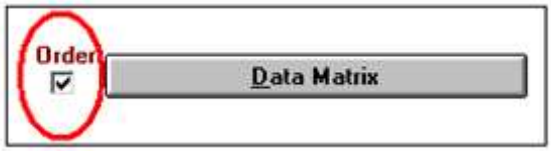
The process of finding the minimax coefficients may be reformulated as *the search for the inner set with the largest minimax error*. That is: Find the set of size M such that ε_θ is maximum. In this regard it is intuitively appealing that the initial inner set should cover (in some sense) the whole range of set S. Denoting by $\varepsilon_\theta(t)$ the maximum error of the inner set at iteration *t*, we may assume that the target (final) error $\varepsilon_\theta(\tau)$ will be reached in a smaller number of iterations. Accordingly we may adopt the following strategy:

a) Find the vector whose coordinates correspond to the smallest values for all the variables. Call this **O**.
b) Find the Euclidean distance between all vectors in **S** and **O**.
c) Order the vectors by their distance to **O**.
d) Select the M vectors equi-distant from **O** to **| D |**.

In the illustration the vectors farthest from **O** is **A**, and the selected vectors are **O**, **F**, **G**, **C** and **A**.

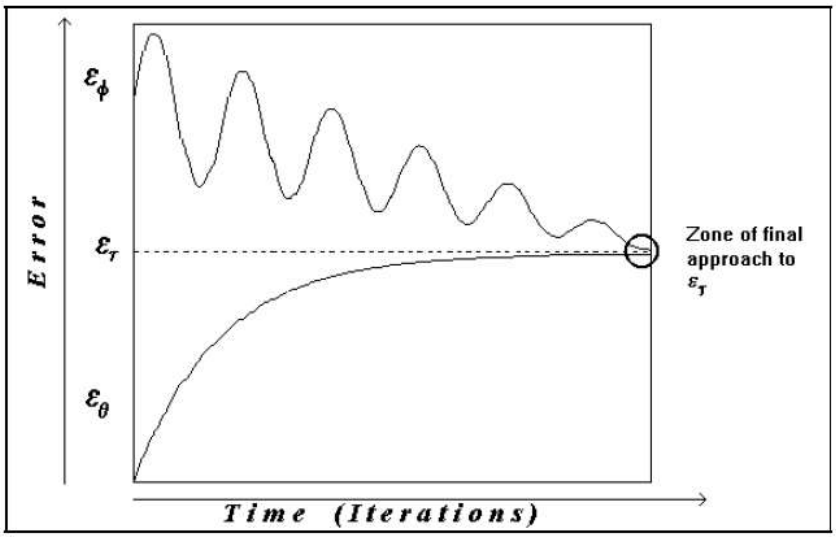


This option is selected when the "Order" (by their Euclidean distance) option is enabled.

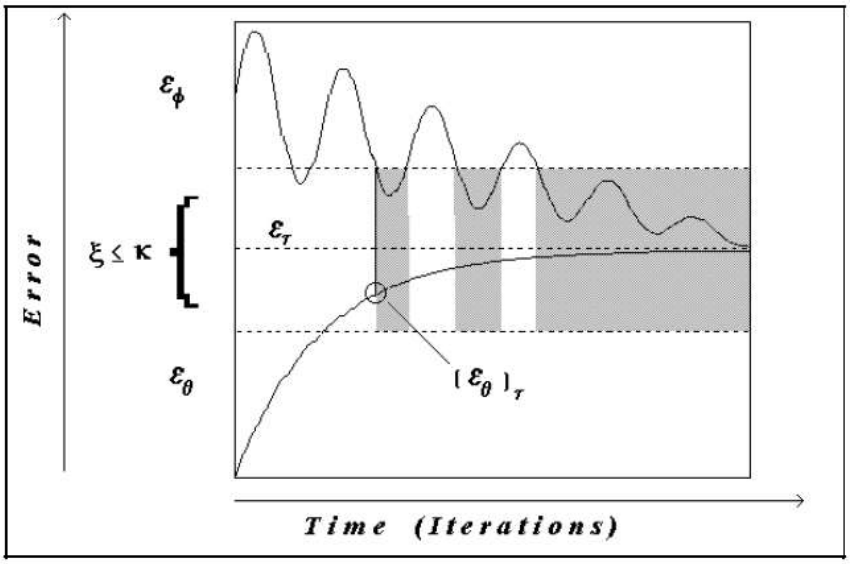


Heuristic 2. Quasi-Minimax.

When the algorithm converges, ε_θ and ε_ϕ tend to each other as shown in the next illustration.



The convergence is of linear order and, in the very last iterations, does not improve the solution significantly, as shown in the next figure.



Zone of final approach to best fit.

The following strategy suggests itself.

a) Define a threshold κ such that $1-(\varepsilon_\phi / \varepsilon_\theta) \leq \kappa$.

b) When condition (a) is reached, stop the algorithm.

This is a *quasi-minimax* solution because the "true" minimax coefficients are not reached. However, this approach does not seriously hamper the quality of the solution but, on the other hand, may save a significant number of iterations.

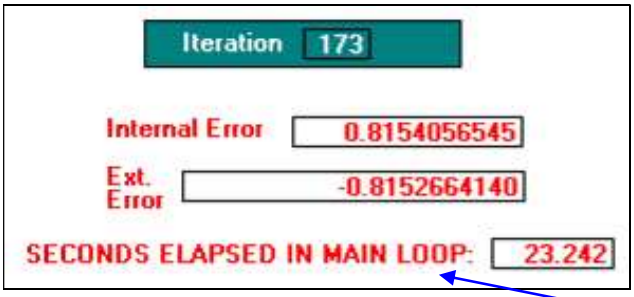
In the program, the number in the box is the value of κ .

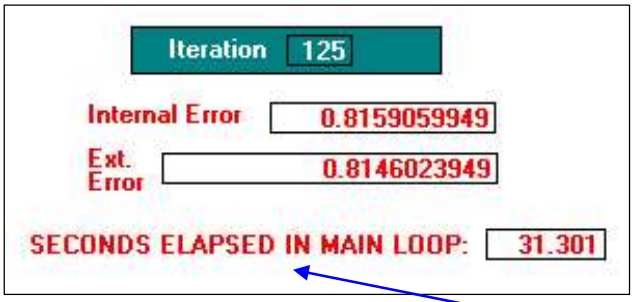
Heuristic 3. Sampling the External Set.

In the Fast Ascent Algorithm considerable time may be spent in the evaluation of ε_ϕ . As pointed out, it is O(MN). Typically N >> M. For instance, in the example, Z3VARS.TXT consists of 300 objects. If the degrees of the variables are defined as =2 for all three variables we have that M=28 and N=272. Therefore, the calculation of any one of the external errors needs 55 x 272 FLOPS. [The "55" results from the number of products implied in the collection of monomials of an approximant of the chosen degrees]. Therefore, there are 14,960 ($\approx 2MN$) FLOPS per iteration to calculate ε_ϕ .

An interesting strategy is NOT to calculate the value of ε_ϕ for all the external points. Rather, one may sample **E** and save computations.

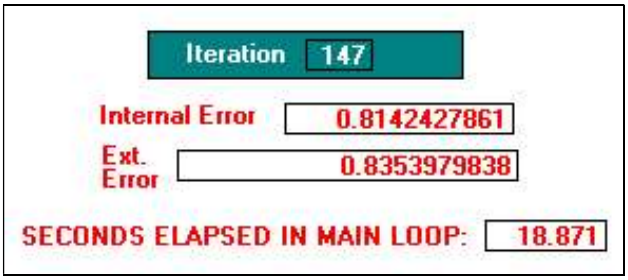
In the program one chooses how many of the elements of **E** are required to satisfy the minimax condition and how reliable this estimate is to be. In the example, we require that 95% of the elements (272 x .95 ≈ 258) satisfy the minimax condition with probability 0.99. That is, we risk that 14 of the objects are NOT minimax in 99% of the cases. Under this assumption we need only to sample 100 (instead of 272) elements of **E**. This means that there are only 5,500 FLOPS per iteration. An efficiency improvement of 63.23% Of course we know that not all the vectors will satisfy the norm. On the other hand, this strategy improves the performance on the algorithm in the case of outliers in the data.





In the figure above, the left box corresponds to the sampled data; the right box corresponds to the full data case. Notice that the number of iterations increased from 125 to 173 when data is sampled. However, the execution time decreased from 31.301 to 23.242 seconds.

If all three heuristics are applied with spread out vectors, $\kappa = 0.05$ *quasi-minimax*, 95% compliance and 99% reliability, we get 147 iterations in 18.871 seconds. This is illustrated en the next figure.




```
function getSampleSize
*
*
*     Size of sample   --> N
*     % in norm        --> inNorm
*     Elements in norm --> En=N*inNorm
*     Confidence       --> confidence
*     Prob. that S elements are in norm --> Ps
*
En=ceil(N*inNorm)
En=iif(En=N,N-1,En)
Ps=1
Conf=1-confidence
for i=0 to N-1
      Ps=Ps*((En-i)/(N-i))
      if Ps<=Conf
            exit
      endif
endfor
return i
```

Conclusions

The fast ascent algorithm achieves reasonable results for difficult approximation problems. In the example, the degree of the polynomial is 6 ($X^2Y^2Z^2$) and data was tackled without scaling. Higher stability may be achieved by enclosing the values of D in a closed interval. For instance, the interval [0,1) is recommended.

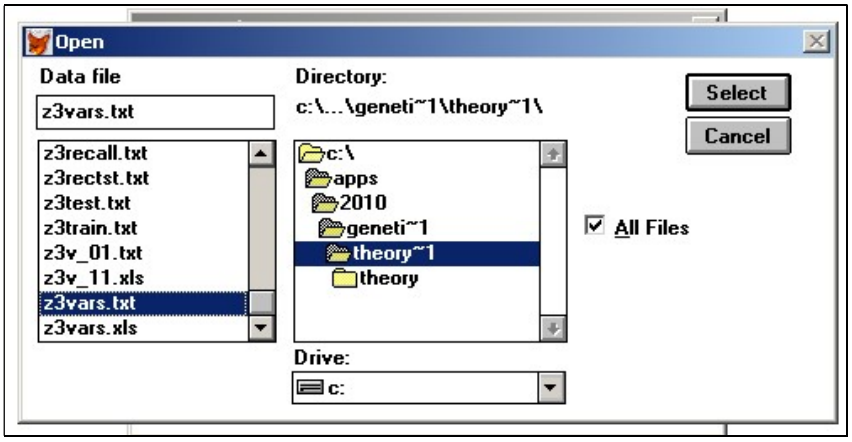
One question which arises immediately is whether there is a way to select the *form* of the approximant. This issue will be analyzed next.

SELECTED MONOMIALS

It is natural to ask oneself if a partial combination of the monomials in equation (0) is able to approximate the data. That is we are given a set of values of V_1, V_2,. . .,V_n and the degrees in P(**v**) = $\sum_{v_1=0}^{d_1} \cdots \sum_{v_n=0}^{d_n} c_{i_1 \ldots i_n} v_1^{i_1} \ldots v_n^{i_n}$

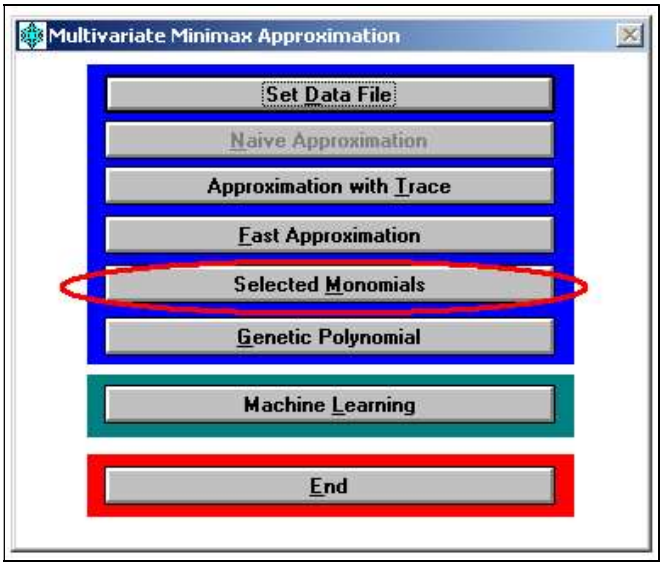
This implies that the number of monomials in (0) is $N_m = \prod_{i=1}^{n} (d_i + 1)$. In the example of (1) n = 3, $d_i = 1 \ \forall i$ and, therefore, N_m = 8. However, had we chosen $d_i = 3$, for instance, we would have gotten N_m = 64. It is implicit that we have to create a map from R^n into R^{N_m}. The question is: "If instead of considering *all* the monomials from (0) we select a subset from the R^{N_m} mapping, will it be able to approximate the original data properly?" The answer has to do with the fact that in selecting a full polynomial we tacitly assume that all the linear combinations of the monomials are required to achieve a good approximation.

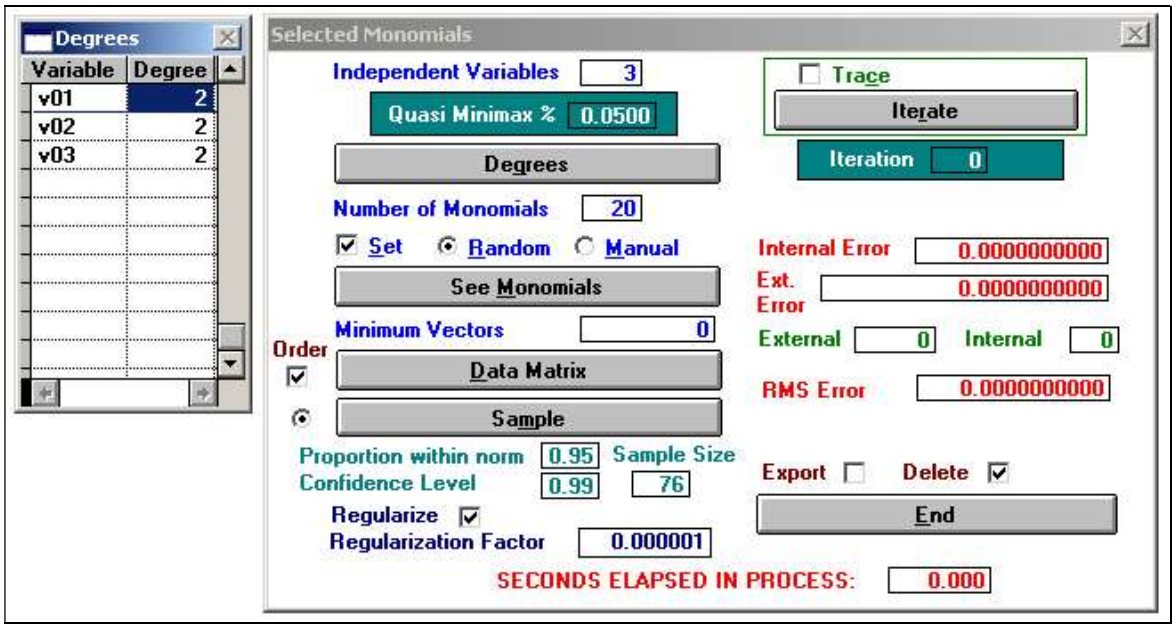
In what follows we present an example of program *GMP1* which was designed to test alternative combinations of monomials.



Data File

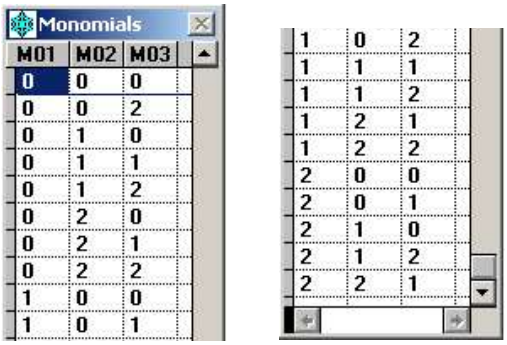
V01	V02	V03	F
1.4111970760	0.6344440890	-1.8566366910	0.8817407070
1.6939553160	-0.3968748860	1.2390210770	0.8962512570
-0.8177116800	-1.4784499750	-2.4402897540	0.3379039350
2.3101339540	-2.9064715600	1.4107567630	1.0114966370
-2.1201298040	-0.5268444960	-1.1892259540	0.9169912640
0.9752722710	-2.5830607740	-0.8718351180	1.1592536870
0.2168954760	1.3984325580	1.0280978450	1.4975148000
-1.6150987990	-2.6173383990	1.2741184140	1.1242601520
-2.6609439140	-1.8622170130	0.3875441500	1.6419683010
-1.7999534200	-0.0745716210	0.4346738710	1.0045321540
2.8365544370	1.0690850650	0.9000680780	1.3925816520
-1.5405914100	-2.8802782300	0.7523868700	1.1189463380
1.9804730000	-0.5067041680	-1.9445359420	1.0917083620
-1.3097231320	2.4770853030	-1.0736765790	1.0065664620
-2.8640745120	1.1921759700	-1.2772764520	0.9169812640
-0.5539860640	2.5708253580	-2.3718380970	1.1194886140



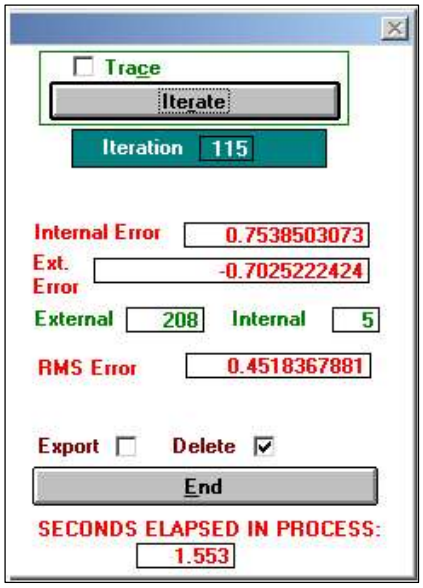


In the figures on the right we can see a random selection of 20 out of the possible 27 monomials which conform the full polynomial.

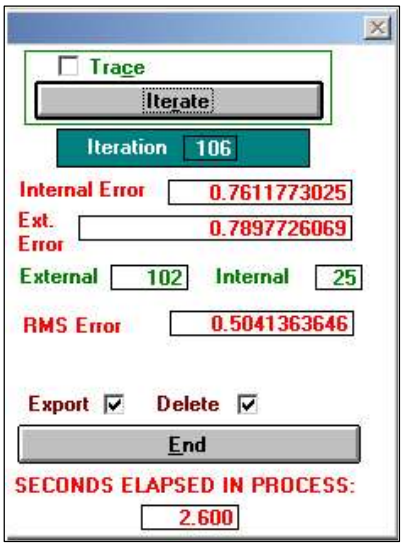
The parameters for the run are shown above. We apply the Fast Ascent Algorithm (FAA) to approximate the original data.



SELECTED MONOMIALS (20 TERMS)

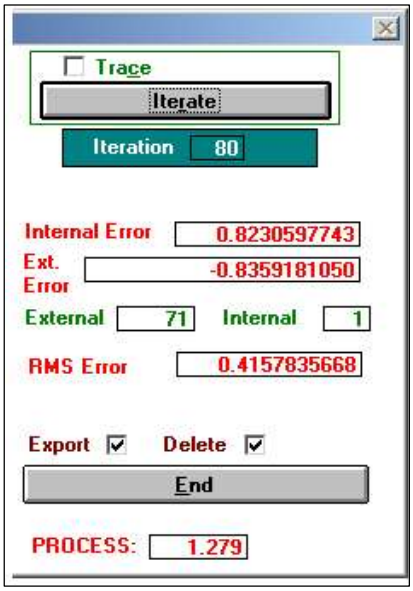


FULL POLYNOMIAL (27 TERMS)

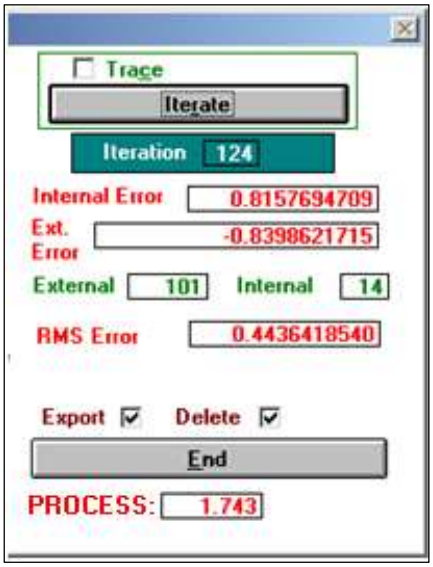


If we eliminate data sampling, we get the following results:

SELECTED MONOMIALS (20 TERMS)



FULL POLYNOMIAL (27 TERMS)



Notice that in both cases the minimax error is approximately 0.84 whereas the RMS error is better (smaller) in the reduced polynomial set (0.416 vs. 0.444). Also notice that in this particular case sampling yields no advantage for the reduced set.

COEFFICIENTS

The calculated coefficients are stored in an EXCEL file: Coefs.xls, as follows.

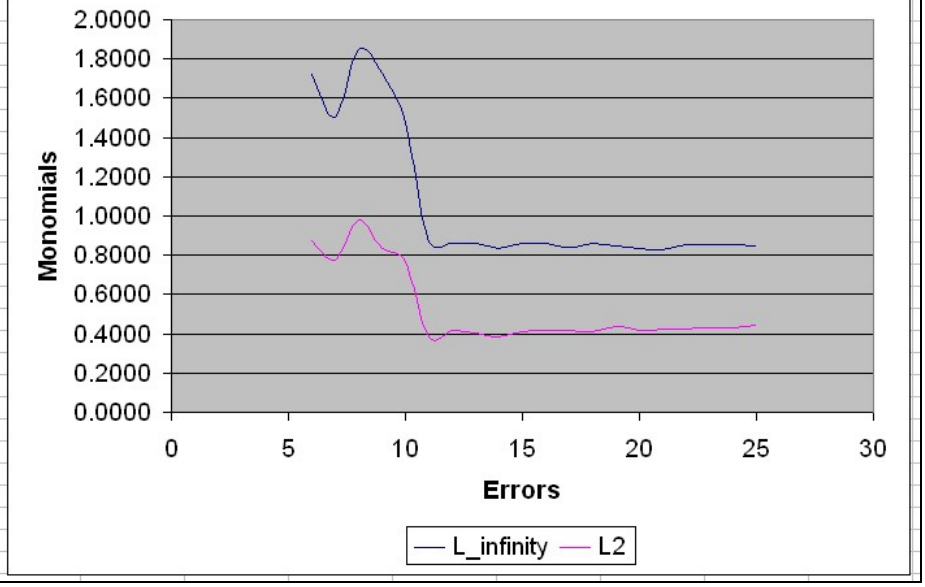
Error	0.835918105			
C000	0.969123726		C102	0.02363432
C002	-0.041908528		C111	-0.01323613
C010	-0.058971973		C112	-0.00011932
C011	1.5962E-06		C121	-0.00259958
C012	0.010994905		C122	-0.00270938
C020	-0.017943117		C200	-0.00160157
C021	-0.015176284		C201	0.00237693
C022	0.019764182		C210	0.03206213
C100	-0.096676792		C212	-0.0067841
C101	-0.020189197		C221	-0.0002111

The indices of the coefficients denote the degrees of the associated variables. For example, calling the independent variables "X", "Y" and "Z", C000 corresponds to $X^0Y^0Z^0$. Likewise C201 corresponds to $X^2Y^0Z^1$, etc. Notice, for instance, that the coefficient associated to YZ is practically zero, indicating that this combination has no significant effect on the functional dependence of this function. Now we explore this fact by reducing the number of required monomials to 19. We get:

Error	0.85326813			
C000	1.08827889		C111	0.00431290
C002	-0.07058305		C112	-0.00306920
C010	-0.04407687		C121	0.00375555
C011	0.01720771		C122	-0.00015849
C012	0.00262038		C200	0.01061092
C020	-0.06372181		C201	0.01066720
C021	0.00347966		C210	0.01812815
C022	0.02973749		C212	-0.00148882
C100	-0.02304457		C221	-0.00720823
C101	-0.01597454			

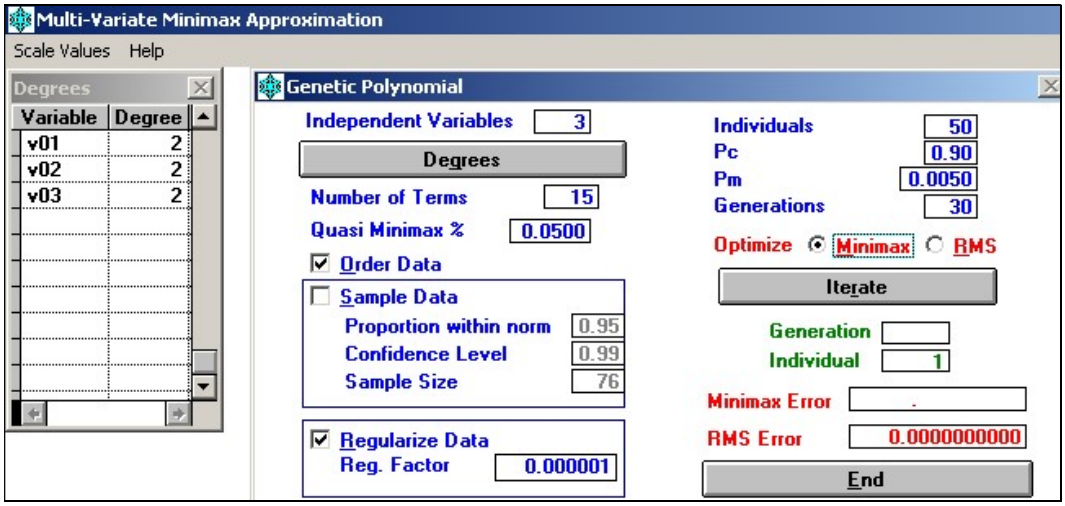
The error has diminished slightly. We may explore systematically the number of required monomials to get, successively:

Monomials	L_infinity	L2
6	1.7198	0.8733
7	1.5066	0.7757
8	1.8448	0.9791
9	1.7296	0.8369
10	1.4745	0.7710
11	0.8662	0.3818
12	0.8591	0.4183
13	0.8637	0.4048
14	0.8349	0.3869
15	0.8599	0.4101
16	0.8619	0.4176
17	0.8415	0.4171
18	0.8627	0.4101
19	0.8473	0.4358
20	0.8359	0.4158
21	0.8253	0.4231
22	0.8513	0.4227
23	0.8559	0.4326
24	0.8568	0.4324
25	0.8467	0.4441



Notice that the approximation errors decrease sharply when the number of monomials passes from 10 to 11. It reaches its smallest value when there are 15 monomials. However, the question arises (since monomials were generated randomly): Is there a best combination of monomials?

To answer this question we wrote a program which tries to minimize the approximation error using a genetic algorithm. The settings are as follows:



The results:

FIRST 20 BEST FITNESS

1	6	11	16
0.78220500	0.78588597	0.78588607	0.79532022
0.78220500	0.78588607	0.79175993	0.79660747
0.78220500	0.78588607	0.79280282	0.79707511
0.78220500	0.78588607	0.79285829	0.79707680
0.78588589	0.78588607	0.79383880	0.79707892

AVERAGE FITNESS 0.79523700

The Ascent Algorithm always obtains the coefficients in the minimax sense. But the GA is allowed to evolve either optimizing the L_∞ errors or the L_2 (RMS) errors. Now we optimize the RMS error with identical parameters (as above).

The results:

FIRST 20 BEST FITNESS

1	6	11	16
0.37212064	0.38028093	0.38055885	0.38201498
0.37896054	0.38028093	0.38163674	0.38201887
0.37896083	0.38028093	0.38163674	0.38289850
0.38028093	0.38028093	0.38180485	0.38321264
0.38028093	0.38028093	0.38186874	0.38321584

AVERAGE FITNESS 0.38266483

Notice that in both cases the L and the L2 errors have decreased relative to the random selection of monomials. Clearly, had we chosen the monomial combination which the GA selected, we would have gotten better results that from a purely random choice.

Two Questions

There are two issues of primary importance to explore:

a) What are the possible forms of the approximants

b) How should we select the number of elements in the approximant.

APPENDIX 1
SYSTEMS OF SIMULTANEOUS LINEAR EQUATIONS

A method to solve a system as (3) is Gauss-Jordan elimination (see, for instance, http://en.wikipedia.org/wiki/-Gauss%E2%80%93Jordan_elimination).

SYSTEMS OF SIMULTANEOUS LINEAR EQUATIONS

CODE

```
#
#       Find the coefficients for (M)X(M+1) system
#           ON INPUT: data is in ab[M][M+1] matrix
#           ON OUTPUT: coefficients are in cf[M]
#
def lassol():
    for i in range(0,M):
       rowmax=abs(ab[i][0])
       for j in range(1,M+1):
           rowmax=max(rowmax,abs(ab[i][j]))
       #endFor
       if (rowmax==0):
         return False
       #endIf
       scale=1/rowmax
       for j in range(0,M+1):
           ab[i][j]=ab[i][j]*scale
       #endFor
    #endFor
    for k in range(0,M):
        big=0
        for i in range(k,M):
            temp=abs(ab[i][k])
            if (big<temp):
                big=temp
                ipiv=i
            #endIf
        #endFor
        if (big==0):
            return False
        #endIf
#
#     Exchange column with largest element
#
        if (ipiv!=k):
            for i in range(k,M+1):
                temp=ab[k][i]
                ab[k][i]=ab[ipiv][i]
                ab[ipiv][i]=temp
                  #endFor
           #endIf
#
#     Eliminate all in column except first
#
```

```
        for i in range(k+1,M):
            quot=ab[i][k]/ab[k][k]
            for j in range(k+1,M+1):
                ab[i][j]=ab[i][j]-quot*ab[k][j]
                  #endfor
        #endFor
    #endFor
    if (ab[M-1][M-1]==0):
        return False
    #endIf
    cf[M-1]=ab[M-1][M]/ab[M-1][M-1]
    for i in range(M-2,-1,-1):
        sum=0
        for j in range(i+1,M):
            sum=sum+ab[i][j]*cf[j]
        #endFor
        cf[i]=(ab[i][M]-sum)/ab[i][i]
    #endFor
    return True
#endLassol
```

APPENDIX 2
COMPUTATION OF A DETERMINANT

To calculate a determinant we may apply a simplified version of Gauss-Jordan's algorithm. In that case the cost of the determinant is $O(M^3)$. The pseudo code follows.

```
function determ
parameter M
*
*      on input:
*
*      A    - M x M array
*      det  -  variable for determinant
*
*      on output:
*
*      det <- determinant of A
*
*      F(alse) if unstable
*      T(rue)  if stable
*
priv i,j,k,kp1,quot
*
*     Eliminate all in column except first
*
for k=1 to M
    kp1=k+1
    for i=kp1 to M
      quot=A(i,k)/A(k,k)
      for j=kp1 to M
         A(i,j)=A(i,j)-quot*A(k,j)
      endfor
    endfor
endfor
det=1
*
*     Get determinant from main diagonal
*
for i=1 to M
   if A(i,i)=0
      wait window 'Determinant is ZERO' time 1
      det=0
      return .F.
   endif
   det=det*A(i,i)
endfor
return .T.
```

APPENDIX 3
EXAMPLE OF SAMPLING IN THE FAST ASCENT ALGORITHM

N	300	===>	NÚMERO DE OBJETOS EN LA MUESTRA
M	28	===>	NÚMERO DE OBJETOS EN EL CONJUNTO INTERNO
N-M	272	===>	NÚMERO DE OBJETOS EN EL CONJUNTO EXTERNO
C	0.95	===>	PORCENTAJE DE OBJETOS QUE SATISFARÁN LA NORMA EN EL CONJUNTO EXTERNO
R	0.99	===>	PROBABILIDAD DE QUE EL PORCENTAJE "C" SEA CORRECTO
T	258	===>	NÚMERO DE OBJETOS QUE SATISFARÁN LA NORMA EN EL CONJUNTO EXTERNO
K	286	===>	NÚMERO TOTAL DE OBJETOS QUE SATISFARÁN LA NORMA [T+M]

I	PROBABILIDAD DE ESTAR EN NORMA PARA LA MUESTRA #I	PROBABILIDAD DE QUE TODAS LAS MUESTRAS ESTÉN EN LA NORMA	I	EN NORMA	P(TODAS)	I	EN NORMA	P(TODAS)	I	EN NORMA	P(TODAS)	TAMAÑO DE MUESTRA OK
1	0.9485	0.9485	26	0.9433	0.8400	51	0.9369	0.1794	76	0.9289	0.0316	N
2	0.9483	0.8995	27	0.9431	0.7922	52	0.9367	0.1680	77	0.9286	0.0294	N
3	0.9481	0.8992	28	0.9429	0.7469	53	0.9364	0.1573	78	0.9282	0.0273	N
4	0.9480	0.8988	29	0.9426	0.7041	54	0.9361	0.1473	79	0.9278	0.0253	N
5	0.9478	0.8984	30	0.9424	0.6635	55	0.9358	0.1378	80	0.9275	0.0235	N
6	0.9476	0.8981	31	0.9421	0.6251	56	0.9355	0.1289	81	0.9271	0.0217	N
7	0.9474	0.8977	32	0.9419	0.5888	57	0.9352	0.1206	82	0.9267	0.0202	N
8	0.9472	0.8973	33	0.9417	0.5545	58	0.9349	0.1127	83	0.9263	0.0187	N
9	0.9470	0.8969	34	0.9414	0.5220	59	0.9346	0.1053	84	0.9259	0.0173	N
10	0.9468	0.8966	35	0.9412	0.4913	60	0.9343	0.0984	85	0.9255	0.0160	N
11	0.9466	0.8962	36	0.9409	0.4623	61	0.9340	0.0919	86	0.9251	0.0148	N
12	0.9464	0.8958	37	0.9407	0.4348	62	0.9336	0.0858	87	0.9247	0.0137	N
13	0.9462	0.8954	38	0.9404	0.4089	63	0.9333	0.0801	88	0.9243	0.0126	N
14	0.9459	0.8950	39	0.9402	0.3845	64	0.9330	0.0747	89	0.9239	0.0117	N
15	0.9457	0.8946	40	0.9399	0.3614	65	0.9327	0.0697	90	0.9235	0.0108	N
16	0.9455	0.8942	41	0.9397	0.3396	66	0.9324	0.0650	91	0.9231	0.0100	S
17	0.9453	0.8938	42	0.9394	0.3190	67	0.9320	0.0606	92	0.9227	0.0092	S
18	0.9451	0.8934	43	0.9391	0.2996	68	0.9317	0.0564	93	0.9222	0.0085	S
19	0.9449	0.8930	44	0.9389	0.2813	69	0.9314	0.0526	94	0.9218	0.0078	S
20	0.9447	0.8926	45	0.9386	0.2640	70	0.9310	0.0489	95	0.9213	0.0072	S
21	0.9444	0.8922	46	0.9383	0.2477	71	0.9307	0.0455	96	0.9209	0.0066	S
22	0.9442	0.8918	47	0.9381	0.2324	72	0.9303	0.0424	97	0.9205	0.0061	S
23	0.9440	0.8913	48	0.9378	0.2179	73	0.9300	0.0394	98	0.9200	0.0056	S
24	0.9438	0.8909	49	0.9375	0.2043	74	0.9296	0.0366	99	0.9195	0.0052	S
25	0.9435	0.8905	50	0.9372	0.1915	75	0.9293	0.0340	100	0.9191	0.0047	S