Matlab software for
Minimum Description Length Shape Modelling
Version 2

Hans Henrik Thodberg

Draft, 25th March 2003

Abstract. This note contains information on how to use the open source Matlab implementation of the Minimum Description Length approach to Shape Modelling.

1. Introduction

This document should be read together with the paper “Minimum description Length Shape and Appearance Models”, which introduces the method and describe its basic properties.

The first thing you may want to do with the software is to check that your installation reproduces the results in the paper. This is done by running the following scripts. The indicated run times are for Matlab on a 1.2 GHz Windows PC.

Table 1: Test runs

Script	**Comment**	**Run time**
`runBoxBump.m`	runs 40 passes as in the paper	10 minutes
`runMetacarpalMode1.m`	runs 40 passes as in the paper	10 minutes
`runFemurFreeMode2.m`	runs 40 passes as in the paper	Ca. 15 minutes

Each script produces six figures. The number of passes is controlled by the variable `nPasses`. After you have made a run, you can run say 10 additional passes by setting `nPasses=10` and executing `RunExtra.m`. The information of the extra passes is then appended to the existing history in the plots.

2. How to apply the method on your data

You can run the method on your own data by writing a `runXXX` script. The easiest case is when you don’t want to use an annotated master example; then you can easily construct the run script in analogy with the script `runFemurFreeMode2` shown here:

```
clear;
InitiateMDL;

studyName='Femur Free Ends';
[C, Cn] = FormFemur;

ends = 2;
P = 64;
ps = [32 64 65   16 48   8 24 40 56];
Astart = 0.04;
Aend = 0.04;

mode = 2;
sigmaCut = 0.003;
nPasses = 40;
Run;
```

`InitiateMDL` sets default values for various variables and should always be called. `studyName` is the string used as heading on all plots. `C` and `Cn` is where you put the contours to be processed. `C` is a matrix of complex numbers: each row is a contour, i.e. the numbers are the points of the polyline that defines the contour. The number of columns in `C` corresponds to the longest of the contours, and `Cn` is a column vector with the length of each contour. The remaining entries in that row must be set to the values of the last point in the contour. Thus for contour `s` the length is `Cn(s)` and the contour is in `C(s,1:Cn(s))` and `C(s, Cn(s)+1:end)` should be equal to `C(ss, Cn(ss))`. No two subsequent points in a contour must be equal.

`ends = 2` means that it is an open shape with free end-points. `P = 64` means that we want 64 marks on the shape. P must a power of 2. Since the shape is open the actual number of marks is P+1=65. `ps` contains the list of nodes, i.e the active marks whose location are optimised for each shape individually. They are written in order of the level of nodes: 32, 64 and 65 are at the middle, end and start nodes respectively and are on level 1, 16 is between the start node and node 32, and 48 between middle and end and are on level 1, and 8, 24, 40 and 56 are on level 3.

`Astart = 0.04` indicates that we want *on average* to skip an arc length of 0.04 of the contour in the start. `Aend = 0.04` is likewise skipped in the end (recall than arc length is normalised internally to a total of one for each shape). This means that if one shape wants to start at 0.02, another must start at 0.06 etc. Thus `Astart` defines a constraint on the start node and `Aend` a constraint on the end node. There are also constraints on the remaining seven internal nodes; their node parameters should *on average* be 0.5. The constraints are implemented in the nodeCost term. Since we are running in mode 2 the constraint will not be exactly obeyed - if you want that, use mode 4.

`mode=2` was explained already, and `sigmaCut` denotes how accurate we what to model the shapes, roughly corresponding to a relative precision of 0.3%.

`nPasses = 40` is usually sufficient, but sometimes the optimasation converges after only 10 passes. If you are curious, set `nPasses = 10` and look at the plots, and then set `nPasses = 30` and execute `RunExtra`. That is equivalent to `nPasses = 40`.

The `Run` and `RunExtra` commands finish by showing some plots in figure 1-6. Figure 1 is the history of node parameters, which gives an excellent overview of the course of the 9*32 = 288 dimensional optimisation process and reveals convergence or trends. Figure 2 is the total cost, also helpful for watching for convergence. Figure 3 is the PCA at the start of the analysis. This is useful if you start the search from a configuration, which is already good, for instance a manual annotation, are definition based on some heuristic principle. You can for instance used the MDL approach to fine-tune a manual method and so it is interesting so see what the model look like at the start compared to at the end which is shown in figure 4. Figure 5 shows the break down of the MDL const function into the terms corresponding to the modes. A cost term of one corresponds to lambda equal to the cut value. Finally figure 6 shows the shapes with the MDL optimised nodes.

To summarise, if you have open curves like the femurs, replace `FormFemurs` with your code and just run. Increase `Astart` and `Aend`, if some shapes try to move beyond the end. If you have closed curves, again with no annotated master example, you would write the following for the metacarpal data:

```
clear;
InitiateMDL;

studyName='Metacarpal’;
[C, Cn] = FormFemur;

ends = 1;
P = 64;
ps = [32 64    16 48   8 24 40 56];

mode = 2;
sigmaCut = 0.003;
nPasses = 40;
Run;
```

The changes are `ends = 1` (signalling closed curve), and node 65 was omitted. Node 64 is the location of the end node, which is now also the start node. And `Astart` and `Aend` are omitted.

Running in this way with no annotation is a neutral choice: We are seeking a reparametrisation, which brings the shape in correspondence, and which are penalised if they are uneven on average in arc length. If you want the average location to be *exactly* even in arc, use mode 4.

Notice that in the metacarpal example it is assumed that all contours are oriented the same way around the object, and that the start of the contours is at approximately corresponding points across the set.

Additional illustrative experiments

A number of additional experiments can be performed with the scripts below. These experiments illustrate properties of the method and provide examples of how to run the more advanced modes.

Table 2: Other illustrative runs

Script	**Comment**	**Run time**
`runBoxBumpTrue.m`	Initiates the shapes with the ‘true’ positions of the marks and runs 10 passes. This illustrates that the MDL solution is a little different from the ’true’ positions.	2-3 minutes
`runBoxBumpRepro.m`	Runs two experiments with boxBump from different initial configurations showing that in the large `nPasses` limit the two converges to the same result (although the lower right corner of the box converges extremely slow).	10 minutes
`runMetacarpalMode3.m`	Runs 40 passes in mode 3	8 minutes
`runFemurFreeMode1.m`	Runs 40 passes in mode 1, illustrating a tendency to run-away.	Ca. 19 minutes
`runFemurFreeMode3.m`	Runs 40 passes in mode 3, giving uniform marks on average.	Ca. 16 minutes
`runFemurFreeMode4.m`	Runs 40 passes in mode 4, giving uniform marks on average.	Ca. 16 minutes
`runFemurFixed.m`	Runs 40 passes fixed endpoints. Actually it converges after 10 passes	Ca. 12 minutes

The purpose of these five runs is the following:

`runBoxBumpTrue.m` illustrates that the small deviation been the MDL solution and the ‘true’, seen at the bump is not a deficiency of the optimisation method, but a reflection of the cost function itself.

`runBoxBumpRepro.m` demonstrates how to verify that the result does not depend on the initial configuration. Such runs can be extended to a more complete study on your data, in case you want to verify to what extent your found MDL solution is unique on your data.

`runMetacarpalMode3.m` should be compared with `runMetacarpalMode1.m` which is run in mode 1. You can see that example 1 which ends up being extreme with mode 1, is more typical when run in mode 3. This shows that mode 1 introduces some unwanted bias.

`runFemurFreeMode1.m` shows that mode 1 with free ends lead to several runaway for the femur data.

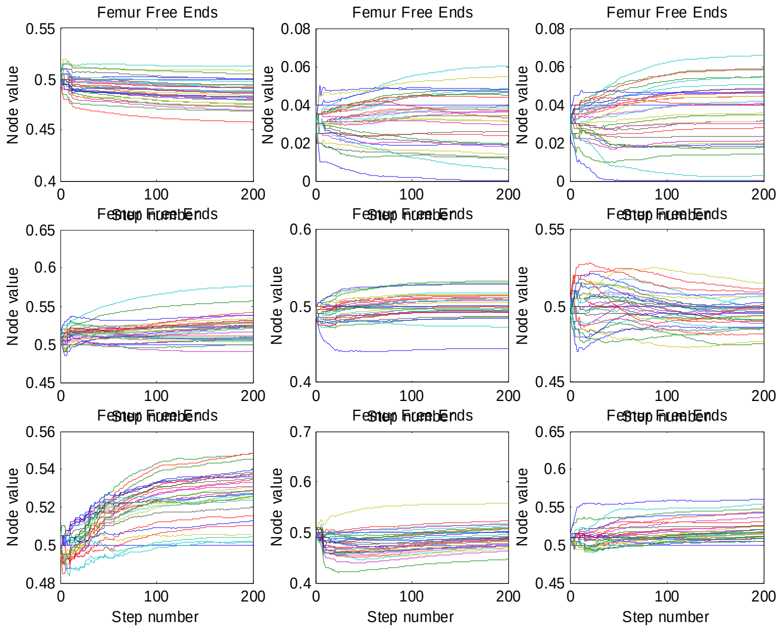


Figure 1: Mode 1: The seventh parameter (lower left) attains mean 0.525, while the master is fixed at 0.500. Likewise for the ninth**.**

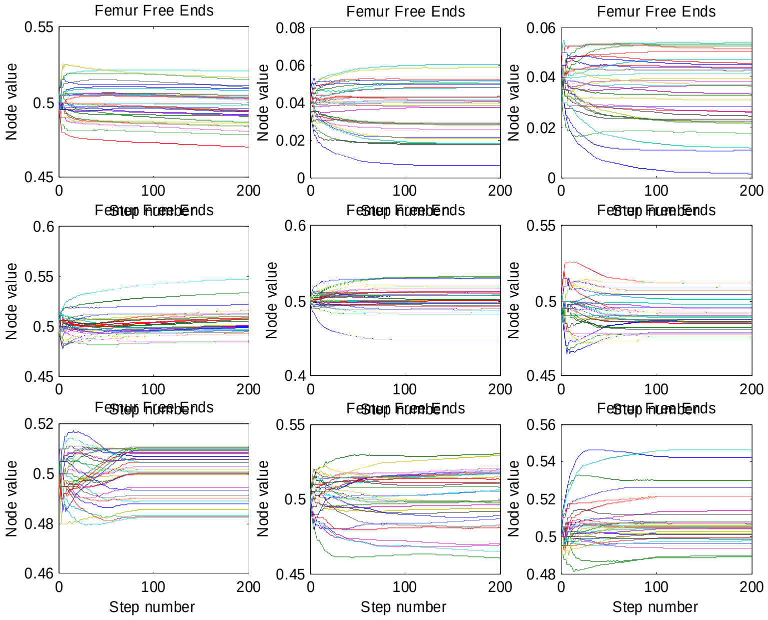


Figure 2: Mode 2: Here the averages are fixed at 0.500 the now free example 1 attains 0.488 and .496 (blue), so now its no longer mthe minimum but the 4-5 lowest

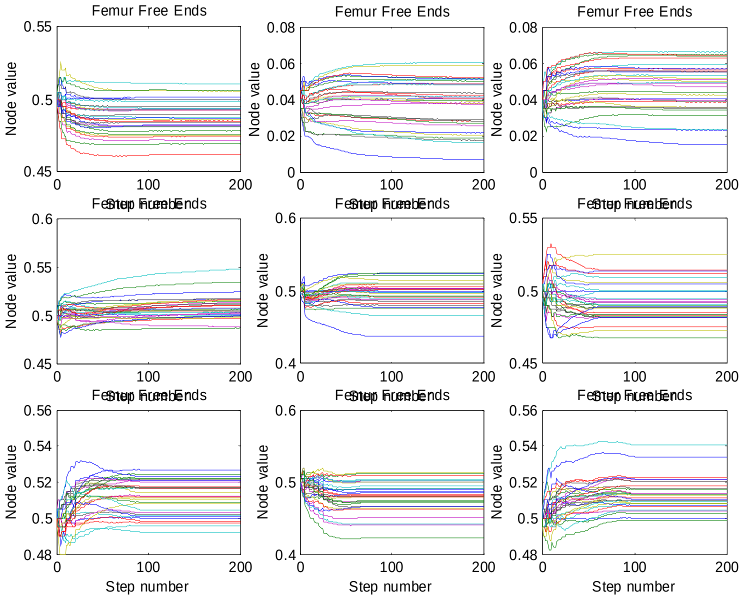


Figure 3: Mode 3

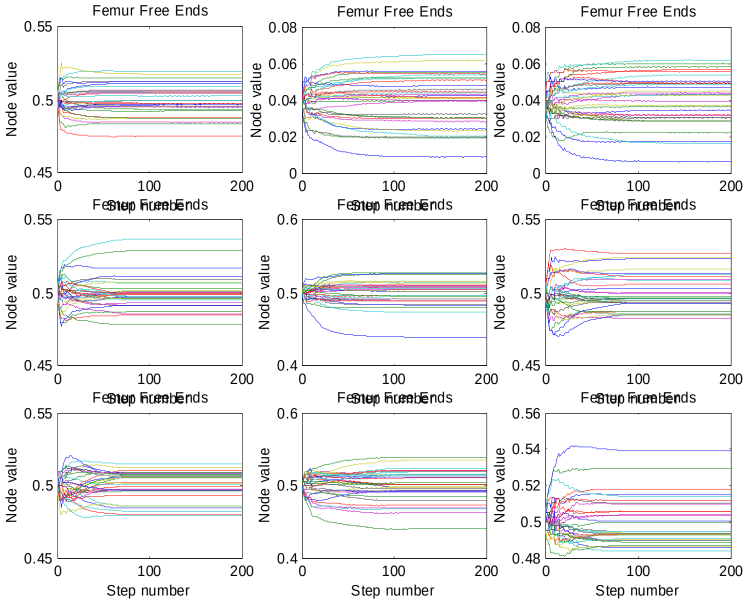


Figure 4: Mode4.

`runFemurFreeMode4.m` demonstrates mode 4, which is similar to mode 2, but the result of mode 4 are nodes uniformly in arc length on average, while this is only approximately true for mode 2.

The datasets `meta2.mat` and `fermur32.mat` are enclosed for the purpose of verifying, that the results in the paper agree with the software. You are allowed to use these data for other academic purposes without permission from the author.

3. More on the Structure of the Software

Contours are represented as polylines given by a vector of complex numbers. Notice that the imaginary axis points upwards, while most image data has the y-axis pointing downwards, so remember this when you convert data from *x*-*y* coordinates to complex numbers, i.e. use $c = x - i\,y$.

Given a contour and a chosen number of marks, e.g. 64, the mark coordinates are computed from the contour and the mark parameters, which are denoted a or A.

`Run.m` was implemented as a script in order to make it work on the workspace variables. This can be used for further analysis, plotting, and for additional runs using `RunExtra.m.`

As described in the paper, the algorithm can be run in four modes. Mode 1 uses a fixed example. This sometimes leads to run-away, and in general it leads to the unwanted effect that case 1 is treated differently from the other cases as demonstrated in `runMetacarpalMode3.m` above. Therefore, despite the nice idea of a single labelled example mode 1 is really not a lasting option of this framework, and it is included merely for historical purposes, and to illustrate why it is not desirable.

In stead of mode 1, three new modes are provided, mode 2, 3, and 4. They all rely on adding second term, the nodeCost to the MDL cost function to get a total cost function which is then minimised in the usual way. Mode 2 is the simplest. Here you simply provide the vector aTarget to be used directly in nodeCost.

Mode 3 and 4 are more sophisticated. Here aTarget in nodeCost is replaced by a dynamic target aTargetNow, which initially is aTarget, but which is adjusted at the start of every pass. The adjustment is made so as obtain one of two desired goals:

Mode 3: `aTargetNow` is adjusted so that `A(1,:) = aTarget`

Mode 4: `aTargetNow` is adjusted so that `mean(A) = aTarget`.

The central optimisation routing exists in two version: the function `Optimise.m` that supports mode 1 and Mohammed and `OptimiseCon.m` which supporting mode 2, 3, 4, which operates with various constraints on the node parameters through the node cost. There is a lot of overlap between these two routines, but the choice was made to keep each version simple.

The code was optimised for speed, and indeed it is the speed that is the most unique quality of the software. As mentioned in the paper the tree elements in bold in this pseudo code, each take 1/3 of the time.

```
Loop over passes
  Loop over nodes
    Loop over 5 steps
      Loop over examples
        Loop over + and - step
           Probe a(node) = a(node) +- step of example
           Recompute marks of example
           Do Procrustes of set
           Do PCA of set
           Compute new MDL
           If new MDL is lower accept and break loop
           Undo a(node) change
        End of +- step loop
      End of example loop
      If <20% of a(node)’s changed, divide step(node) by 2
    End of step loop
  End of node loop
End of passes loop
```

Recompute the marks involves a lot of book keeping and this would run much faster in C. Procrustes was done a complex eigenvalue problem, which allows the use of the fast build-in function and comparable to a C-version. Likewise the PCA is close to optimal.

Another reason for the speed is the hierarchical node structure, where only the 8 highest level nodes are optimised, while the remaining 56 are uniformed in arc length.

An overview of the run script parameters are tabulated in table 3; a description was also given above in relation to the femur run

Table 3: Parameters of the optimisation script Run

Parameter	Explanation
`studyName`	String used as title of plots
`C  Cn`	The contours are the complex rows of `C`, `Cn` is the length of each contour, i.e. contour r is in `C(r,1:Cn(r))`
`ends`	`ends=0`: open curve with fixed ends `ends=1`: closed curves `ends=2`: open curve with free ends
`P`	Granularity (or resolution) of shape: The shape is a sampling of the curve, The sampling points are called marks Closed curves have `P` sampling point. `P/2` and `P` are on the top level. Open curves have `P+1` marks `P/2`, `P` and `P+1` are the middle, end and start respectively.
`ps`	`ps` are List of nodes gives as a vector of the number of the marks.
`mode`	Mode=1,2,3 or 4 controls the use of templates and Node-Cost, see the text
`sigmaCut`	The level at which shape information becomes irrelevant
`nPasses`	The number of passes used for the optimisation

If you want to provide a single example where the node positions are true, to act as matseer you should used the parameters in table 4. The true position of a mark can only be used for marks of the top levels. The master must be example 1. In the example metacarpal and boxBump the nodes on level 1, 2 and 3 where fixed for the master example. One could allow fixing only part of the nodes on level 3, but all the nodes at the higher levels should be fixed. The fixing is done by letting the contour have points at the nodes. SO if point 29 on contour 1 is fixed

Table 4: Additional parameters used to define annotated examples (the script InitiateMDL sets `useMasterNodeValues=0` i.e. the default is not to use this feature)

Parameter	Explanation
`useMasterNodeValues`	1 indicates that `MasterNodeValues` are to be used.
`MasterNodeValues`	List of the location of the nodes on the master given as a vector of point numb3rs in the contour, see text
`initAll`	1 if all the shapes should be initialised according to `MasterNodeValues`

There are two examples of this in from the paper: In the `boxBump` we know that the true points are `MasterNodeValues = [1 2 3 4 5 6 7 25 26];`

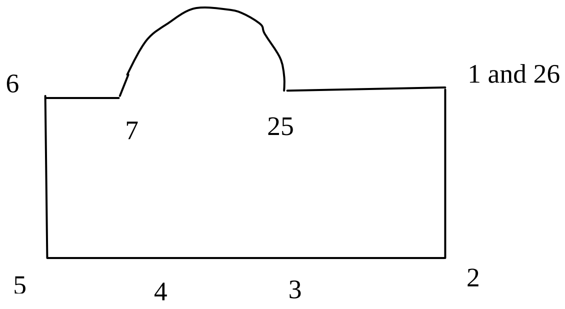
As shown in figure …

And in the metacarpal case the contours are using a heuristic tatdraiatoin along a bone axis and the truth according to these is given by `MasterNodeValues = [1 31 71 111 141 171 211 251 281];` In the metacarpal case the optimisation is in fact started using exactly this alignment, so all the shapes are started here, as signalled by the `initAll=1`. `InitAll=1` is also used in `runBoxBumpTrue` above. The default value of `initAll` is 0 and is set by `InitiateMDL`.

```
P = 64;
ps = [32 64  16 48  8 24 40 56]; % 64 is the same as 0
useMasterNodeValues = 1;
MasterNodeValues = [1 2 3 4 5 6 7 25 26];

P = 64;
ps = [32 64  16 48  8 24 40 56];
useMasterNodeValues = 1;
MasterNodeValues = [1 31 71 111 141 171 211 251 281];
initAll = 1;
```

When you are using a master example the way to convey the information about the true location of the marks on the master example is through the parameters in table 4. The true locations must be among the nodes, and the locations must be among the points defining the contours. For example the `boxBump` contours are defined as in the following figure, using 26 points where 1 and 26 are at the same location. 8 of these points are selected as nodes of the master. They are listed i.e. in the `runBoxBump` below node: 64 8 16 24 32 40 48 56 are at `C([,1 2 3 4 5 6 7 25 26]).`



The `runBoxBump` script creates 24 random shapes according to the parameters given to `FormBoxBump`. The last parameter 0.7 controls how much the bum's position varies horizontally, the second the vertical box side length (the first parameter is not used). Thus the figures are created with two degrees of freedom, so we expect only two eigenvalues to be significant. The random generator is initialised in the routine so you should obtain the same data when running it.