Exercises with 'A 99 line topology optimization code written in Matlab' *

Ole Sigmund
Department of Solid Mechanics, Building 404,
Technical University of Denmark
DK-2800 Lyngby, Denmark

May 27, 2009

1 Introduction

The following exercises are based on the paper *A 99 line topology optimization code written in Matlab* (Sigmund 2001). Both the source code and a preprint of the paper can be downloaded from the world wide web at `www.topopt.dtu.dk`. The exercises require some basic knowledge of mechanics, finite element analysis and optimization theory.

Try to solve at least problems 1-7 during the course. If you have previous experience or you are fast you may solve one or more of problems 8 through 13 in section 4.

The solutions must be presented in a poster session (consisting of a maximum of 16 A4 pages laid out on a table) and a copy of the poster and important source code(s) must be handed over to Jakob S. Jensen on tuesday June 16th.

The accompanying paper (Sigmund 2001) (see also the appendix in (Bendsøe and Sigmund 2004)) gives a lot of hints on how to solve problems 1 through 6.

The course web-site is www.topopt.dtu.dk/DCAMM/. From this web-site you may download the MMA subroutines and possibly other things during the course.

2 Getting started

- Form a group of preferably 2 students and get usernames and passwords from Jakob S. Jensen. Try to make sure that at least one group member has a mechanical engineering background and one has Matlab programming experience. Also try to hook up with somebody whom you don't know before hand.
- Login to a computer (Login group is "WIN")
- Start the internet browser and go to the TOPOPT web-page: `www.topopt.dtu.dk`
- Press button "Matlab program"
- Download the Matlab code `top.m` to your home directory "My Documents"
- Leave the browser and start up Matlab
- Run the default MBB-example by writing `top(60,20,0.5,3.0,1.1)` in the Matlab command line

*Intended for the DCAMM-course: *Topology Optimization - Theory, Methods and Applications* held at DTU, Lyngby, Denmark, June 10-16, 2009.

- Start experimenting with the code
- Keep an original version of the Matlab code and start editing new versions
- Extensions to the Matlab code such as a few lines which create a displacement picture and the equations for the strain-displacement matrix are given in appendices A and B and on the course web-page.

3 Recommended problems

The downloaded Matlab code solves the problem of optimizing the material distribution in the MBB-beam such that its compliance is minimized. A number of extensions and changes in the algorithm can be thought of.

Problem 1: Test influence of discretization, filter size and penalization power

Use the downloaded Matlab code to investigate the influence of the filter size `rmin`, the penalization power `penal` and the discretization (`nelx*nely`) on the topology of the MBB-beam (default example). Discuss the results.

Problem 2: Implement other boundary conditions

Change boundary and support conditions in order to solve the optimization problems sketched in Figure 1. Does the direction of the forces change the design?

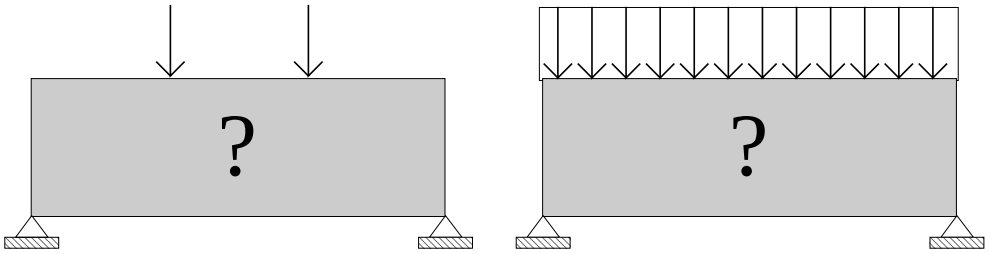

Figure 1: Topology optimization of a "bridge" structure. Left: two (simultaneous) point loads and right: distributed load.

Problem 3: Implement multiple load cases

Extend the algorithm such that it can solve the two-load case problem shown in Figure 2. Discuss the difference in topology compared to the one-load case solution from Figure 1(left).

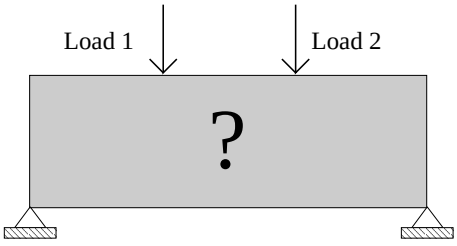

Figure 2: Topology optimization of a bridge structure with two load cases.

Problem 4: Implement passive elements

In some cases, some areas may be required to take the minimum density value (e.g. a hole for a pipe). Add the necessary lines to the program such that the problem shown in Figure 3 can be solved. What is the difference in compliance compared to Problem 2 (left)?

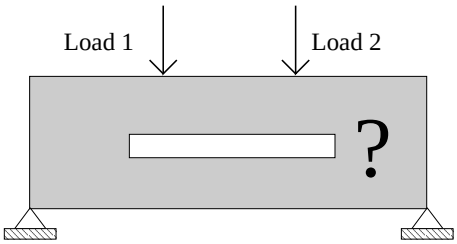

Figure 3: Topology optimization of a bridge structure with two load cases and a center area prescribed to be void.

Problem 5: Method of Moving Asymptotes (MMA)

Krister Svanberg from KTH in Stockholm Sweden has written an optimization routine called Method of Moving Asymptotes (MMA) (Svanberg 1987). Rewrite the topology optimization code such that it calls the MMA Matlab routine instead of the Optimality Criteria Solver. Use the developed software to optimize the default MBB-beam and possibly other examples. How does it compare to the OC solver?

The MMA routines mmasub.m and subsolv.m can be downloaded from the course web-page www.topopt.dtu.dk/DCAMM. The documentation for the Matlab MMA code is found in appendix C and in Svanbergs notes in the handouts. The program solves the following optimization problem

$$\left.\begin{array}{rcll} \min\limits_{\boldsymbol{x},\boldsymbol{y},z} & : & f_0(\boldsymbol{x}) + a_0\ z + \sum\limits_{i=1}^{m}(c_i y_i + \frac{1}{2} d_i y_i^2) & \\ \text{subject to} & : & f_i(\boldsymbol{x}) - a_i\ z - y_i \leq 0, \qquad i = 1,\ldots,m & \\ & : & x_j^{min} \leq x_j \leq x_j^{max}, \qquad j = 1,\ldots,n & \\ & : & y_i \geq 0, \qquad i = 1,\ldots,m & \\ & : & z \geq 0 & \end{array}\right\}, \tag{1}$$

where m and n are number of constraints and number of design variables respectively, $\mathbf{x}$ is the vector of design variables, $\mathbf{y}$ and z are positive optimization variables, f_0 is the objective function, $f_1,\ldots,f_m$ are the constraint functions ($f_i(\boldsymbol{x}) \leq 0$) and a_i, c_i and d_i are positive constants which can be used to determine the type of optimization problem.

If we want to solve the simpler problem

$$\left.\begin{array}{rcll} \min\limits_{\boldsymbol{x}} & : & f_0(\boldsymbol{x}) & \\ \text{subject to} & : & f_i(\boldsymbol{x}) \leq 0, \qquad i = 1,\ldots,m & \\ & : & x_j^{min} \leq x_j \leq x_j^{max}, \qquad j = 1,\ldots,n & \end{array}\right\}, \tag{2}$$

we must make sure that the optimization variables $\mathbf{y}$ and z from the original optimization problem (1) are equal to zero at optimum. This is obtained if we select the constants to the following values (suggested by Krister Svanberg)

$$a_0 = 1,\ a_i = 0,\ c_i = 1000,\ d = 0. \tag{3}$$

The call of the MMA routine requires the determination of sensitivities `df0dx`, `df0dx2`, `dfdx`, `dfdx2` corresponding to the derivatives $\frac{\partial f_0}{\partial x_j}$, $\frac{\partial^2 f_0}{\partial x_j^2}$, $\frac{\partial f_i}{\partial x_j}$ and $\frac{\partial^2 f_i}{\partial x_j^2}$, respectively. Since second derivatives are difficult to determine in topology optimization problems, we simply set them to zero.

The MMA call is

```
function [xmma,ymma,zmma,lam,xsi,eta,mu,zet,s,low,upp] = ...
    mmasub(m,n,iter,xval,xmin,xmax,xold1,xold2, ...
    f0val,df0dx,df0dx2,fval,dfdx,dfdx2,low,upp,a0,a,c,d);
```

Hints:

- Check appendix C or the `mmasub.m` file for explanations and definitions of the MMA variables
- Watch out for the difference between column and row vectors
- Remember to normalize constraints, i.e. instead of $V(\mathbf{x}) - V^* \leq 0$ use $V(\mathbf{x})/V^* - 1 \leq 0$
- In order to convert a Matlab matrix to a Matlab vector you may make use of the Matlab command `reshape` (type `help reshape` in the Matlab prompt to get help on `reshape`)

Problem 6: Mechanism synthesis

Extend the Matlab code to include compliant mechanism synthesis. Solve the inverter problem and possibly the gripper problem sketched in Figure 4.

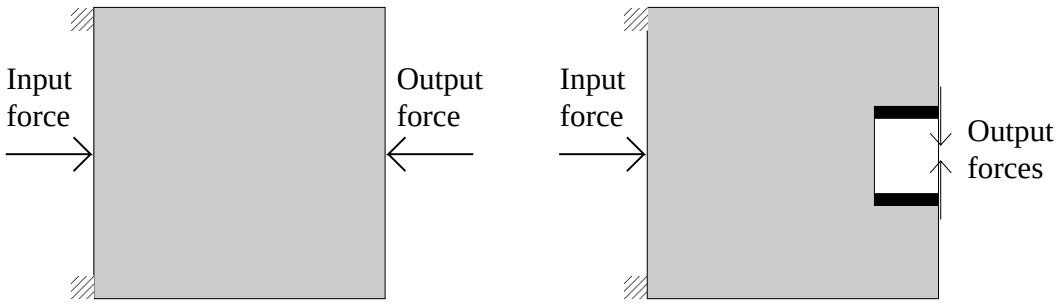

Figure 4: Synthesis of compliant mechanisms. Left: inverter example and Right: gripper example.

Using the formulation from the paper (Sigmund 1997), the goal of mechanism design is to maximize the work performed on a spring with a constraint on the input displacement. The optimization problem for mechanism synthesis is then

$$\left.\begin{aligned} \min_{\boldsymbol{x}} \;&:\; u_{out} \\ \text{subject to}:\; &V(\mathbf{x}) = \mathbf{v}^T\,\mathbf{x} \leq V^* \\ &:\; u_{in} \leq u_{in}^* \\ &:\; \mathbf{K}\mathbf{u} = \mathbf{f}_{in} \\ &:\; 0 < x_j^{min} \leq x_j \leq 1, \qquad j = 1,\ldots,n \end{aligned}\right\}, \tag{4}$$

Hints:

- The input displacement is found as $u_{in} = \mathbf{u}^T\mathbf{l}_{in}$ where $\mathbf{l}_{in}$ is a unit vector which has the value one at the input degree of freedom and is zero for all other degrees of freedom.

- The output displacement is found as $u_{out} = \mathbf{u}^T \mathbf{l}_{out}$ where $\mathbf{l}_{out}$ is a unit vector which has the value one at the output degree of freedom and is zero for all other degrees of freedom.
- Use the adjoint method to determine sensitivities
- The output spring is added to the analysis by adding an output spring stiffness k_s at the position of the output degree of freedom in the global stiffness matrix.
- Make use of symmetry
- Check the correctness of the derivatives of objective function and constraints by the finite difference method for a number of different elements. Remember to turn of the checkerboard filter during this test.
- In order to stabilize convergence you may change the values of `asyincr` and `asydecr` in the `mmasub.m` routine to 1.05 and 0.65, respectively (strategy for moving of asymptotes). You may also introduce external fixed movelimits.
- Start with Young's modulus E=100 and spring stiffnesses and input forces unity.
- Instead of the extra constraint on input displacement you may use a strain based input actuator, i.e. add a spring at the input point.

Problem 7: Optimization of dynamically loaded structures

Extend the Matlab code from Problem 5 to deal with *dynamic loads*. With a dynamic load of angular frequency ω, the basic equation to be solved is changed from $\mathbf{KU} = \mathbf{F}$ to:

$$\mathbf{SU} = \mathbf{F}, \tag{5}$$

where $\mathbf{S}$ is a system matrix (or dynamic stiffness matrix) defined as

$$\mathbf{S} = \mathbf{K} + i\omega\mathbf{C} - \omega^2\mathbf{M}, \tag{6}$$

where $\mathbf{M}$ is the global mass matrix and $\mathbf{C} = \alpha\mathbf{M}+\beta\mathbf{K}$ is the global mass- or stiffness-proportional damping matrix.

Optimize a 2 by 1 cantilever beam (supported on the left side and harmonically loaded at the center of the right side) for different values of ω using the following objective function:

$$c = u_{tip}\overline{u}_{tip} = \overline{\mathbf{U}}^T\mathbf{LU} \tag{7}$$

where $\mathbf{L}$ is a zero global matrix with a 1 in the diagonal corresponding to the output point (NB! Make sure to define $\mathbf{L}$ as a sparse matrix).

Hints:

- The mass matrix $\mathbf{M}$ is obtained by assembling the local mass matrices found in appendix B.
- Use the expression for the sensitivities found in (Jensen 2009).
- Use penalization factor 3 for stiffness and 1 for mass.
- Plot a frequency response curve for the initial structure. Try optimizing with different initial structures e.g. homogeneous structure or structure optimized for a static load or lower angular frequency.

- Start with small values of ω and gradually increase the value.
- Use the damping parameters α and β to stabilize the optimization. Recommended values are $E = 1$, $\rho = 1$, $\nu = 0.3$, $\alpha = 0.05$, $\beta = 0.002$, $\omega \in [0, 0.5]$ and volume fraction 0.5.
- Use external movelimits to stabilize convergence.

Voluntary extensions:

- Experiment with different penalization factors for mass and stiffness.
- Add a constraint on static compliance.

4 Voluntary Matlab problems

Problem 8: Min-Max formulation of multiple load cases

Minimize the maximum compliance of a three load case problem. This problem can be solved by fiddling with the constants in the MMA optimizer.

An optimization problem looking like

$$\left.\begin{array}{rll} \min\limits_{\boldsymbol{x}} : & \max\limits_{k=1,\ldots,p} \{|h_k(\boldsymbol{x})|\} & \\ \text{subject to} : & g_i(\boldsymbol{x}) \le 0, \qquad i = 1, \ldots, q & \\ : & x_j^{min} \le x_j \le x_j^{max}, \qquad j = 1, \ldots, n & \end{array}\right\}, \tag{8}$$

can be re-written to

$$\left.\begin{array}{rll} \min\limits_{\boldsymbol{x},\, z} : & z & \\ \text{subject to} : & h_i - z \le 0, \qquad i = 1, \ldots, p & \\ : & -h_i - z \le 0, \qquad i = 1, \ldots, p & \\ : & g_i(\boldsymbol{x}) \le 0, \qquad i = 1, \ldots, q & \\ : & x_j^{min} \le x_j \le x_j^{max}, \qquad j = 1, \ldots, n & \end{array}\right\}, \tag{9}$$

which may be solved by MMA using the constants

$$\begin{aligned} m &= 2p + q \\ f_0(\mathbf{x}) &= 0 \\ f_i(\mathbf{x}) &= h_i(\mathbf{x}), \qquad i = 1, \ldots, p \\ f_{p+i}(\mathbf{x}) &= -h_i(\mathbf{x}), \qquad i = 1, \ldots, p \\ f_{2p+i}(\mathbf{x}) &= g_i(\mathbf{x}), \qquad i = 1, \ldots, q \\ a_0 &= 1 \\ a_i &= 1, \qquad i = 1, \ldots, 2p \\ a_{2p+i} &= 0, \qquad i = 1, \ldots, q \\ d_i &= 0, \qquad i = 1, \ldots, m \\ c_i &= 1000, \qquad i = 1, \ldots, m \end{aligned} \tag{10}$$

Problem 9: Restriction methods

Implement the density filter or perimeter constraint instead of the sensitivity filter to ensure well-posedness of the topology optimization problem. The perimeter constraint may be applied as a penalty term to the objective function or as a separated constraint.

You may also try other restriction methods described in (Bendsøe and Sigmund 2004, Sigmund 2007).

Problem 10: Other optimization algorithms

Implement one of the Matlab Optimization Library's solvers (e.g. nonlinear programming (`fmincon`), quadratic programming (`quadprog`) or Sequential Linear Programming (using `linprog`). Do the alternative algorithms perform better or worse than OC or MMA?

Problem 11: Mechanisms with multiple outputs

Design an "elevator mechanism" as shown in Figure 5 (the platform must remain horizontal during elevation) or a gripping mechanism with parallel moving jaws.

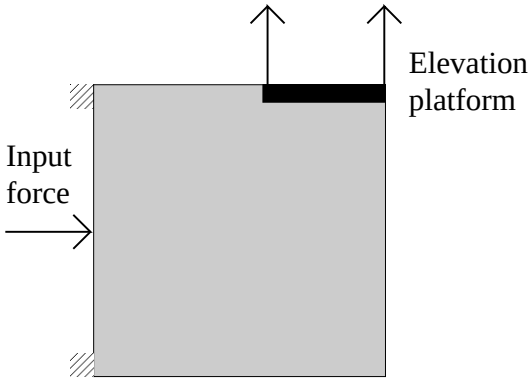

Figure 5: Mechanism synthesis for an "elevator mechanism". The platform must remain horizontal during elevation.

Problem 12: Three dimensions

How many lines of Matlab code does it take to optimize the topology of a three-dimensional MBB-beam? Implement the code and run some examples.

Problem 13: Others

Look in Bendsøe and Sigmund (2004) and get inspired to solve some other problems like thermal loads, conduction, selfweight etc.

References

Bendsøe, M. P. and Sigmund, O.: 2004, *Topology Optimization - Theory, Methods and Applications*, Springer Verlag, Berlin Heidelberg.

Jensen, J. S.: 2009, A note on sensitivity analysis of linear dynamic systems with harmonic excitation, *Report*, Department of Mechanical Engineering, Technical University of Denmark.

Sigmund, O.: 1997, On the design of compliant mechanisms using topology optimization, *Mechanics of Structures and Machines* **25**(4), 493–524.

Sigmund, O.: 2001, A 99 line topology optimization code written in MATLAB, *Structural and Multidisciplinary Optimization* **21**, 120–127. MATLAB code available online at: www.topopt.dtu.dk.

Sigmund, O.: 2007, Morphology-based black and white filters for topology optimization, *Structural and Multidisciplinary Optimization* **33**(4-5), 401–424.

Svanberg, K.: 1987, The Method of Moving Asymptotes - A new method for structural optimization, *International Journal for Numerical Methods in Engineering* **24**, 359–373.

A Appendix: MATLAB extension for plotting displacements

To obtain plots with displacements exchange the line

```
  colormap(gray); imagesc(-x); axis equal; axis tight; axis off;pause(1e-6);
```

with the lines

```
%  colormap(gray); imagesc(-x); axis equal; axis tight; axis off;pause(1e-6);
  colormap(gray); axis equal;
  for ely = 1:nely
    for elx = 1:nelx
      n1 = (nely+1)*(elx-1)+ely;
      n2 = (nely+1)* elx   +ely;
      Ue = 0.005*U([2*n1-1;2*n1; 2*n2-1;2*n2; 2*n2+1;2*n2+2; 2*n1+1;2*n1+2],1);
      ly = ely-1; lx = elx-1;
      xx = [Ue(1,1)+lx Ue(3,1)+lx+1 Ue(5,1)+lx+1 Ue(7,1)+lx  ]';
      yy = [-Ue(2,1)-ly -Ue(4,1)-ly   -Ue(6,1)-ly-1 -Ue(8,1)-ly-1]';
      patch(xx,yy,-x(ely,elx))
    end
  end
  drawnow; clf;
```

Note that the factor 0.005 is a scaling factor that may be freely chosen.

B Appendix: MATLAB mass and strain-displacement matrices for 4-node element

The strain displacement matrix ($\varepsilon = \mathbf{B}\mathbf{u_e}$)

```
bmat = [-1/2  0    1/2  0   1/2 0   -1/2  0
          0   -1/2  0   -1/2 0   1/2 0     1/2
         -1/2 -1/2 -1/2  1/2 1/2 1/2 1/2  -1/2];
```

and the mass matrix (with total mass equal to unity)

```
m0 = [4/9 0 2/9 0 1/9 0 2/9 0
      0 4/9 0 2/9 0 1/9 0 2/9
      2/9 0 4/9 0 2/9 0 1/9 0
      0 2/9 0 4/9 0 2/9 0 1/9
      1/9 0 2/9 0 4/9 0 2/9 0
      0 1/9 0 2/9 0 4/9 0 2/9
      2/9 0 1/9 0 2/9 0 4/9 0
      0 2/9 0 1/9 0 2/9 0 4/9]/(4*nel);
```

and the constitutive matrix for plane stress

```
Emat = E/(1-nu^2)*[ 1 nu 0
                    nu 1 0
                    0  0 (1-nu)/2];
```

C Appendix: MMA Matlab documentation

You may download the MMA-code from the web-page www.topopt.dtu.dk/DCAMM/

```
%    This is the file mmasub.m
%
function [xmma,ymma,zmma,lam,xsi,eta,mu,zet,s,low,upp] = ...
mmasub(m,n,iter,xval,xmin,xmax,xold1,xold2, ...
f0val,df0dx,df0dx2,fval,dfdx,dfdx2,low,upp,a0,a,c,d);
%
%    Written in May 1999 by
%    Krister Svanberg <krille@math.kth.se>
%    Department of Mathematics
%    SE-10044 Stockholm, Sweden.
%
%    Modified ("spdiags" instead of "diag") April 2002
%
%
%    This function mmasub performs one MMA-iteration, aimed at
%    solving the nonlinear programming problem:
%
%      Minimize  f_0(x) + a_0*z + sum( c_i*y_i + 0.5*d_i*(y_i)^2 )
%    subject to  f_i(x) - a_i*z - y_i <= 0,  i = 1,...,m
%                xmin_j <= x_j <= xmax_j,    j = 1,...,n
%                z >= 0,   y_i >= 0,         i = 1,...,m
%*** INPUT:
%
%   m    = The number of general constraints.
%   n    = The number of variables x_j.
%  iter  = Current iteration number ( =1 the first time mmasub is called).
%  xval  = Column vector with the current values of the variables x_j.
%  xmin  = Column vector with the lower bounds for the variables x_j.
%  xmax  = Column vector with the upper bounds for the variables x_j.
%  xold1 = xval, one iteration ago (provided that iter>1).
%  xold2 = xval, two iterations ago (provided that iter>2).
%  f0val = The value of the objective function f_0 at xval.
%  df0dx = Column vector with the derivatives of the objective function
%          f_0 with respect to the variables x_j, calculated at xval.
% df0dx2 = Column vector with the non-mixed second derivatives of the
%          objective function f_0 with respect to the variables x_j,
%          calculated at xval. df0dx2(j) = the second derivative
%          of f_0 with respect to x_j (twice).
%          Important note: If second derivatives are not available,
%          simply let df0dx2 = 0*df0dx.
%  fval  = Column vector with the values of the constraint functions f_i,
%          calculated at xval.
%  dfdx  = (m x n)-matrix with the derivatives of the constraint functions
%          f_i with respect to the variables x_j, calculated at xval.
%          dfdx(i,j) = the derivative of f_i with respect to x_j.
%  dfdx2 = (m x n)-matrix with the non-mixed second derivatives of the
```

```
%          constraint functions f_i with respect to the variables x_j,
%          calculated at xval. dfdx2(i,j) = the second derivative
%          of f_i with respect to x_j (twice).
%          Important note: If second derivatives are not available,
%          simply let dfdx2 = 0*dfdx.
%  low   = Column vector with the lower asymptotes from the previous
%          iteration (provided that iter>1).
%  upp   = Column vector with the upper asymptotes from the previous
%          iteration (provided that iter>1).
%  a0    = The constants a_0 in the term a_0*z.
%  a     = Column vector with the constants a_i in the terms a_i*z.
%  c     = Column vector with the constants c_i in the terms c_i*y_i.
%  d     = Column vector with the constants d_i in the terms 0.5*d_i*(y_i)^2.
%
%*** OUTPUT:
%
%  xmma  = Column vector with the optimal values of the variables x_j
%          in the current MMA subproblem.
%  ymma  = Column vector with the optimal values of the variables y_i
%          in the current MMA subproblem.
%  zmma  = Scalar with the optimal value of the variable z
%          in the current MMA subproblem.
%  lam   = Lagrange multipliers for the m general MMA constraints.
%  xsi   = Lagrange multipliers for the n constraints alfa_j - x_j <= 0.
%  eta   = Lagrange multipliers for the n constraints x_j - beta_j <= 0.
%   mu   = Lagrange multipliers for the m constraints -y_i <= 0.
%  zet   = Lagrange multiplier for the single constraint -z <= 0.
%   s    = Slack variables for the m general MMA constraints.
%  low   = Column vector with the lower asymptotes, calculated and used
%          in the current MMA subproblem.
%  upp   = Column vector with the upper asymptotes, calculated and used
%          in the current MMA subproblem.
%
 epsimin = sqrt(m+n)*10^(-9);
 feps = 0.000001;
 asyinit = 0.5;
 asyincr = 1.2;
 asydecr = 0.7;
```