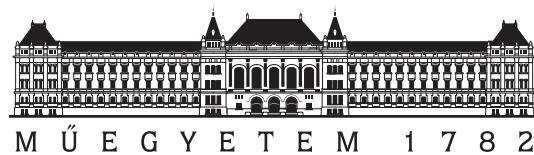


# MDBM

## Multi-Dimensional Bisection Method

User's Guide  
Fast computation of all roots of a non-linear function for  
higher parameter dimensions and co-dimensions.



Daniel Bachrathy  
Department of Applied Mechanics  
Budapest University of Technology and Economics

Numerical methods made by

*DB products*  
*(Digital Butterfly)*

2014

### **Acknowledgement**

This research was supported by the European Union and the State of Hungary, through a European Social Fund in the framework of TÁMOP 4.2.4. A/1-11-1-2012-0001 "National Excellence Program".

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Introduction . . . . .	1
<b>2</b>	<b>Program description</b>	<b>2</b>
2.1	Required form of the function . . . . .	2
2.2	The Developed Matlab algorithm . . . . .	3
2.3	Input parameters of <code>db_mdbm</code> file . . . . .	3
2.3.1	Definition of the parameter space - <code>ax</code> . . . . .	3
2.3.2	Definition file of the functions - <code>bound_fuction_name</code> . . . . .	4
2.3.3	Number of the iterations - <code>Niteration</code> . . . . .	4
2.3.4	Additional parameters - <code>par</code> . . . . .	4
2.3.5	Additional parameters - <code>triangulation</code> . . . . .	5
2.4	The system files <code>fval_my_nonlin_system</code> file . . . . .	5
<b>3</b>	<b>Examples</b>	<b>7</b>
3.1	Basic Examples . . . . .	7
3.1.1	1-parameter function, co-dimension 1 . . . . .	7
3.1.2	2-parameter function, co-dimension 1 . . . . .	9
3.1.3	2-parameter function, co-dimension 2 . . . . .	10
3.1.4	3-parameter function, co-dimension 1 . . . . .	11
3.1.5	3-parameter function, co-dimension 2 . . . . .	12
3.1.6	3-parameter function, co-dimension 3 . . . . .	13
3.2	Plotting options . . . . .	14
3.2.1	outputs . . . . .	14
3.2.2	inputs . . . . .	14
3.3	Complex example . . . . .	18
3.3.1	Mandelbrot-set . . . . .	18
3.3.2	Catastrophe surface . . . . .	20
3.3.3	Stability of the turning process with traditional process damping	21
3.3.4	Unite norm . . . . .	23
3.3.5	Section of surfaces, solution of transcendent equation, error calculation . . . . .	24
<b>References</b>		<b>26</b>

# Chapter 1

## Introduction

### 1.1 Introduction

The bisection method – or the so-called interval halving method – is one of the simplest root-finding algorithms which is used to find zeros of continuous non-linear functions. This method is very robust and it always tends to the solution if the signs of the function values are different at the borders of the chosen initial interval.

Geometrically, root-finding algorithms of  $f(x) = 0$  find one intersection point of the graph of the function with the axis of the independent variable. In many applications, this 1-dimensional intersection problem must be extended to higher dimensions, e.g.: intersections of surfaces in a 3D space (volume), which can be described as a system on non-linear equations. In higher dimensions, the existence of multiple solutions becomes very important, since the intersection of two surfaces can create multiple intersection curves.

In this User's Guide, I give a description of the Matlab package of the Multi-Dimensional Bisection Method.

The theoretical details and the flowchart of the computation is presented in [1]. If you use this code, please, cite my work!

Important notice: users of MDBM are assumed to have a basic knowledge about Matlab programming. The manual on the package does not replace the corresponding text books!

# Chapter 2

## Program description

### 2.1 Required form of the function

The numerical method is implemented to determine the roots of the function given in the following form

$$\mathbf{f}(\mathbf{x}) = \mathbf{0}, \quad (2.1)$$

in other form

$$\begin{aligned} f_1(p_1, p_2, \dots, p_i, \dots, p_{N_{dim}}) &= 0, \\ f_2(p_1, p_2, \dots, p_i, \dots, p_{N_{dim}}) &= 0, \\ &\vdots \\ f_C(p_1, p_2, \dots, p_i, \dots, p_{N_{dim}}) &= 0, \end{aligned} \quad (2.2)$$

where  $\mathbf{p}$  is the general coordinate vector (the independent parameter vector), with general variable  $i = 1, 2, \dots, N_{dim}$ . The number of equations, which defines the co-dimensions  $C$  must be smaller or equal than the parameter dimensions  $N_{dim}$ , so  $C \leq N_{dim}$ . Note, that the  $N_{dim} - C$  value defines the dimension of the solutions (E.g.: 3 parameter -  $N_{dim} = 3$  with two equation  $C = 2$ , defines curves  $N_{dim} - C = 1$ ).

The MDBM is efficient if  $N_{dim} - C = 1$  or  $2$ . In case of  $N_{dim} - C = 0$ , fine initial mesh is necessary in some problems.

The MDBM is designed and optimized for  $N_{dim} = 2, 3$ . Higher values for  $N_{dim}$  ( $4, 5, 6$ ) and  $N_{dim} - C$  ( $3, 4$ ) can also be used, however, the computational time and the memory consumption of the MDBM is much larger in these cases. Theoretically the MDBM method is programmed for any high values of the  $N_{dim}$  and  $C$ .

## 2.2 The Developed Matlab algorithm

### Program files

The db\_mdbm file contains all necessary code for the computation, which can be called as follows:

```
outputsol  
=db_mdbm(ax,bound_fuction_name,Niteration,par,triangulation);
```

The output is a struct which contains the interpolated roots in outputsol.posinterp, and the corresponding gradients of the functions outputsol.gradient, the triangulation in DTalphashapecell in different levels (if the input triangulation=true), and many other variables (which can be used for development purpose).

## 2.3 Input parameters of db\_mdbm file

**ax** - (struct) description of the parameter region

**bound\_fuction\_name** - (string) name of the 'system files'

(e.g: bound\_fuction\_name='fval\_my\_nonlin\_system') which evaluates the functions.

**Niteration** - (integer  $\geq 1$ ) number of the iteration halving along all the parameter dimension

**par** - (struct) additional parameters for the computation (optional)

**triangulation** - (logical) if it is true, then the determined point cloud is triangulated (optional)

### 2.3.1 Definition of the parameter space - ax

First of all, lets assume that the parameter vector **p** determines an n-cube with  $N_{dim}$  dimension ( $N_{dim} = 2$  - rectangle,  $N_{dim} = 3$  - "cube") that is: **size(p) = N<sub>dim</sub>**.

One input of the method is an initialized (coarse) net within this n-cube. Its resolution must be defined along each direction (dimension) and must be paced in a structure **ax(i).val** where  $i = 1, 2, \dots, N_{dim}$ . It is typically given as follows:

```
ax(1).val = linspace(p1min,p1max,Np1);  
ax(2).val = logspace(9,10,10);  
ax(3).val = (0.1 : 0.1 : 2).^(−1);
```

 (2.3)

Note in the example above, that it is not necessary to have a uniform linear mesh. In some case it is very useful to use some power of a linear grid as given in the third

line of the equation above. Typically, the length of these vectors are between 6 and 30, depending on the problem.

### 2.3.2 Definition file of the functions - bound\_fuction\_name

The function values  $f_1, f_2, \dots, f_C$  must be defined by the user in a separate Matlab function file (\*.m). The string `bound_fuction_name` must contains the name of this file, e.g.: `bound_fuction_name='fval_my_nonlin_system'`;

or

`bound_fuction_name='fval_complex_3_turning_stability'`;

In the later case, this m-file must be called as follows:

```
f= fval_complex_3_turning_stability(ax);
```

or

```
f= fval_complex_3_turning_stability(ax,par);
```

The `par` can contain additional parameters if it is provided in the `db_mdbm` function.

Details of the input and output parameters of this file will be discussed later.

It can be an in-line function, too e.g.:

```
bound_fuction_name=@(x) sin(x(1,:))+cos(x(2,:)),
```

however, a non-linear system is usually described in a more complex way and it is better to place it in a separate m-file.

### 2.3.3 Number of iterations - Niteration

The Multi-Dimensional-Bisection-Method needs the number of the iteration `Niteration` as a parameter. Note, that `Niteration = 1` means, that only the coarse mesh will be analysed. Typical values of  $N_{iteration}$  are 4-5-6-(7-8-9) for 2 parameter problems ( $N_{dim} = 2$ ) and 2-3-(4) for 3 parameter problems ( $N_{dim} = 3$ ). Note, that the memory consumption of the computation increases exponentially as a function of  $N_{iteration}$ :

$$\sim const \times 2^{(N_{dim}+1)N_{iteration}}.$$

E.g.: for a 3 parameter problem with 4 iteration ( $N_{dim} = 3, N_{iteration} = 4$ ) it is  $2^{16} = 65\,536$  times the memory consumption of the computation along the initial mesh only.

Some memory management is built into the numerical method, but not in each steps. So, for a new problem, the `Niteration` should increase step-by-step while the memory consumption of the method is monitored, otherwise Matlab (and/or the operation system) might collapse (or simply will not respond).

### 2.3.4 Additional parameters - par

In many situation extra (constant) parameters are necessary for the computations. Instead of using global variables, these variables can be placed in this `par` variable, typically as a struct. E.g. the constant parameters of a dynamic system can be placed in a struct and provided to the `db_mdbm` file, and then it can be reached in the `'fval_my_nonlin_system'` file as follows:

```

par.mass=2;
par.damping=0.1';
par.stiffness=10';
:
outputsol=db_mdbm(ax,'fval_my_nonlin_system',4,par);
and the file is defined as follows:
function fvals=fval_my_nonlin_system(ax,par)
s=par.stiffness;
:

```

This `par` file can be updated during the computation if necessary, the files should be formed as follows:

```

function [fvals,parout]=fval_my_nonlin_system(ax,par)
:
par.A(end+1)=...;
parout=par;
:
```

### 2.3.5 Additional parameters - triangulation

The output of the MDBM method is basically a point clouds. If `triangulation=true`, then the point cloud is triangulated to be plotable as a line or surface, however, in case of larger  $N_{dim} (>3)$  it could take a very long time. If only the point cloud should be presented or during the testing phase of the actual problem, it is not necessary to compute the triangulation `triangulation=false`.

Note, that the default value is `triangulation=true`. It is also important to note, that the triangulation is problematic at the borders of the parameter space, and in case of close objects (like close perpendicular lines or intersecting lines).

## 2.4 The system file: `fval_my_nonlin_system`

This function is provided by the user, which gives the values of the functions of the specific problem.

This file must be called as e.g.:

```
[f,paramout] = fval_my_nonlin_system(ax,par),
```

where the inputs `ax` represent a bunch of (unordered) points (with number  $N_p$ ) in the selected 'parameter' region/domain. `ax` is a matrix with `size(ax) = [Ndim, Np]`, where  $N_{dim}$  is the dimension of the parameter region (see Subsec. 2.3.1). For the  $n^{th}$  point:

$$\begin{aligned}
p_{1,n} &= \mathbf{ax}(1, n) \\
&\dots \\
p_{i,n} &= \mathbf{ax}(i, n) \\
&\dots \\
p_{N_{\text{dim}},n} &= \mathbf{ax}(N_{\text{dim}}, n).
\end{aligned} \tag{2.4}$$

The input `par` is optional and it can contain some further parameters computed by the user (or necessary for the user). If it is updated during the computation than the new values should be place in the output `paramout` (it is also optional).

The output of this function must be a matrix `size(f) = [C, N_p]`. In the columns, the function values are placed  $f_1, f_2, \dots, f_C$ . Each row refers to each row of the parameter values of the `ax` variable. (See the realization in the basic example files `fval_basic_X_pardimX_codimX.m`, where different types of evaluation are presented.

# Chapter 3

## Examples

Many type of examples are given with their corresponding Matlab m-files, which can be used as a base for a new problem.

The examples are described from the simplest problem to more complex ones. The first Section presents very basic examples to help understanding the definition of the parameter space and the number of co-dimensions. The second Section presents the plotting possibilities and special presentation forms of the output solution of the MDBM method. In the third Section, some complex examples where a complicated problem is solved with special functions and special plotting options.

### 3.1 Basic Examples

These simple examples present computations for 1 to higher dimensional problems with 'default' computation and plotting options. The first few examples are described here, with the corresponding functions. All the other functions and calculations can be found in the following files:

```
run_test_basic_9_pardim4_codim3.m  
fval_basic_9_pardim4_codim3.m
```

where the number after the `pardim` refers to the number of the independent parameter variables, and the number after the `codim` refers to the one of co-dimensions which is equal to the number of equations.

The `fval_basic_...` function usually contains many different solutions to evaluate the same function values. All the commented versions lead to the same results. The vectorial format is very fast, however, it can not be used in every situation. The `for` cycle is slow but it can be used in any case (see 3.3.1).

#### 3.1.1 1-parameter function, co-dimension 1

```
run_test_basic_1_pardim1_codim1.m  
fval_basic_1_pardim1_codim1.m
```

The roots of a sinusoidal function are determined.

$$f_1(x) = \sin x = 0. \quad (3.1)$$

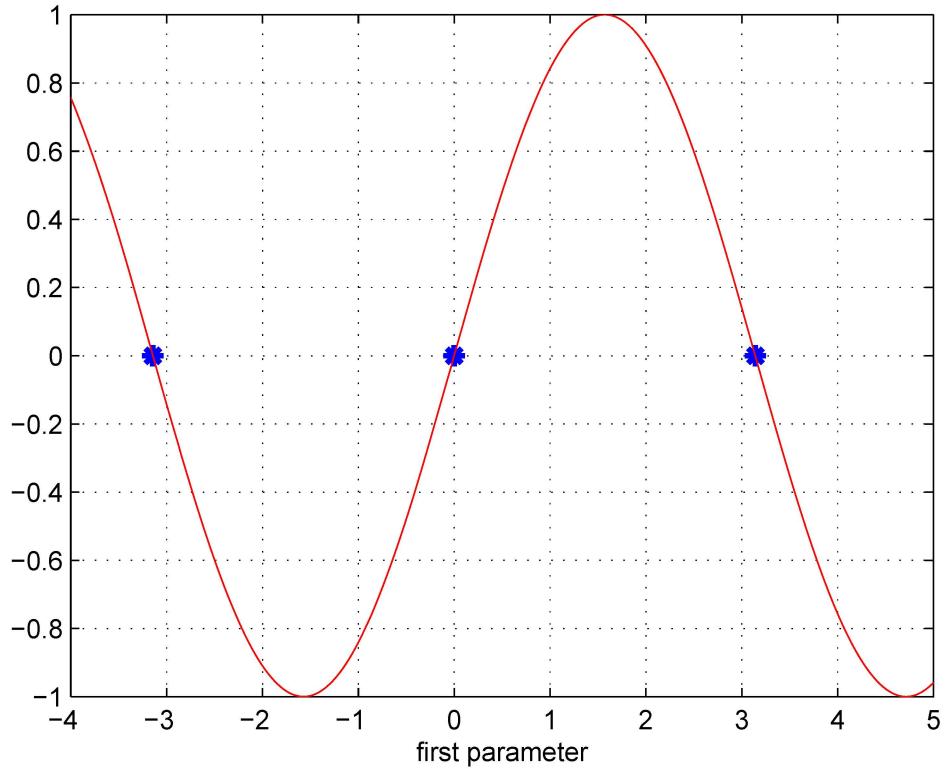


Figure 3.1: Roots of the sin function.

### 3.1.2 2-parameter function, co-dimension 1

```
run_test_basic_2_pardim2_codim1.m  
fval_basic_2_pardim2_codim1.m
```

Finding the roots of a two parameter function, which determines a circle with radius  $R = 2$ , defined by the following equation

$$f_1(x, y) = x^2 + y^2 - R^2 = 0. \quad (3.2)$$

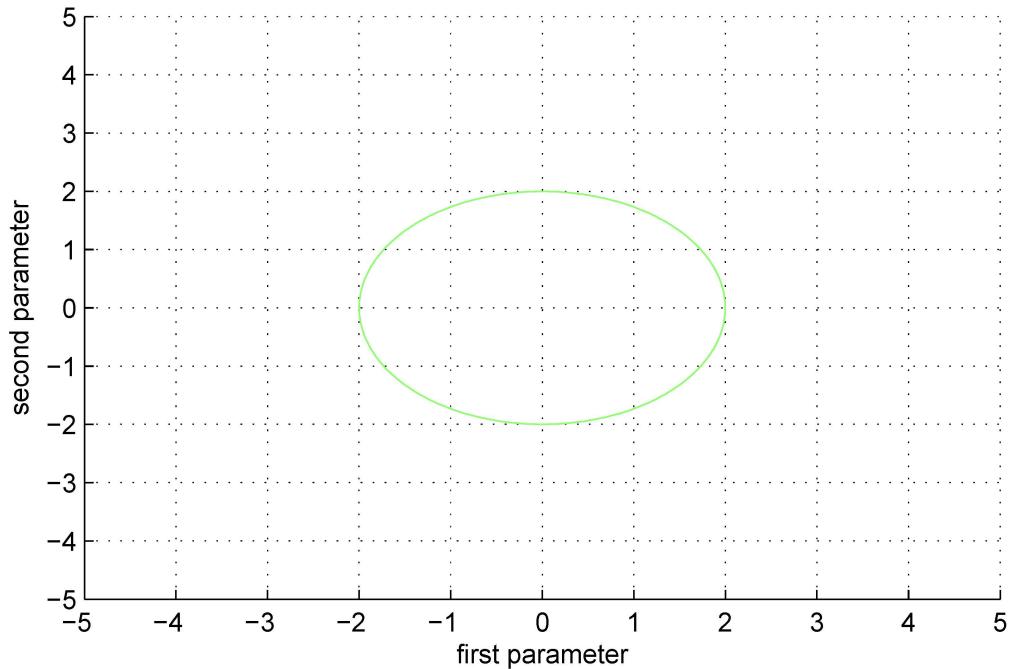


Figure 3.2: The set of roots forms a circle.

### 3.1.3 2-parameter function, co-dimension 2

```
run_test_basic_3_pardim2_codim2.m
fval_basic_3_pardim2_codim2.m
```

Finding the roots of a two parameter functions. Now the function is vector valued; it is defined by the two following scalar equations

$$\begin{aligned} f_1(x, y) &= x^2 + y^2 - R^2 = 0, \\ f_2(x, y) &= x - y = 0. \end{aligned} \quad (3.3)$$

where  $f_1$  is a circle as defined in the previous subsection and  $f_2$  is a straight line  $x = y$ .

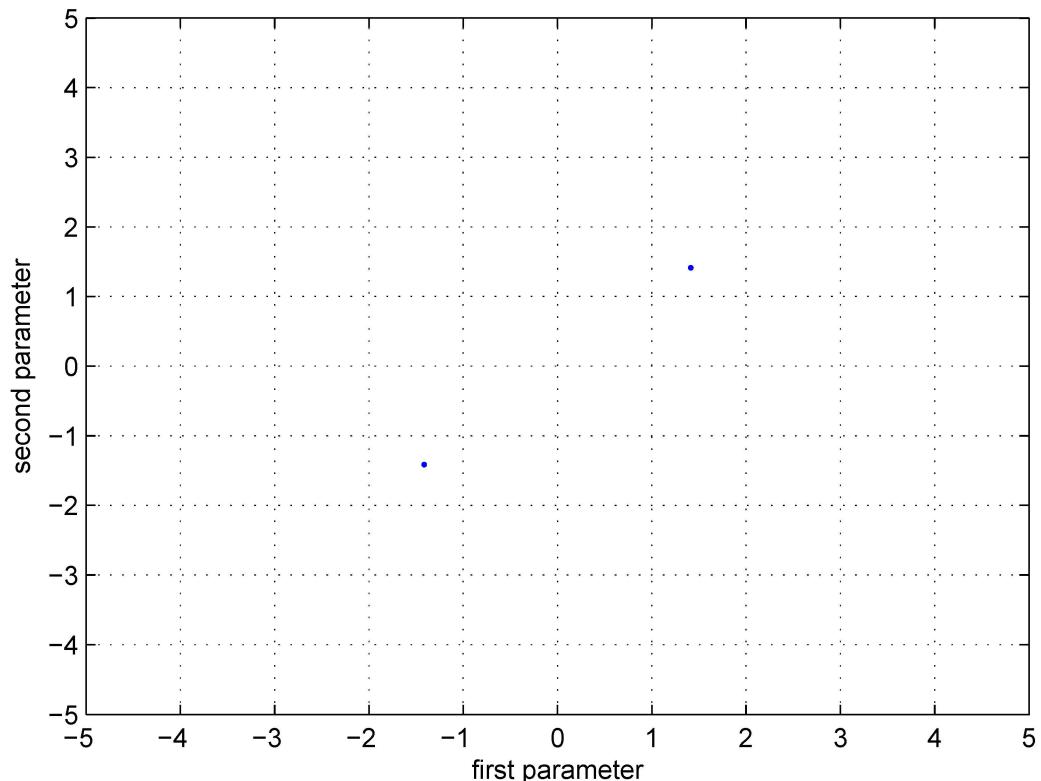


Figure 3.3: The roots of the equations.

### 3.1.4 3-parameter function, co-dimension 1

run\_test\_basic\_4\_pardim3\_codim1.m

fval\_basic\_4\_pardim3\_codim1.m

The function describe a sphere with radius  $R = 2$ .

$$f_1(x, y, z) = x^2 + y^2 + z^2 - R^2 = 0, \quad (3.4)$$

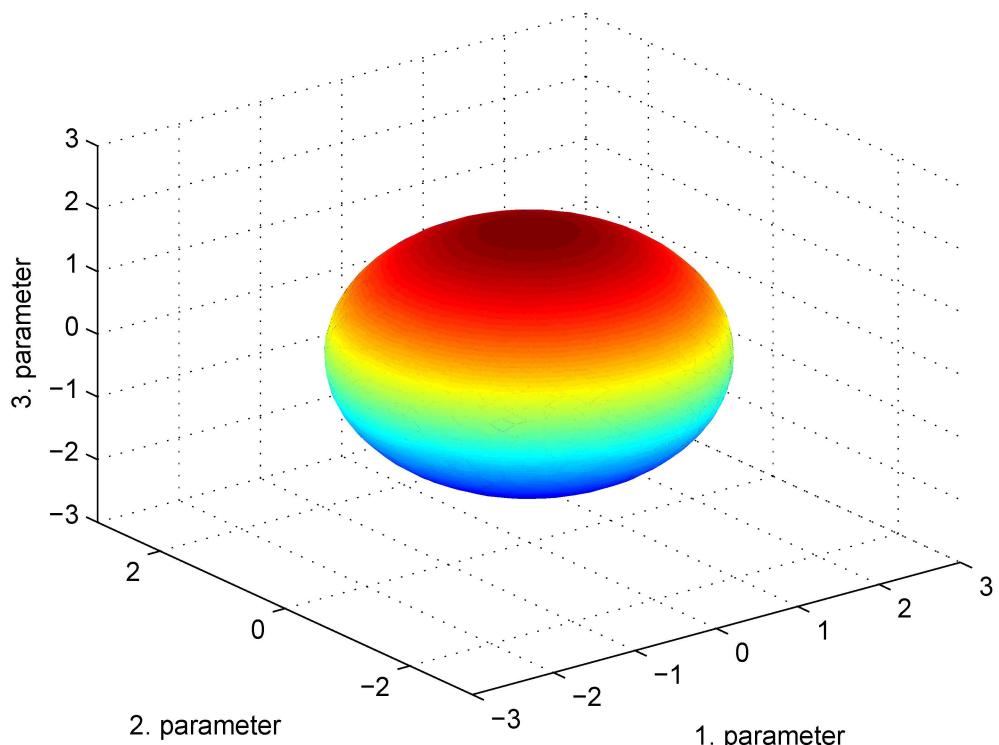


Figure 3.4: The roots of the equations.

### 3.1.5 3-parameter function, co-dimension 2

```
run_test_basic_5_pardim3_codim2.m
fval_basic_5_pardim3_codim2.m
```

We have the sphere as before and we have a plain where  $x = y$ . Now the function is vector valued. We have 3 independent parameter variables and 2 equations, so the resultant objects are curves (here a single circle).

$$\begin{aligned} f_1(x, y, z) &= x^2 + y^2 + z^2 - R^2 = 0, \\ f_2(x, y, z) &= x - y = 0. \end{aligned} \quad (3.5)$$

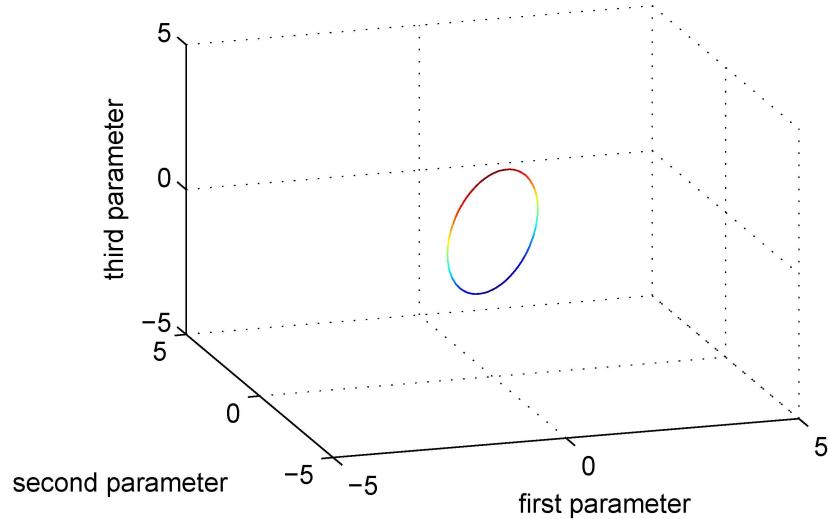


Figure 3.5: The roots of the equations.

### 3.1.6 3-parameter function, co-dimension 3

```
run_test_basic_6_pardim3_codim3.m
fval_basic_6_pardim3_codim3.m
```

We have the sphere as before and now we have two plains where  $x = y$  and  $x = -z$ . Now the function is vector valued, too. We have 3 independent parameter variables and 3 equations, so the resultant objects are points.

$$\begin{aligned} f_1(x, y, z) &= x^2 + y^2 + z^2 - R^2 = 0, \\ f_2(x, y, z) &= x - y = 0. \\ f_3(x, y, z) &= x + z = 0. \end{aligned} \quad (3.6)$$

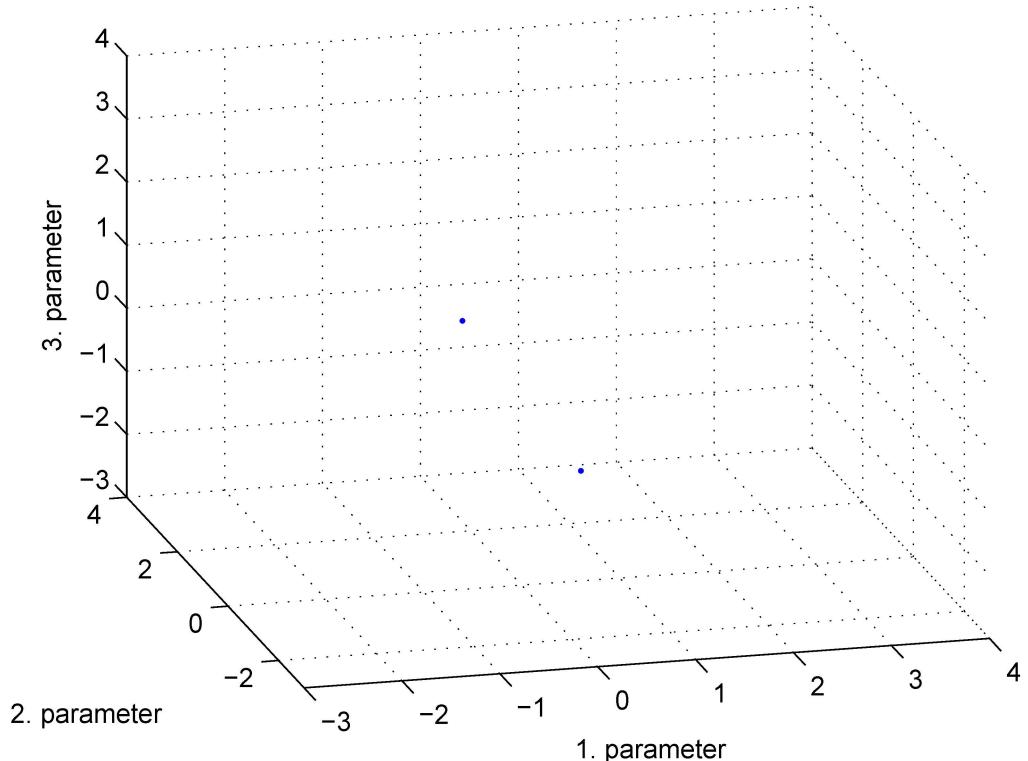


Figure 3.6: The roots of the equations.

## 3.2 Plotting options

The output structure variable of MDBM contains all the information you need to analyse and plot the results yourself, however, an easy-to-use plotting function (`db_plot_mdbm.m`) is provided, which can be used for most of the situation. See the help of this file (`help db_plot_mdbm.m`).

The examples described in the `run_test_plotting_exampels.m` file are easy to understand.

The function `db_plot_mdbm.m` can be called as:

```
graphobjects=
db_plot_mdbm.m(outputsol,plotcolor,dimensionsorder,plotobjdim,gradplot)
```

### 3.2.1 outputs

`graphobjects` contains the object handles of the plotted graphical objects. You can used them to modify their properties e.g. like line width and edge color as follows:

```
set(graphobjects,'LineWidth',3)
set(graphobjects,'Color',[1,0.4,0])
```

See the detailed examples in the `run_test_plotting_exampels.m` file.

### 3.2.2 inputs

Only the first input is necessary, all the others can be omitted, or suppressed by an empty value [] .

The `outputsol` is simply the output structure of the `db_mdbm` file.

The `plotcolor` can define the color of the plotting object, which can be a 'color representing string' (like.: '`k`' -black, '`b`' - blue, '`r`' - red, ...) or an RGB value (e.g: [0.1,0.5,0.7]). The RGB values can not be used for points.

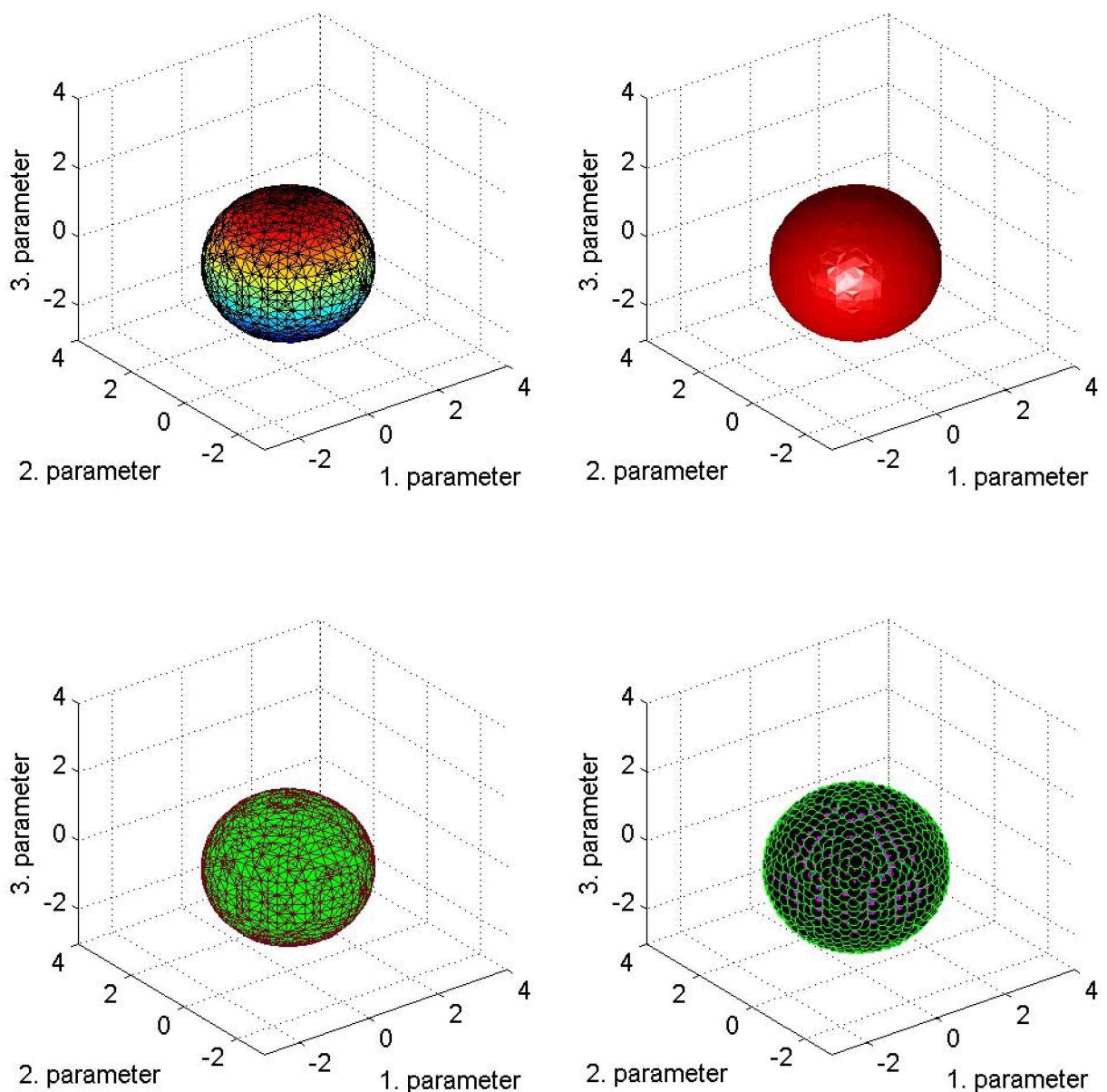


Figure 3.7: Plotting the results in different colors

The **dimensionsorder** can change the order of the parameters in the plot. In case of [3,1,2], it will plot variable 3 along the  $x$  axis, variable 1 along the  $y$  axis, and variable 2 along the  $z$  axis.

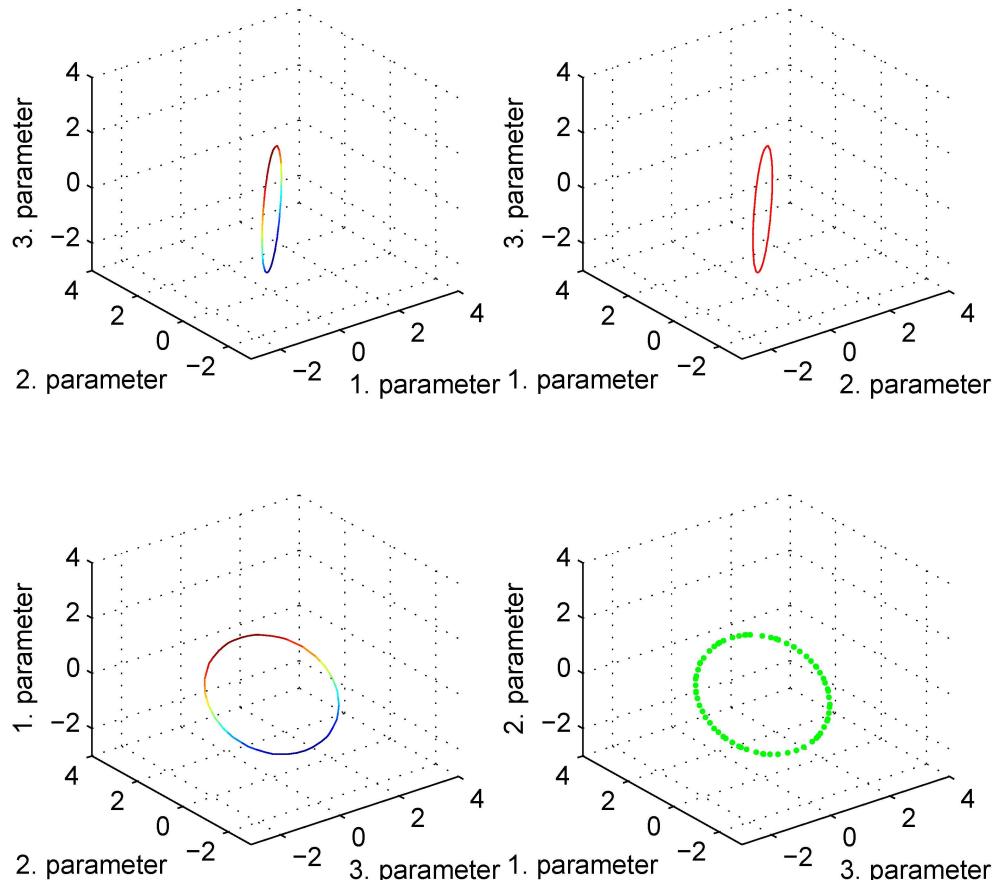


Figure 3.8: Chaging the order of the variables (and the color representation)

In the **plotobjdim**, you can specify your preference for plotting:

- 0 - as a point cloud (or points).
- 1 - as lines.
- 2 - as surfaces.

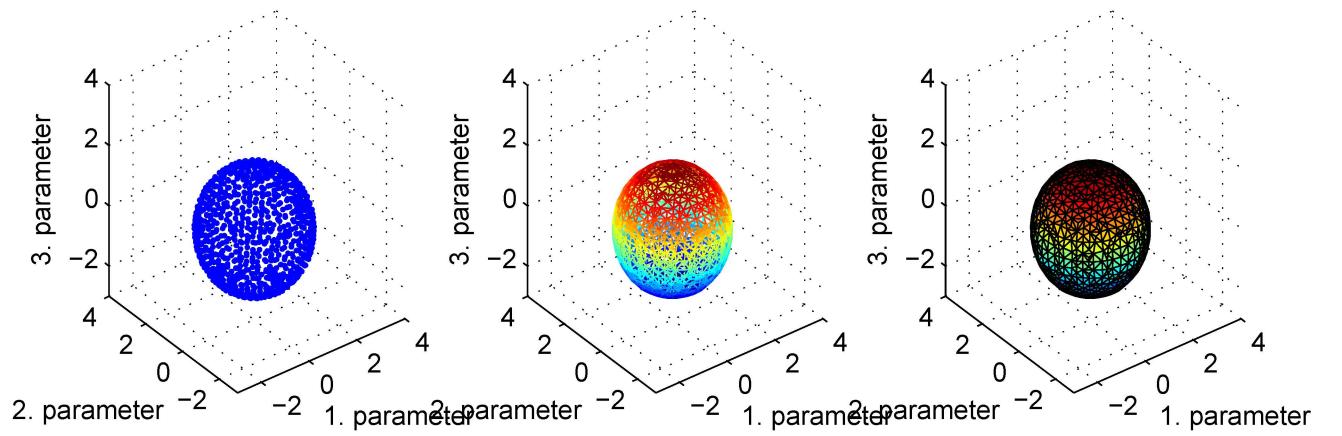


Figure 3.9: Plotting the results with as a point cloud, as lines or as a surface

The `gradplot` (logical value: true (1) or false (0)). If it is true (non-zero value) you can plot the numerically approximated gradient of the functions (which is a side results of the MDBM) as a vector-field along the roots.

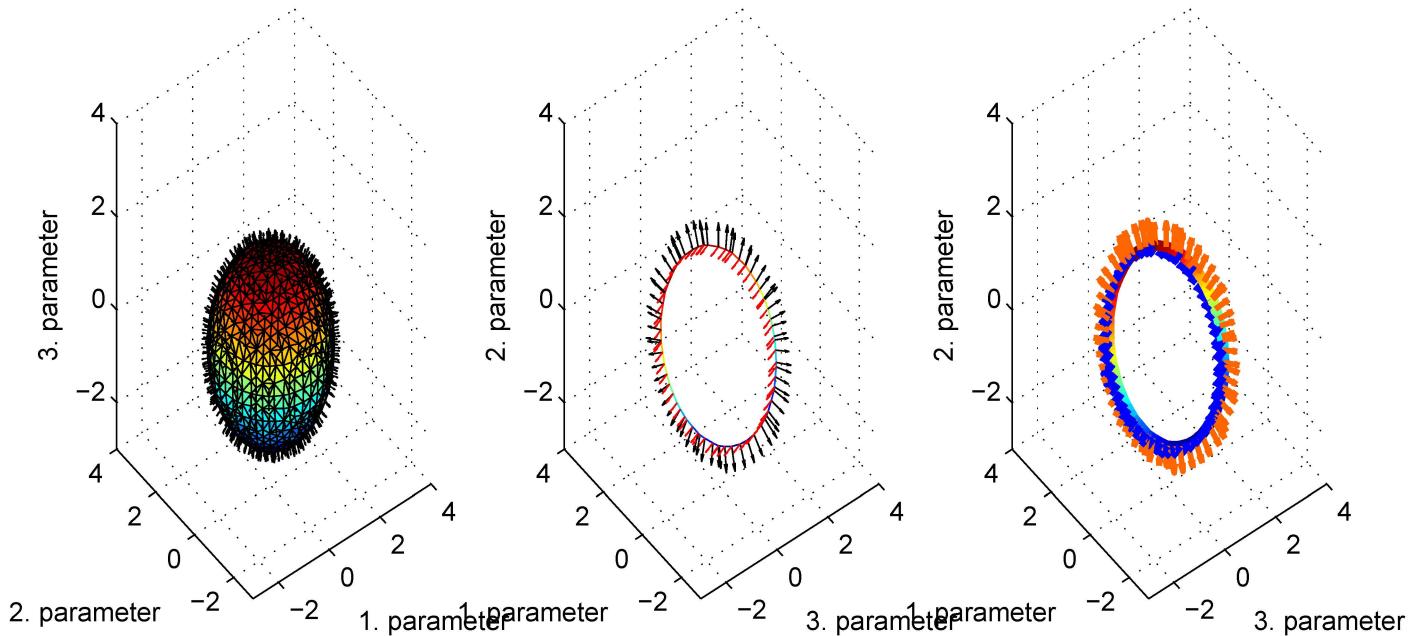


Figure 3.10: Plotting the gradient of the functions along the roots.

### 3.3 Complex example

In this Section, complex problems are solved to show, how the MDBM can be applied for complicated functions, special plotting problems, etc. The presented examples can be used as a base for new problems, you can simply modify the parameter range and the functions to suit your problem.

#### 3.3.1 Mandelbrot-set

```
run_test_complex_1_Mandelbrot_set.m  
fval_complex_1_Mandelbrot_set.m
```

This examples shows, how the MDBM can be used to analyse a complex, non-analytic function. The computation of the Mandelbrot-set is based on an iterative computation, and denotes the regions of the convergence.

Here the function is defined by an iteration

$$\begin{aligned} z_0 &= 0 \\ z_{i+1} &= z^2 + C \end{aligned} \tag{3.7}$$

where the complex number  $C = x + iy$ , where  $x$  and  $y$  are the two parameters.

We search for the boundary of convergence, so our function  $f$  is -1 if convergent (even in the last iteration  $z_i < 2$ ) and 1 if it is divergent. Note, that it is only an illustrative example, which presents fractal like results. The MDBM is not the best way to determine fractals, however it can be used to approximate the fractal dimension based on a box-counting dimension (see [1]).

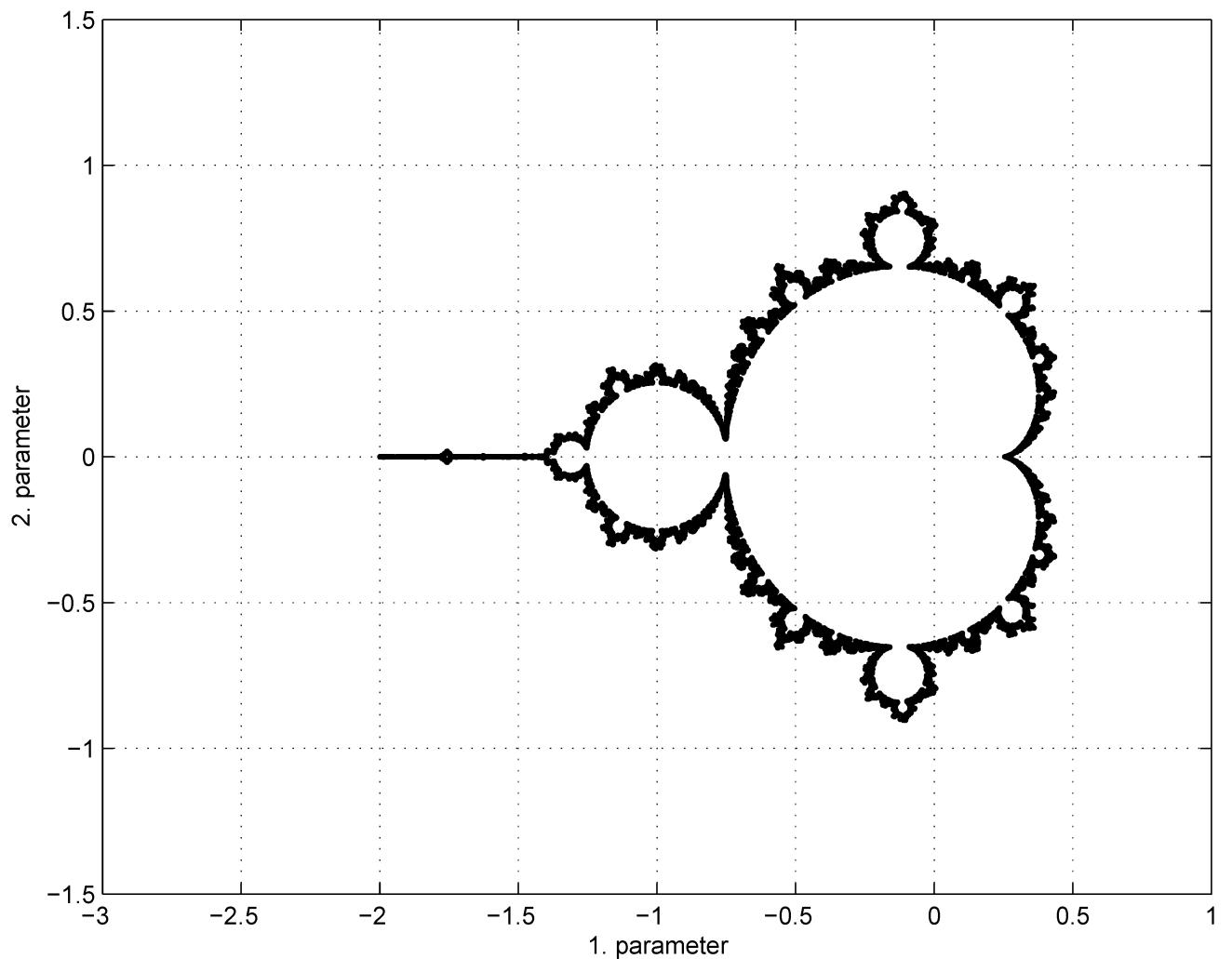


Figure 3.11: Mandelbrot set, roots of a complicated function which are based on an iteration.

### 3.3.2 Catastrophe surface

```
run_test_complex_2_catastrophe_surface.m
fval_complex_2_catastrophe_surface.m
```

These examples show, how to plot different sections of a complex solution and how to present them. See: [http://en.wikipedia.org/wiki/Catastrophe\\_theory](http://en.wikipedia.org/wiki/Catastrophe_theory)

The roots of the following equation are plotted:

$$f(a, b, x) = a \sin(x) + x + b; \quad (3.8)$$

Sections for constant  $a$  and  $b$  values are also computed, the critical points are determined based on a co-dimension 2 problem:

$$\begin{aligned} f_1(a, b, x) &= a \sin(x) + x + b \\ f_2(a, b, x) &= a \cos(x) + 1 \end{aligned} \quad (3.9)$$

Based on the content of the `par` variable, it can be decided which functions should be evaluated. The gradients of the functions values along the results are also plotted.

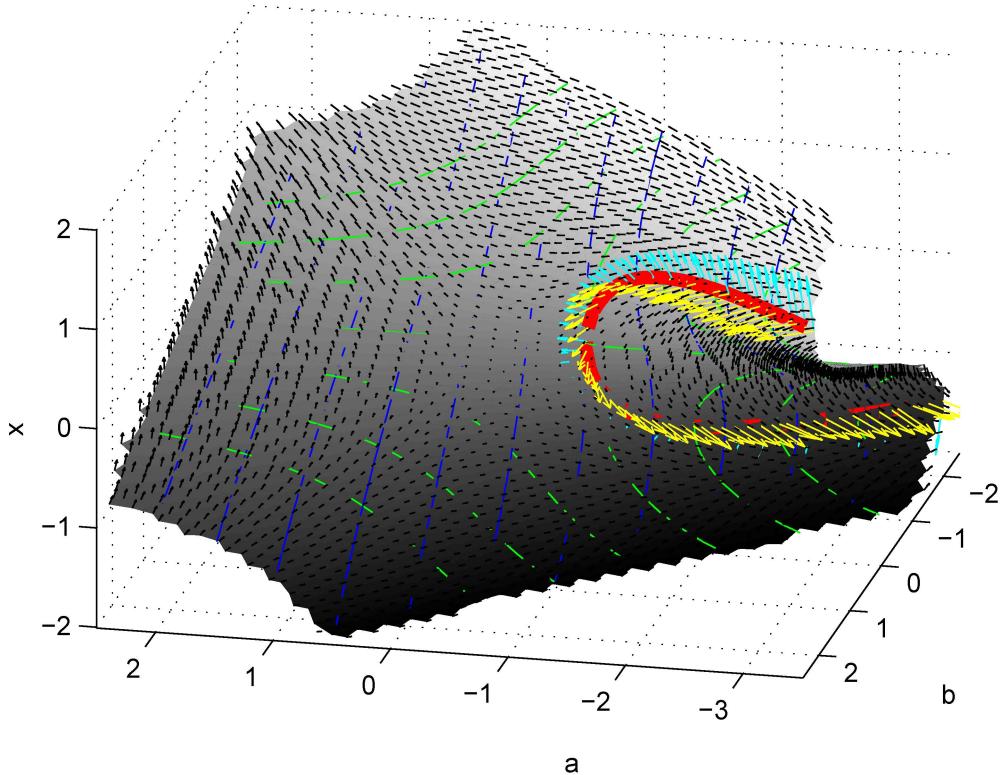


Figure 3.12: Catastrophe surface, and its sections. The critical values and the corresponding gradients are also computed.

### 3.3.3 Stability of the turning process with traditional process damping

```
run_test_complex_3_turning_stability.m
fval_complex_3_turning_stability.m
```

In this example the stability boundaries of the turning process with surface regeneration effect and with process damping are plotted. The corresponding governing equation is given as:

$$\ddot{x}(t) + (2\zeta + C \frac{2\pi}{\Omega})\dot{x}(t) + x(t) = w(x(t - \tau) - x(t)). \quad (3.10)$$

The corresponding characteristic equation is formed as:

$$D = \lambda^2 + (2\zeta + C \frac{2\pi}{\Omega})\lambda + 1 + w(1 - \exp(-\lambda\tau)) = 0. \quad (3.11)$$

In the stability boundary is at  $\lambda = i\omega$ . So we have a three parameter problem ( $D(w, \Omega, \omega)$ ), with two equations (co-dimension 2 problem) if the damping  $\zeta$  and the process damping coefficient  $C$  is considered as a constant parameter:

$$\begin{aligned} f_1(w, \Omega, \omega) &= \text{Re}(D(w, \Omega, \omega)) = 0 \\ f_2(w, \Omega, \omega) &= \text{Im}(D(w, \Omega, \omega)) = 0; \end{aligned} \quad (3.12)$$

In the numerical evaluation of the characteristic equation,  $\zeta$  and  $C$  are placed in the `par` struct. In the following figure the solution curves are plotted.

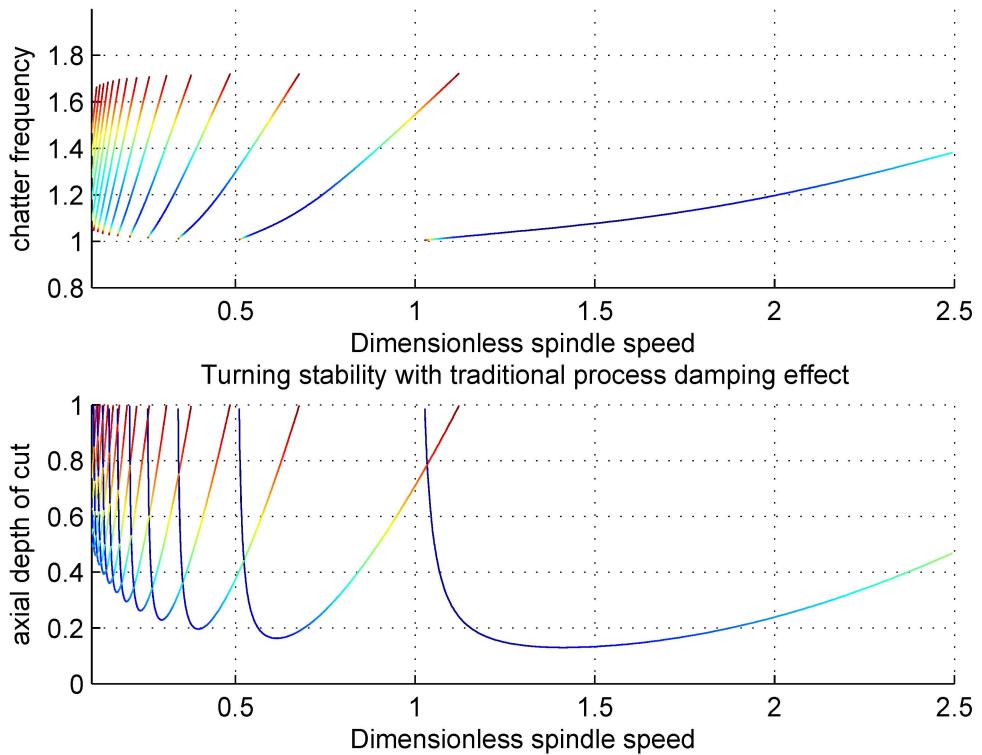


Figure 3.13: Stability of the turning process with traditional process damping.

### 3.3.4 Unite norm

```
run_test_complex_4_unit_circles.m
fval_complex_4_unit_circles.m
```

In this example, the 'unit circles' are computed in different norms and it is shown how to create special plotting.

The unit circle measured by Euclidean distance is given by:

$$f(x, y) = \|(x, y)\|_{L_2} = \sqrt{x^2 + y^2} = (x^2 + y^2)^{0.5}; \quad (3.13)$$

Different norms can be defined by:

$$f(x, y, p) = \sqrt[p]{x^p + y^p}; \quad (3.14)$$

During plotting, the graphical object handle (the output of the plotting function `db_plot_mdbm.m`) is used to modify the presentation.

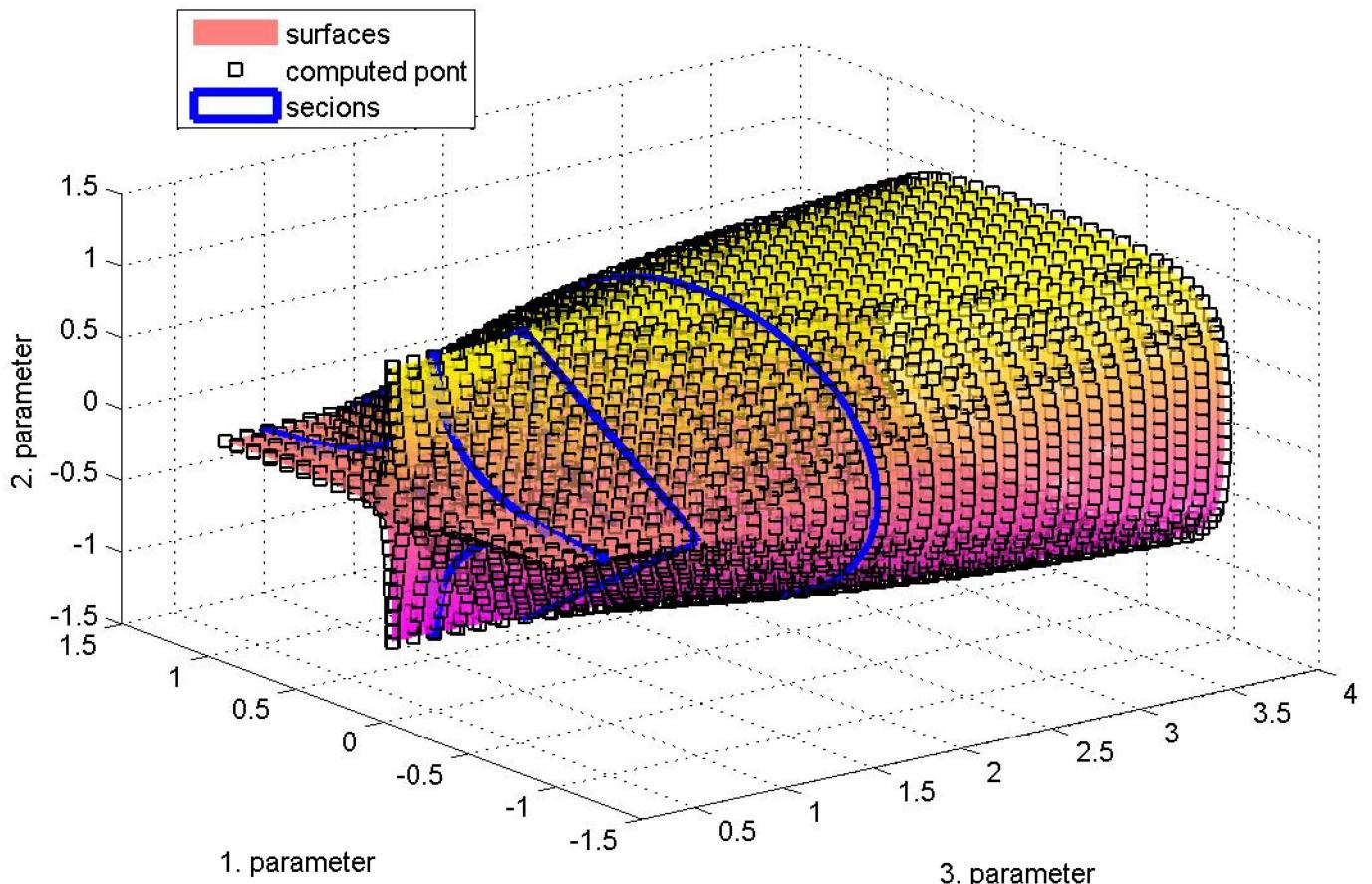


Figure 3.14: Unit 'circles' as a function of the norm value.

### 3.3.5 Section of surfaces, solution of transcendent equation, error calculation

```
run_test_complex_5_transcendent.m  
fval_complex_5_transcendent.m
```

We create special plottings of the roots of two transcendent equations and the common intersection points. We also analyse the error of the computation and we use the output of the MDBM as in input for a different optimisation technique.

The functions given as follows

$$\begin{aligned} f_1(x, y) &= \sin(2y + 0.5x^2) + 0.2x^2 + 0.1y^2 - 1 \\ f_2(x, y) &= \cos(3x) + \sin(2y) - 0.5 \end{aligned} \tag{3.15}$$

The effect of the number of iteration of the MDBM on the function error is analysed. The solution converges to the analytical solution, however, it should be mentioned that the MDBM can not really be used for large iteration numbers (`Niteration >~ 10`), because it can easily lead to memory problems. If high precision is required, the output of the MDBM should be used as an input of different algorithms like Newton's method.

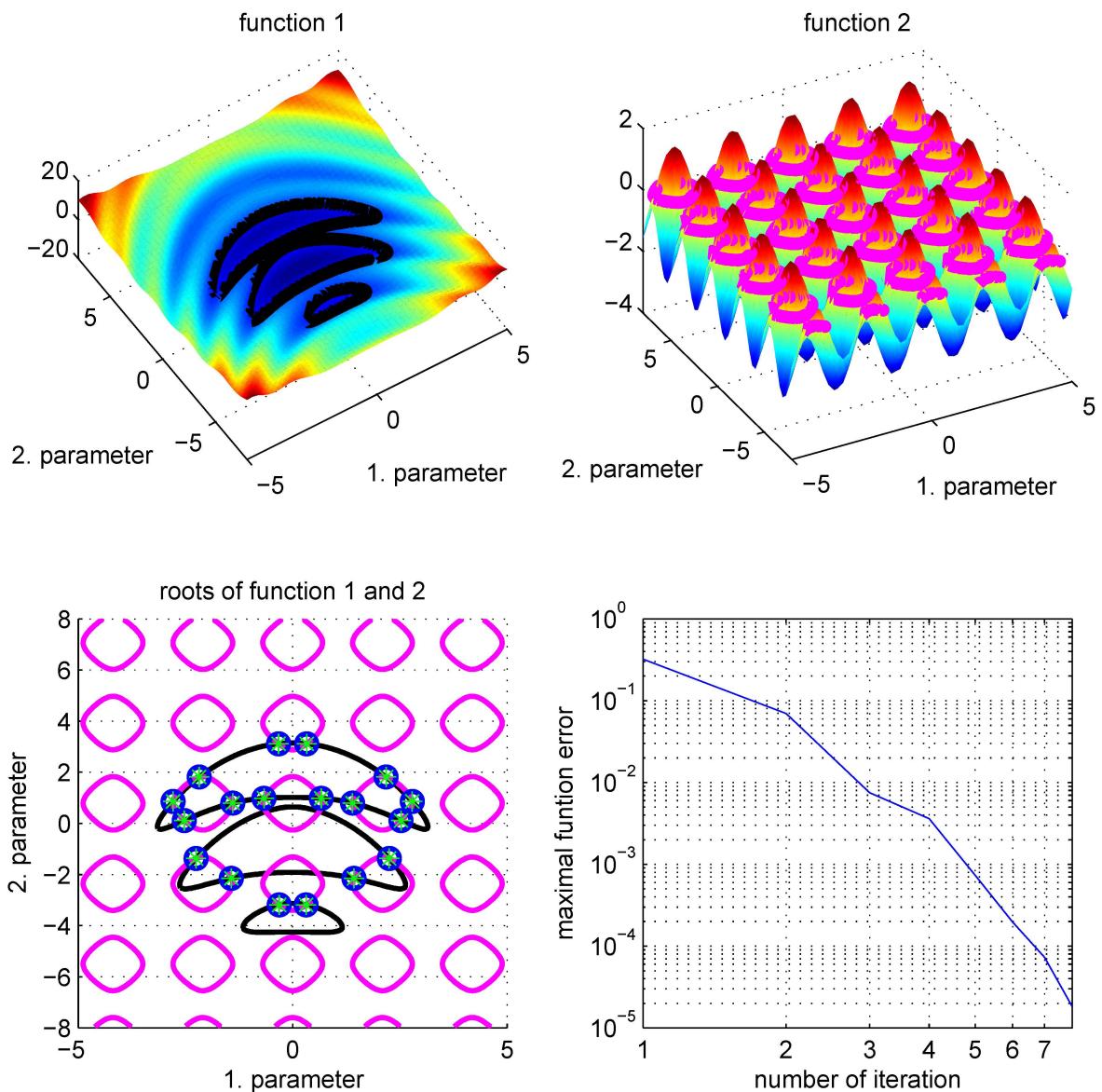


Figure 3.15: Roots of function 1 and 2, the common intersection points, and convergence of the solution as function of the iteration number

# Bibliography

- [1] Bachrathy, D., and Stepan, G., 2012. “Bisection method in higher dimensions and the efficiency number”. *Periodica Polytechnica, Mechanical Engineering*, **56**(2), pp. 81–86.