Compensation Tables in SoftMC system

In SOFTMC system multidimensional compensation tables are used to compensate individual axes. The method of compensation is multidimensional, means that compensation value of an arbitrary point is obtain by multidimensional interpolation between compensation values of neighboring points as described in http://www.library.cornell.edu/nr/bookcpdf/c3-6.pdf :

Interpolation in Two or More Dimensions

In multidimensional interpolation, we seek an estimate of $y(x_1, x_2, \ldots, x_n)$ from an n-dimensional grid of tabulated values y and n one-dimensional vectors giving the tabulated values of each of the independent variables $x_1, x_2, \ldots, x_n$. We will not here consider the problem of interpolating on a mesh that is not Cartesian, i.e., has tabulated function values at “random” points in n-dimensional space rather than at the vertices of a rectangular array. For clarity, we will consider explicitly only the case of two dimensions, the cases of three or more dimensions being analogous in every way.
In two dimensions, we imagine that we are given a matrix of functional values ya[1..m][1..n].We are also given an array x1a[1..m], and an array x2a[1..n]. The relation of these input quantities to an underlying function $y(x_1, x_2)$ is

$$ya[j][k] = y(x1a[j], x2a[k]) \quad (1)$$

We want to estimate, by interpolation, the function y at some untabulated point (x_1, x_2).
An important concept is that of the *grid square* in which the point (x_1, x_2) falls, that is, the four tabulated points that surround the desired interior point. For convenience, we will number these points from 1 to 4, counterclockwise starting from the lower left (see Figure 3.6.1). More precisely, if

$$\begin{aligned} x1a[j] &\le x_1 \le x1a[j+1] \\ x2a[k] &\le x_2 \le x2a[k+1] \qquad (2) \end{aligned}$$

defines j and k, then

$$\begin{aligned} y_1 &\equiv ya[j][k] \\ y_2 &\equiv ya[j+1][k] \\ y_3 &\equiv ya[j+1][k+1] \\ y_4 &\equiv ya[j][k+1] \qquad (3) \end{aligned}$$

The simplest interpolation in two dimensions is *bilinear interpolation* on the grid square. Its formulas are:

$$\begin{aligned} t &\equiv (x_1 - x1a[j])/(x1a[j+1] - x1a[j]) \\ u &\equiv (x_2 - x2a[k])/(x2a[k+1] - x2a[k]) \qquad (4) \end{aligned}$$

(so that t and u each lie between 0 and 1), and

$$y(x_1, x_2) = (1 - t)(1 - u)y_1 + t(1 - u)y_2 + tuy_3 + (1 - t)uy_4 \qquad (5)$$

The implemented compensation method in SOFTMC system is based on equidistant nodes. Compensation values of each axis are given for all nodes of n-dimensional interpolation. So, having n axes in the system and each axis has m_i points on which the compensation values are defined, we will have m_1*m_2*m_3...m_n compensation values for each axis. This means we will have a total of n* m_1*m_2*m_3...m_n elements. These elements form a Compensation Table which is organized in the following way:

A compensation table is divided in n raws as defined by CreateComp command:

CreateComp *<Table Name> <list of dimensions for each axis>*

In our case the list would look like: m_1,m_2,m_3...m_n we call these values "***entries***".

This command will just allocate the necessary memory but will not assign any values, and the axes on which the compensation is activated are still not assigned.

The compensation values (offsets) in the table are accessible for each raw (i-th axis in this example) by:

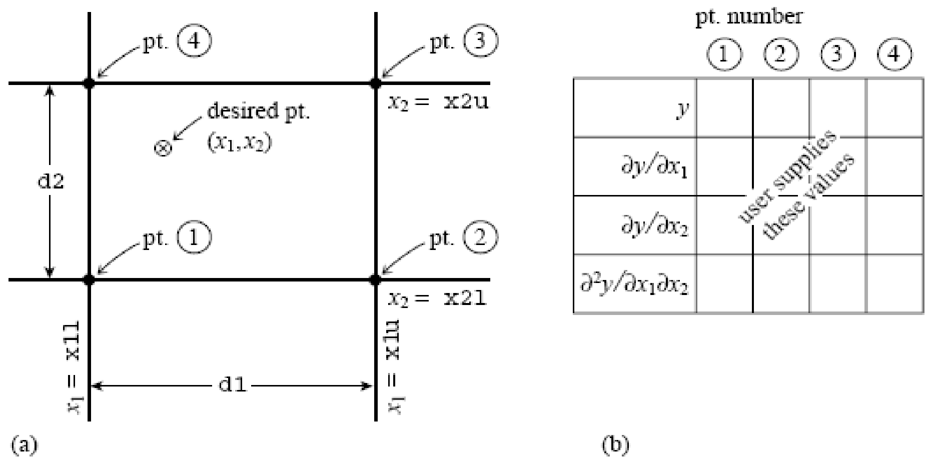

pt. 4
pt. 3
desired pt.
(x_1,x_2)
x_2 = x2u
d2
pt. 1
pt. 2
x_2 = x21
x_1 = x11
d1
x_1 = x1u
(a)
pt. number
1
2
3
4
y
$\partial y/\partial x_1$
$\partial y/\partial x_2$
$\partial^2 y/\partial x_1 \partial x_2$
user supplies these values
(b)

Figure 3.6.1. (a) Labeling of points used in the two-dimensional interpolation routines bcuint and bcucof. (b) For each of the four points in (a), the user supplies one function value, two first derivatives, and one cross-derivative, a total of 16 numbers.

<Table Name>.TargetData[i][k] = < double value>

As each raw actually represent the whole n-dimensional matrix of values it must be organized in strict manner. The values of each raw of TargetData are written as (example of two axis):

A11,A12,....,A1p, A21,A22,..............Ar1,Ar2,....Arp

this for the case of two axis first having ***p*** values and second having ***r*** values (***p, r*** are entered in CreateComp command, "***entries***").

In order to complete definition of the table we must define the compensation function domain, i.e. The source-data, the source data will be defined by pairs of min-max values for each ***entry*** of the compensation table:

<Table Name>.MinPosition[i] = ...
<Table Name>.MaxPosition[i] = ...

Strictly speaking, the compensation table defines n mappings of the form:

COMPi: $[\min_1, \max_1] \times [\min_2, \max_2] \ldots \times [\min_n, \max_n] \rightarrow \boldsymbol{R}$ or in other notation $\boldsymbol{c_i = COMP_i(x_1, \ldots x_n)}$

The $\boldsymbol{c_i}$ values are added to axis values.

Additionally the domain of compensation mapping is accessible (read-only) by auxiliary property matrix SourceTable. Different from TargetTable this does not represent the actual memory values in the table but just gives a way of checking them. The values of source table are organized in same way as target and the only difference is that they do not represent the compensation value of the axis (entry)of the given index. Note that SourceTable values are not used in computation of compensation values, they are computed on-the fly each time user queries them.

The final stop of liniking the table and the axis is done using CompSet command where the source and target list oof axes are connected via the given compensation table:

CompSet <Table Name> <source axes list> on <target axes list>

Where source and target axes list can differ but not have to.

Example

First example will be given on a compensation table of two axes a1 and a2.

First of all a compensation table is defined as global data object in the configuration file:

common shared MyTable as comp

We will spawn the table across two axes on the whole range of their position values, using a predefined step size:

```
 StepSize = 1
Size1 = (a1.pMax-a1.Pmin)/stepsize + 1
Size2 = (a2.pMax-a2.Pmin)/stepsize + 1

' Create the table
CreateComp MyTable Size1 , Size2

MyTable.MinPosition[1] = a1.PMin
MyTable.MaxPosition[1] = a1.PMax

MyTable.MinPosition[2] = a2.PMin
MyTable.MaxPosition[2] = a2.PMax
```

In the above example the distance between the points is 1 user units (deg, mm, whatever it is). Now we have a ready table to fill in. We will define a compensation values on one axis (entry only) as exponential function. As the compensation values are organized in the table as matrices first using the first index we need to enter the values using a general index *i*:

```
MyTable.TargetData[1][i] =  dAxScale * exp(-((MyTable.SourceData[1][i]-145.5)/55)^2)
```

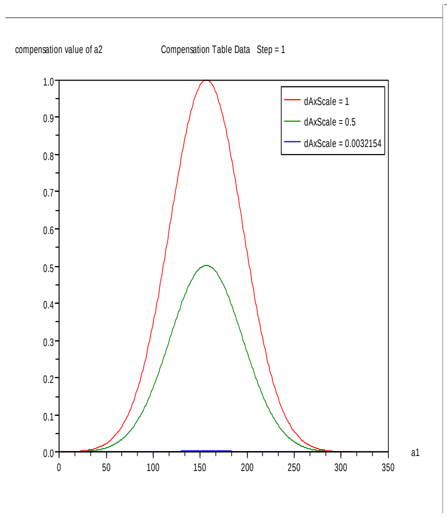
the general index i should represent the table organization so it is defined by:

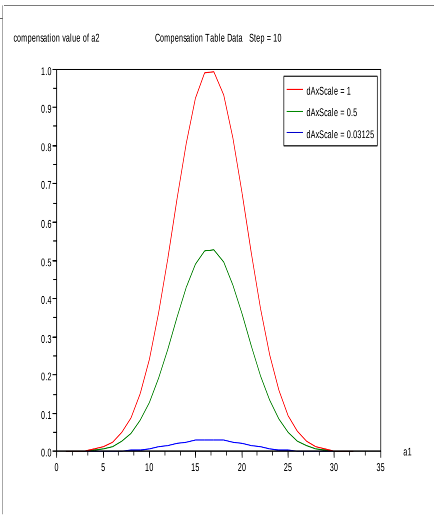
```
     i = (ind2-1)*Size1 + ind1 ' Table index
```

reading the table generated in this way on first last and middle column is done by:

```
open "Table.dat" MOde ="w" As #10
for i = 1 to Size1
  print #10,MyTable.TargetData[1][i];", ";MyTable.TargetData[1][i+int(Size2/2)*Size1];", "; MyTable.TargetData[1][i+int(Size2-1)*size1]
  sleep 10
next
close #10
```

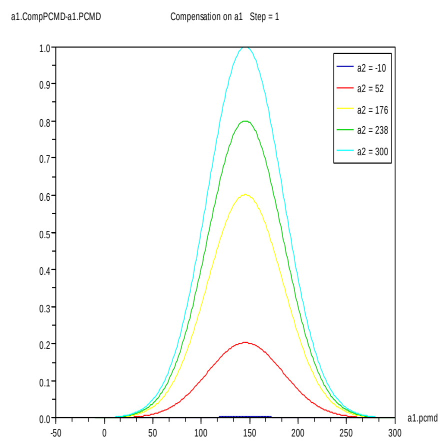
Looking at the table values for different step sizes (1,10,30) gives what we have expected:

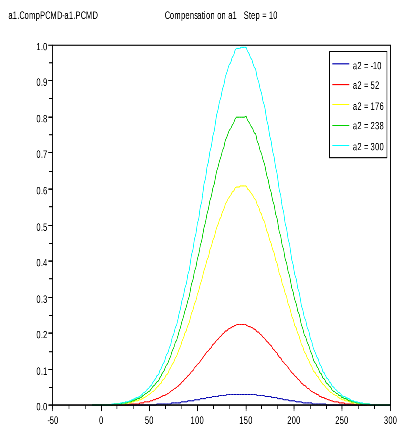


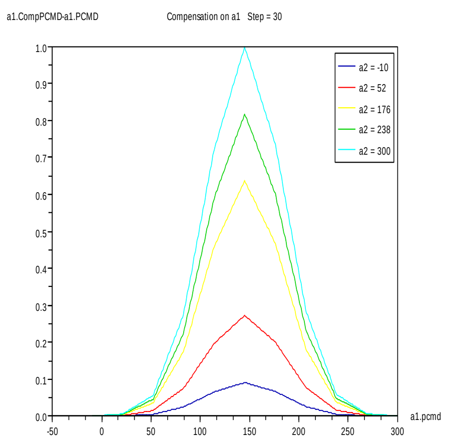


Testing the generated compensation table by moving axis a2 to positions (-10,52,176,238,300) and writing the compensation offset on axis a1 is done by:

```
for i = 1 to 5

    open str$(abs(ax2pos[i]))+".dat" MOde ="w" As #10

    move a2 ax2pos[i]  abs=1 vcruise = 20
    while a2.IsMoving
       sleep 1
    end while
    Print "At axis 2: ", ax2pos[i]

      for lIndex = round(a1.PMin) to round(a1.PMax)
        move a1 lIndex abs=1 vrate=5 arate=5 drate=5
        while (a1.IsMoving)
          sleep(20)
        end while
        print #10, a1.Pfb, a1.CompPCMD-lIndex
      next
      close #10
     Print ax2pos[i], "done .."
next
```







Looking at the results for different step sizes (1,10,30):
which are the expected results. The above example showed a combined cross compensation where the compensation values depend on both axis of the system.

Next example will show the case of simple cross compensation where one axis completely defines compensation values of the other axis:

In this case we define one-entry table as:

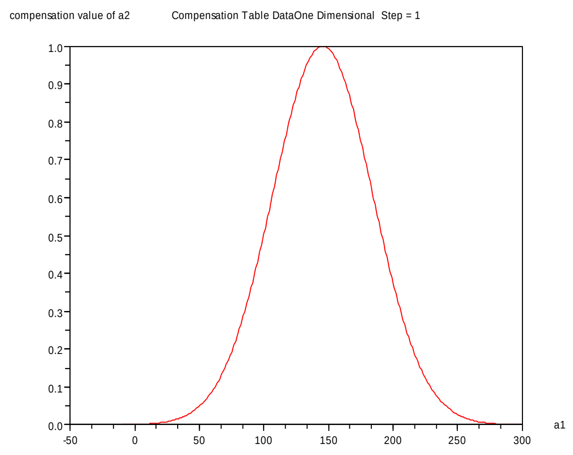
```
StepSize = 1' Choose different step sizes
Size1 = (a1.pMax-a1.Pmin)/stepsize + 1
Size2 = (a2.pMax-a2.Pmin)/stepsize + 1

MaxSize = max(Size1,Size2)

' Create the table
CreateComp MyTable MaxSize

MyTable.MinPosition[1] = a1.PMin
MyTable.MaxPosition[1] = a1.PMax
```

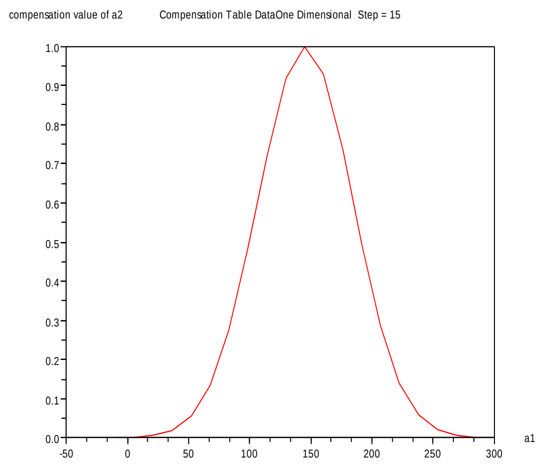
The table is computed as:

```
' Define compensation values
```

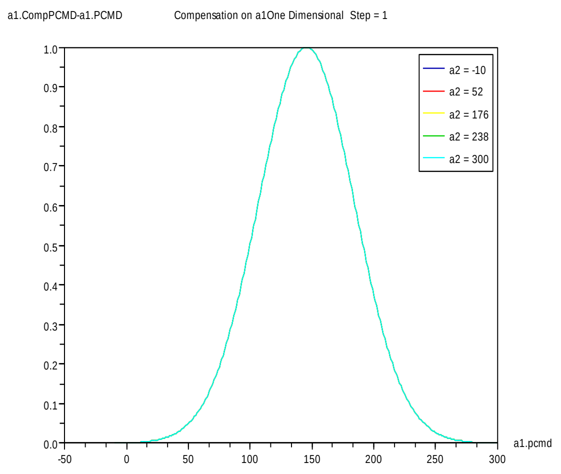
```
for i = 1 to MaxSize  ' second axis points
   MyTable.TargetData[1][i] =   exp(-((MyTable.SourceData[1][i]-145.5)/55)^2) ' Here it is OK
next
```

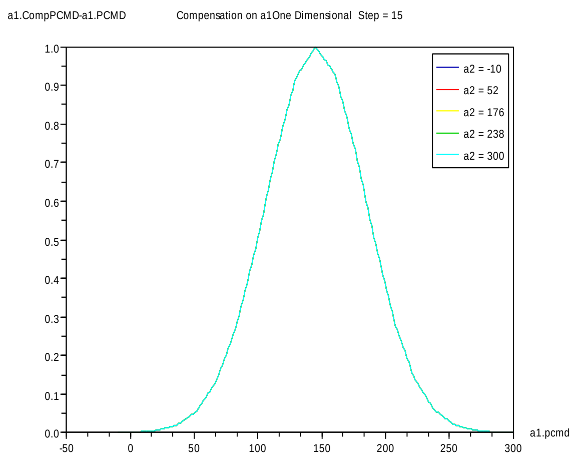
In the same way as before we read the created table values and the compensation values on the axis a2 (for steps of 1 and 15):





As the compensation values depend on one axis only we have all values overlapped:


a1.CompPCMD-a1.PCMD
Compensation on a1One Dimensional Step = 1
a2 = -10
a2 = 52
a2 = 176
a2 = 238
a2 = 300
a1.pcmd


a1.CompPCMD-a1.PCMD
Compensation on a1One Dimensional Step = 15
a2 = -10
a2 = 52
a2 = 176
a2 = 238
a2 = 300
a1.pcmd